Lecture: Convolutional Neural Networks

Juan Carlos Niebles and Ranjay Krishna

Stanford Vision and Learning Lab

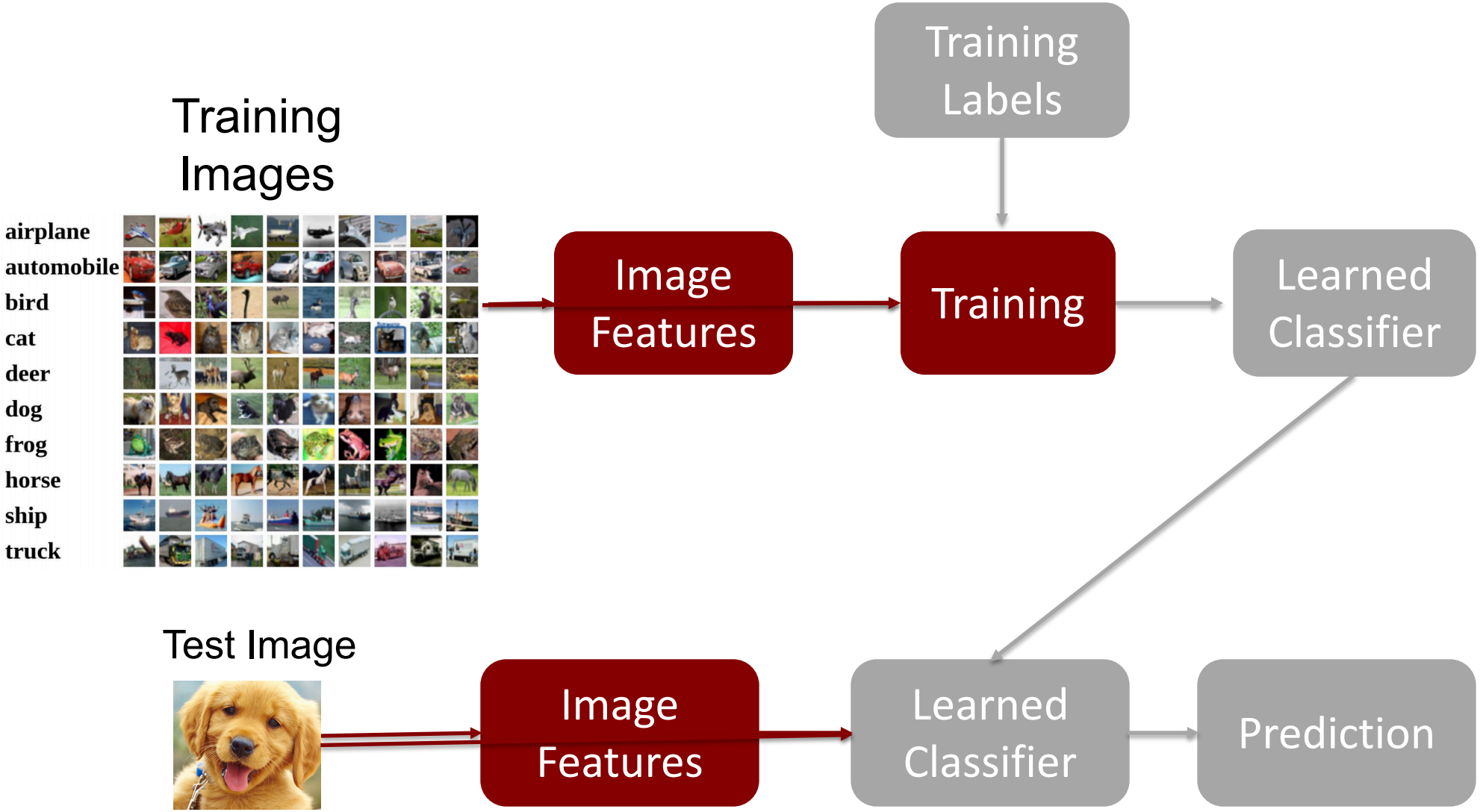
What should you take away from this class?

Pixels	Segments	Images	Videos	Web
Convolutions Edges Descriptors	Resizing Segmentation Clustering	Recognition Detection Ontology	Motion Tracking	Neural networks Convolutional neural networks

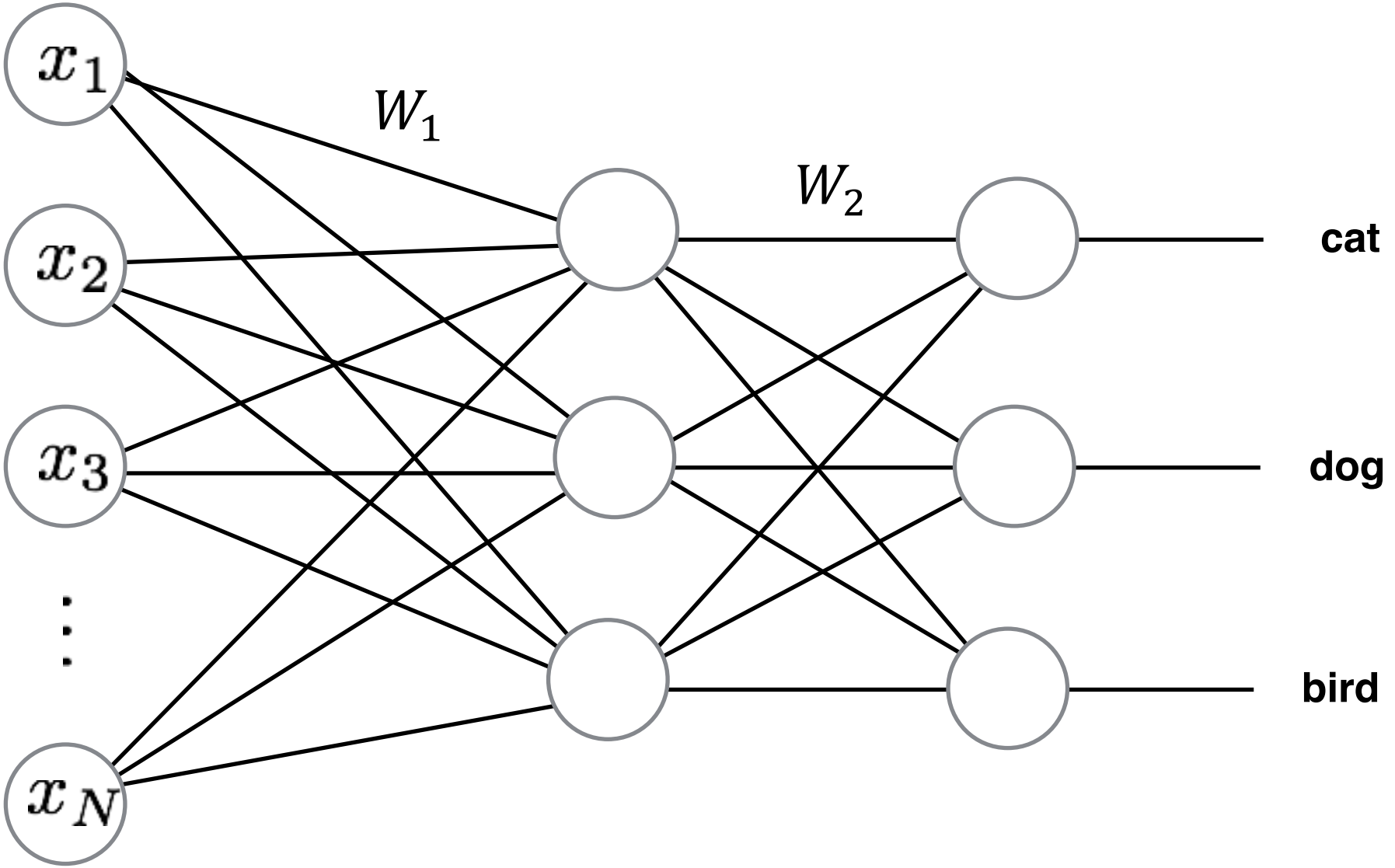
Today's agenda

- Backprop in neural networks
- Convolutional neural networks
- Architecture design
- Exam and brief class overview
- En fin

Recall: image classification pipeline



Let's change the features by adding another layer



Recall: 2 or 3-layer network

- linear classifier:

$$y = Wx$$

- 2-layer network:

$$y = W_2 \max(0, W_1 x)$$

- 3-layer network:

$$y = W_3 \max(0, W_2 \max(0, W_1 x))$$

Today's agenda

- Backprop in neural networks
- Convolutional neural networks
- Architecture design
- Exam and brief class overview
- En fin

Recall: Gradient Descent Pseudocode

```
for i in {0,…,num_epochs}:
    for x, y in data:
```

$$\hat{y} = f(x, W)$$

$$L = \sum_i L_i\left(y_i, \hat{y}_i\right)$$

$$\frac{dL}{dW} = ???$$

$$W := W - \alpha \frac{dL}{dW}$$

Backprop using calculus

$$\begin{aligned} Loss &= \mathrm{L}(\hat{\mathrm{y}}, \mathrm{y}) \\ &= L(f(x, W), y) \\ &= L(W_3 \ \max(0, W_2 \max(0, W_1 x)), y) \end{aligned}$$

$$\frac{\mathrm{dL}}{\mathrm{dW}} = \nabla_W L(W_3 \ \max(0, W_2 \max(0, W_1 x)), y)$$

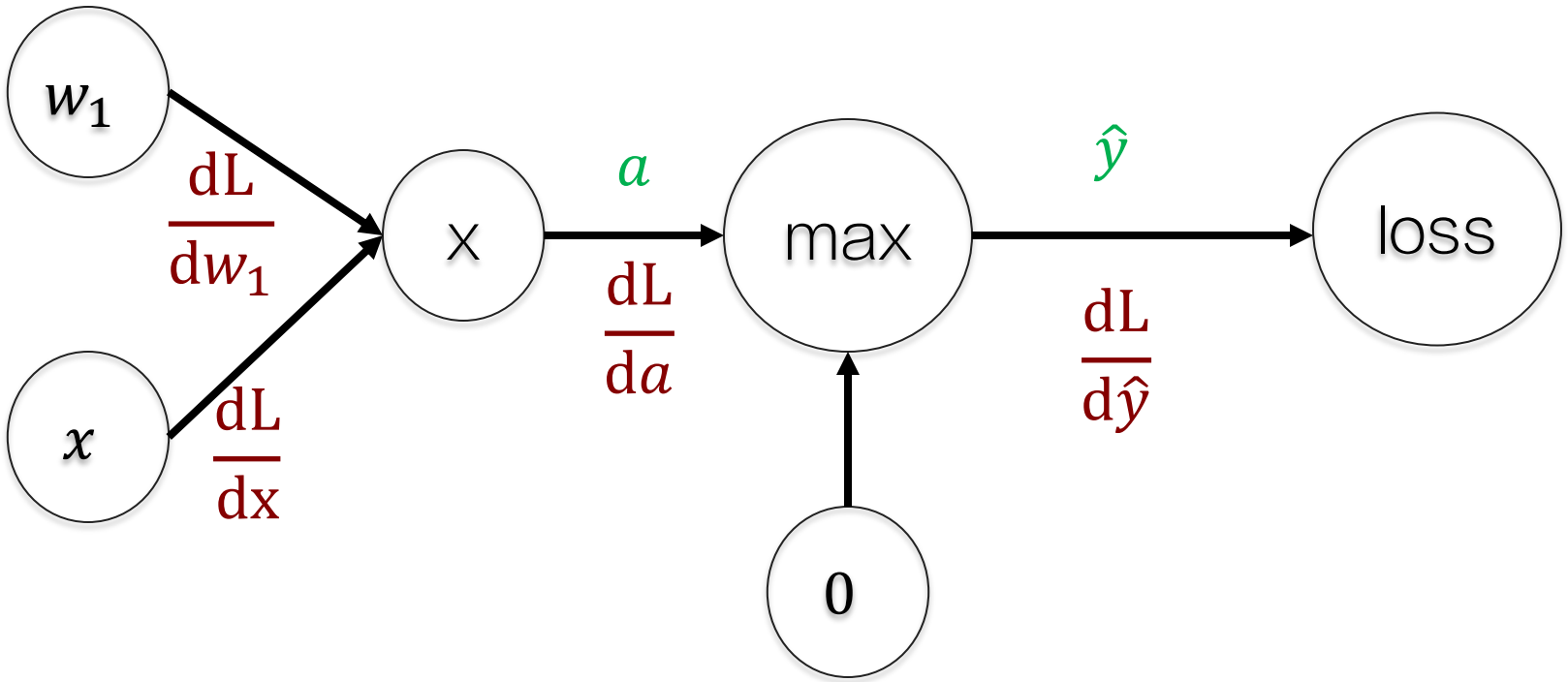
which can get messy really fast. And these are small 3-layer neural networks.

But remember, we learned about the chain rule?

$$\frac{\mathrm{dL}}{\mathrm{dW}} = \frac{\mathrm{dL}}{\mathrm{d\hat{y}}} \frac{\mathrm{d\hat{y}}}{\mathrm{dW}}$$

Generalizing the chain rule for neural networks

- Let's start with an easy 1D example:



$$a = wx$$

$$\hat{y} = \max(0, a)$$

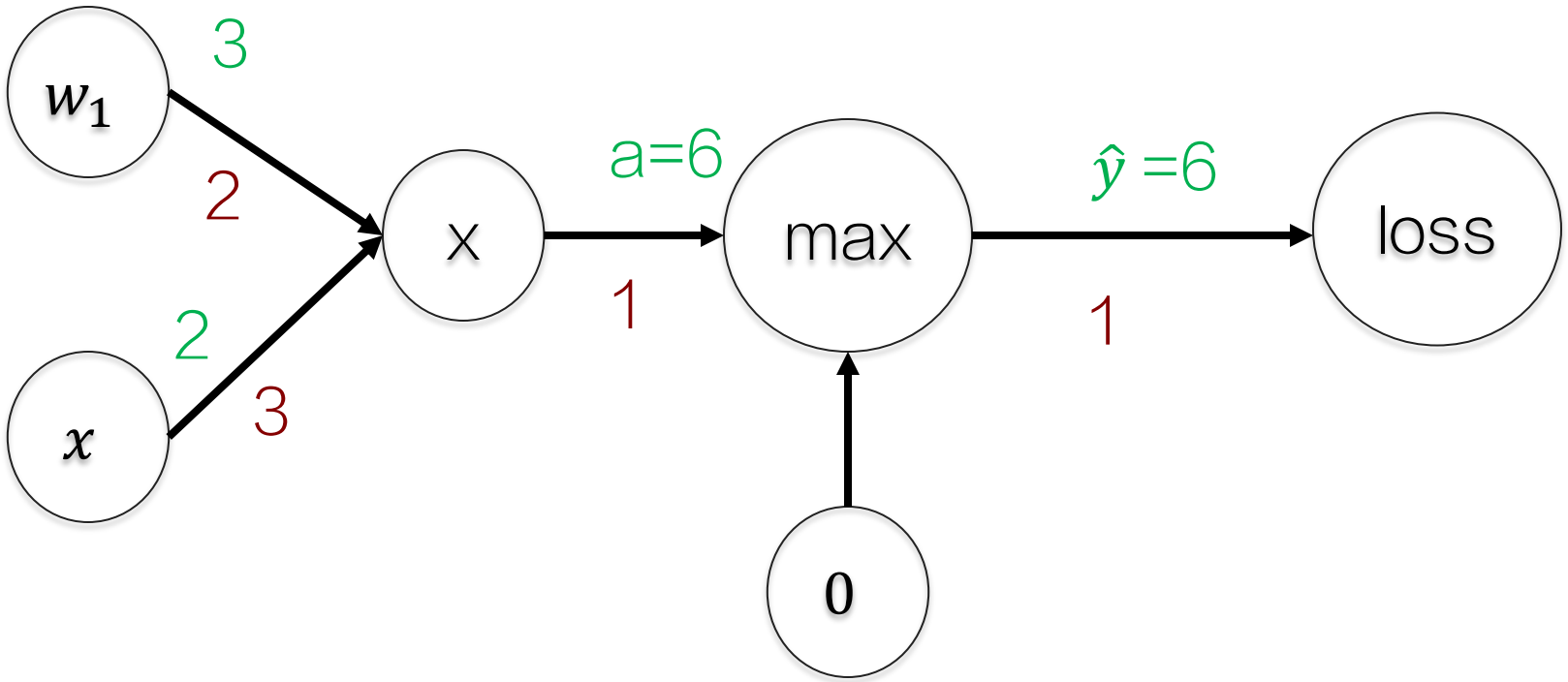
$$\frac{dL}{da} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{da}$$

$$\frac{dL}{dw_1} = \frac{dL}{da} \frac{da}{dw_1}$$

$$\frac{dL}{dx} = \frac{dL}{da} \frac{da}{dx}$$

At the end, we want to calculate $\frac{dL}{dW}$

Generalizing the chain rule for neural networks



$$a = wx$$

$$\hat{y} = \max(0, a)$$

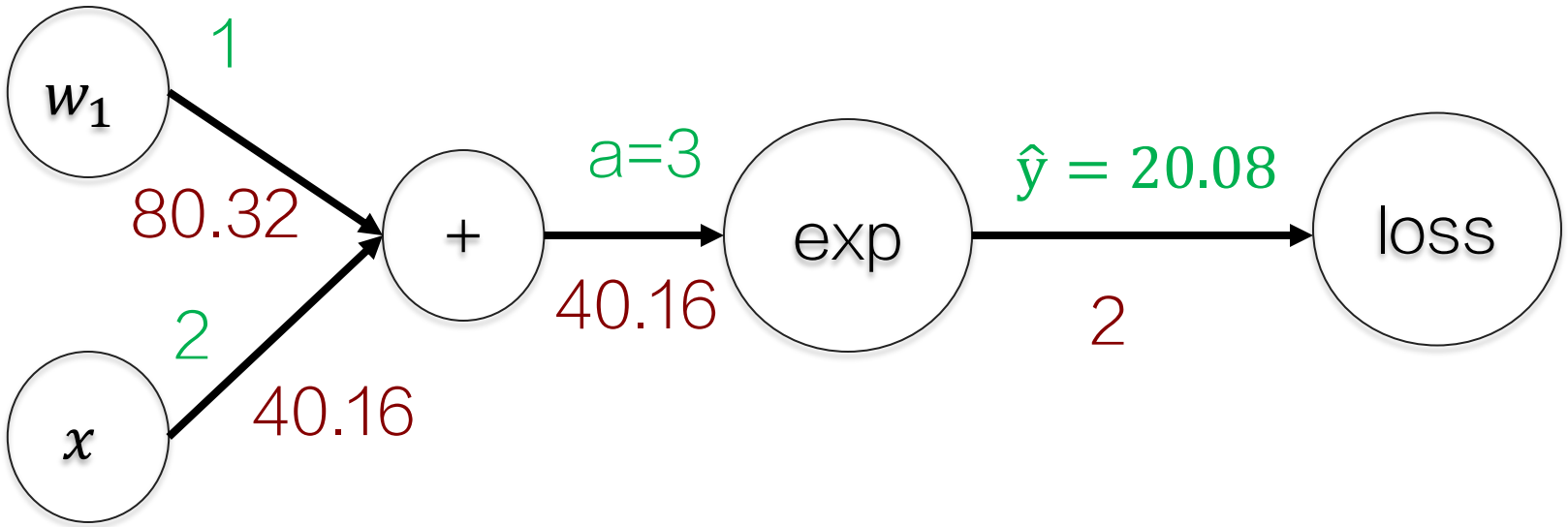
$$\frac{dL}{da} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{da}$$

$$\frac{dL}{dw_1} = \frac{dL}{da} \frac{da}{dw_1}$$

$$\frac{dL}{dx} = \frac{dL}{da} \frac{da}{dx}$$

- Let's assume $x = 2, w_1 = 3$
- Let's assume the loss is 1

Let's try another harder example



$$a = w + x$$

$$\hat{y} = e^{a}$$

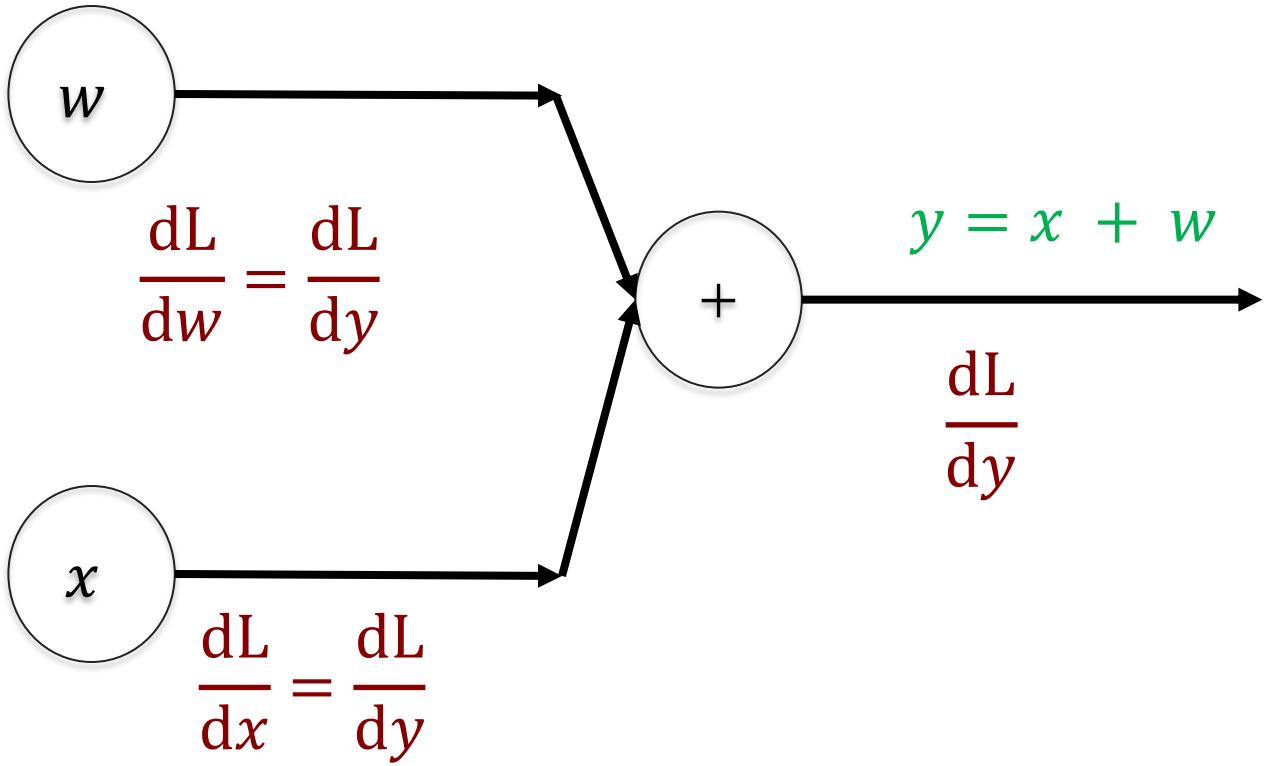
$$\frac{dL}{da} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{da}$$

$$\frac{dL}{dw_1} = \frac{dL}{da} \frac{da}{dw_1}$$

$$\frac{dL}{dx} = \frac{dL}{da} \frac{da}{dx}$$

General rules for calculating gradients

- Addition: gradient distributor

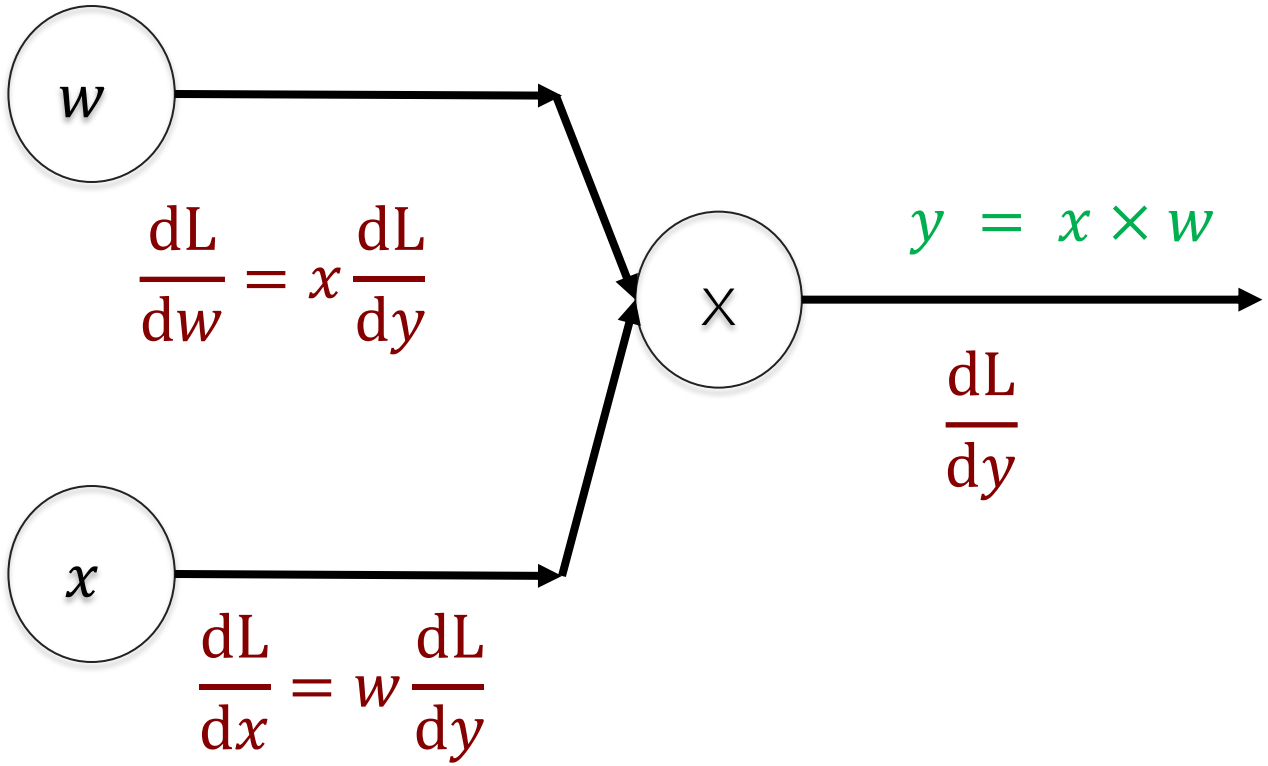


$$\frac{dy}{dx} = 1$$

$$\frac{dy}{dw} = 1$$

General rules for calculating gradients

- Multiplication: gradient swap multiplier

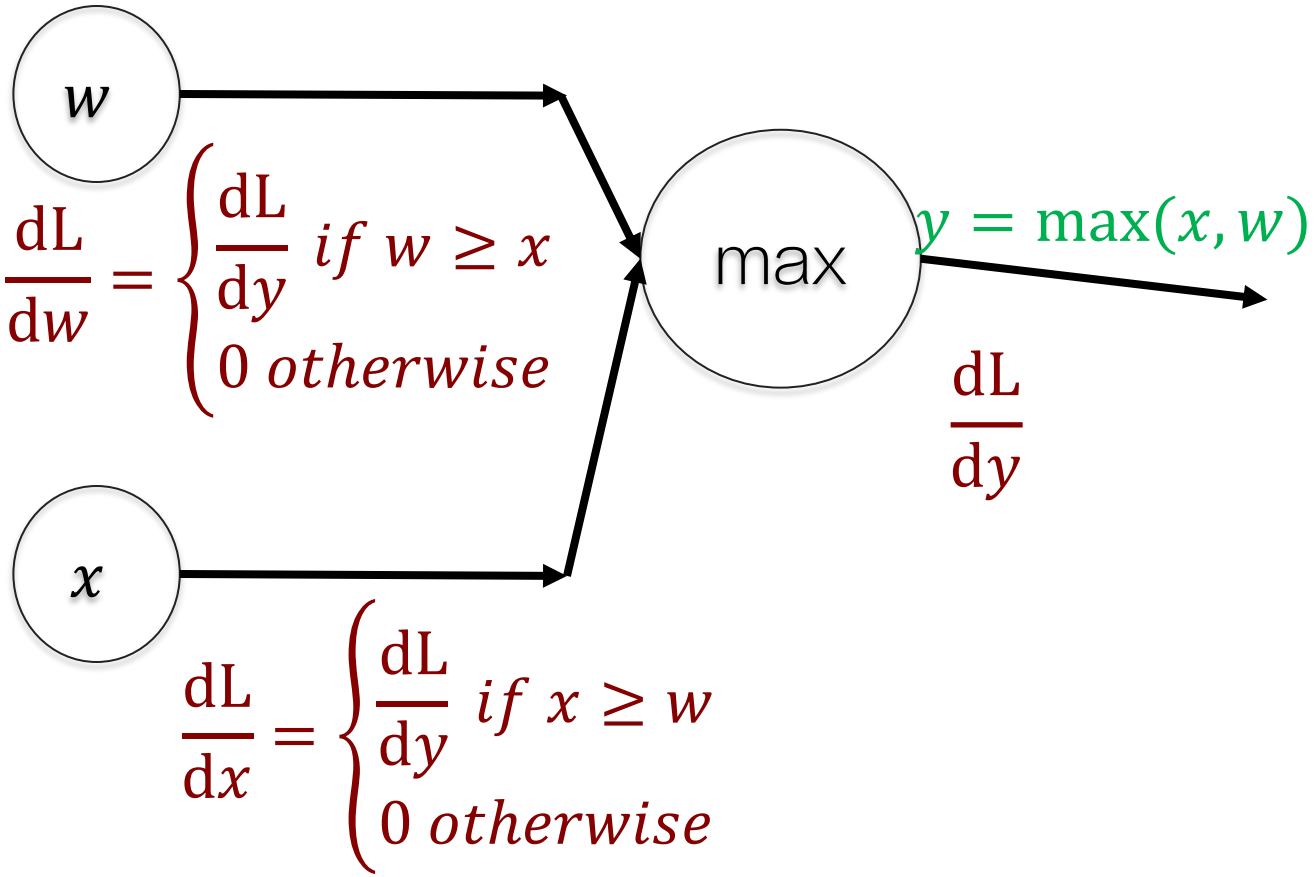


$$\frac{dy}{dx} = w$$

$$\frac{dy}{dw} = x$$

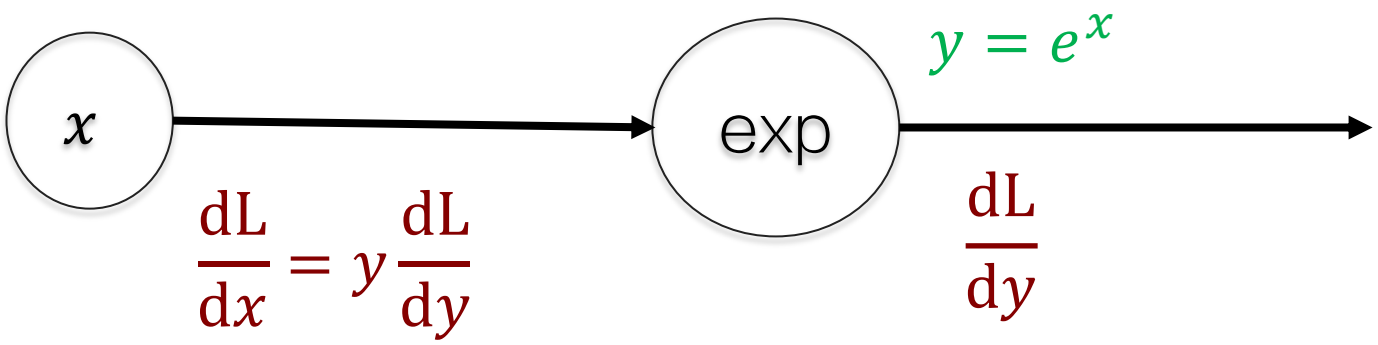
General rules for calculating gradients

- ReLU or max(0, ⋯): gradient multiplexer



General rules for calculating gradients

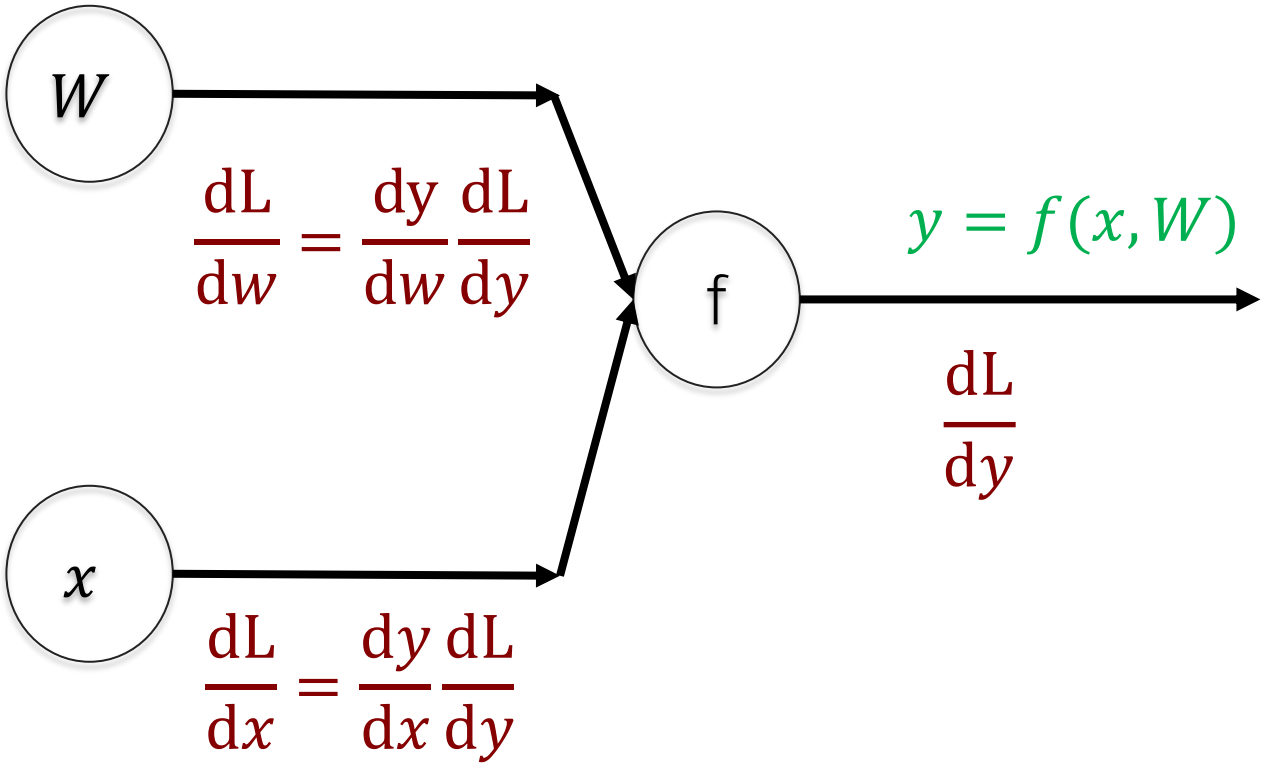
- Exponentiation: gradient output multiplier



$$\frac{dy}{dx} = e^x$$

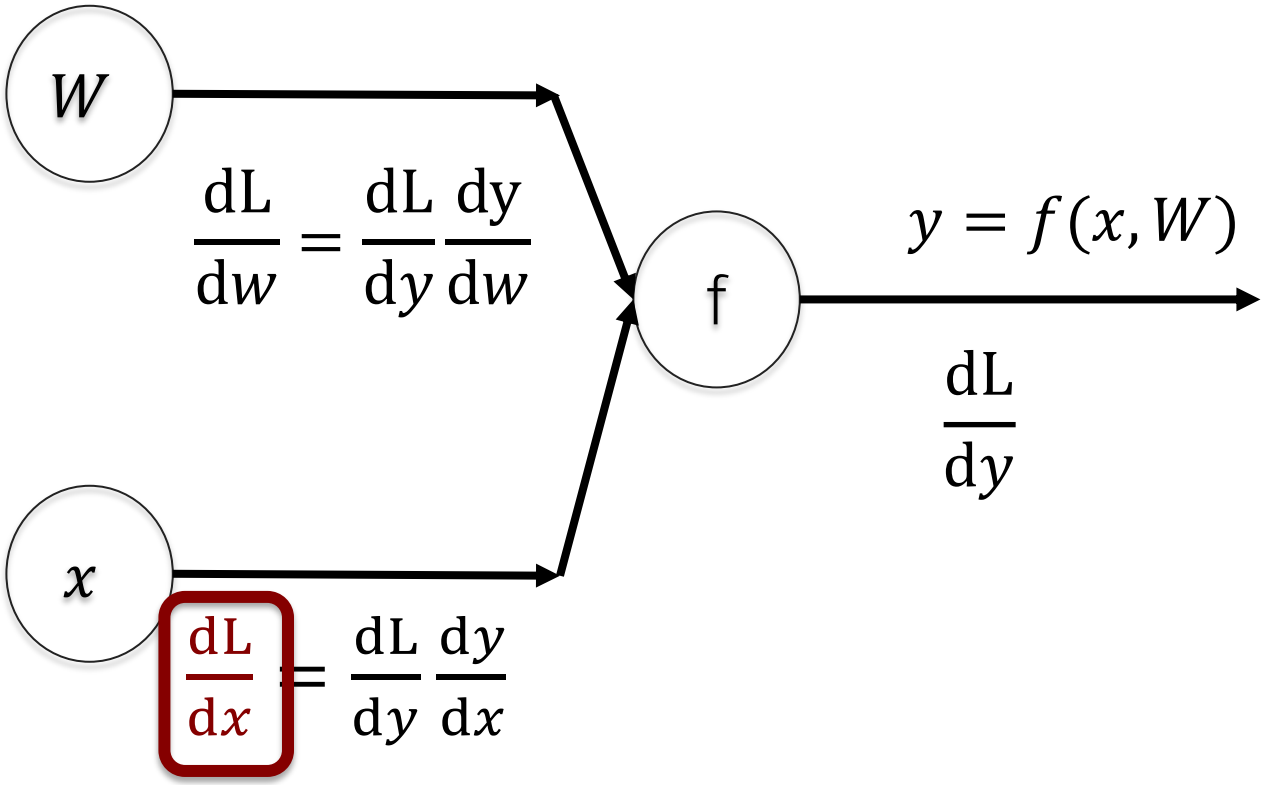
Backprop with gradients

- Remember that the dimensions of a variable and its gradients have to be the same.
- Dimensions of x and $\frac{dL}{dx}$ are the same



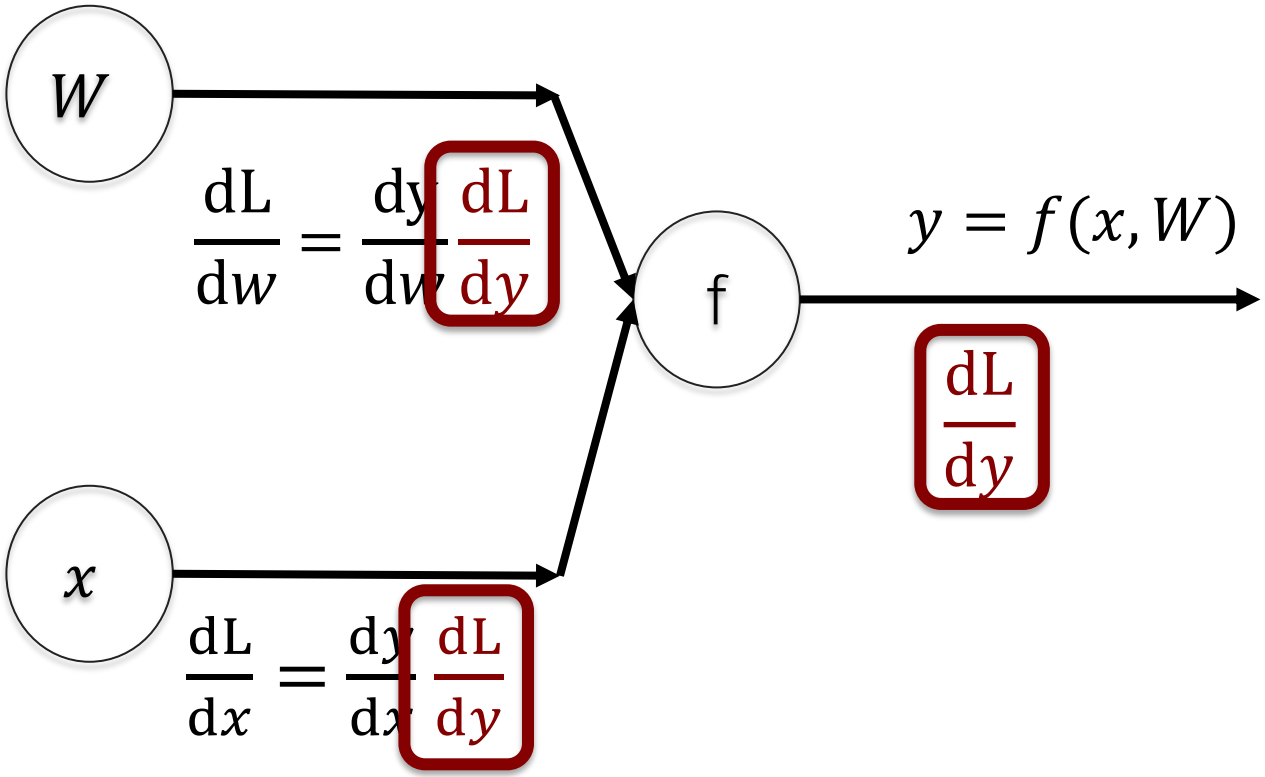
Backprop with gradients

- If $x \in \mathbb{R}^M$, then $\frac{\mathrm{dL}}{\mathrm{d}x} \in \mathbb{R}^M$



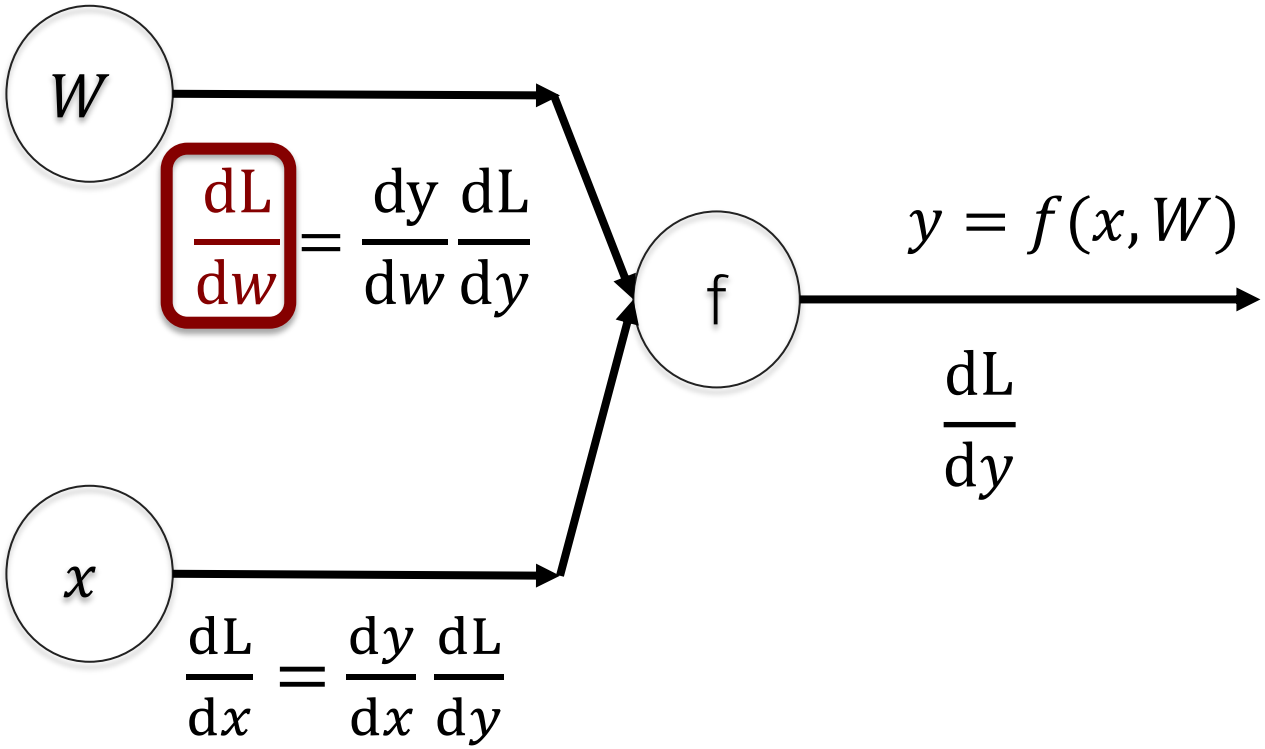
Backprop with gradients

- If $x \in \mathbb{R}^M$, then $\frac{\mathrm{dL}}{\mathrm{d}x} \in \mathbb{R}^M$
- Similarly, if $y \in \mathbb{R}^N$, then $\frac{\mathrm{dL}}{\mathrm{d}y} \in \mathbb{R}^N$



Backprop with gradients

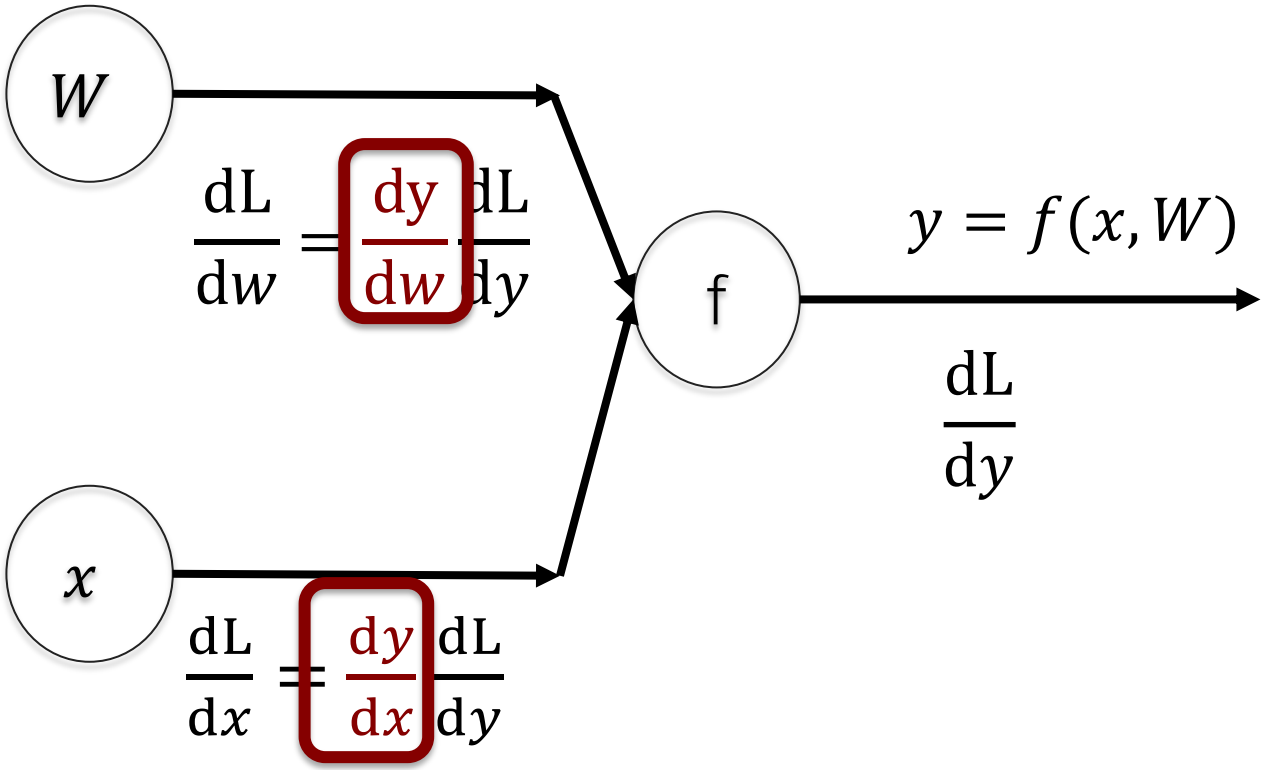
- If $x \in \mathbb{R}^M$, then $\frac{dL}{dx} \in \mathbb{R}^M$
- Similarly, if $y \in \mathbb{R}^N$, then $\frac{dL}{dy} \in \mathbb{R}^N$
- So, $W \in \mathbb{R}^{N \times M}$, then $\frac{dL}{dW} \in \mathbb{R}^{N \times M}$



Backprop with gradients

- If $x \in \mathbb{R}^M$, then $\frac{\mathrm{dL}}{\mathrm{d}x} \in \mathbb{R}^M$
- Similarly, if $y \in \mathbb{R}^N$, then $\frac{\mathrm{dL}}{\mathrm{d}y} \in \mathbb{R}^N$

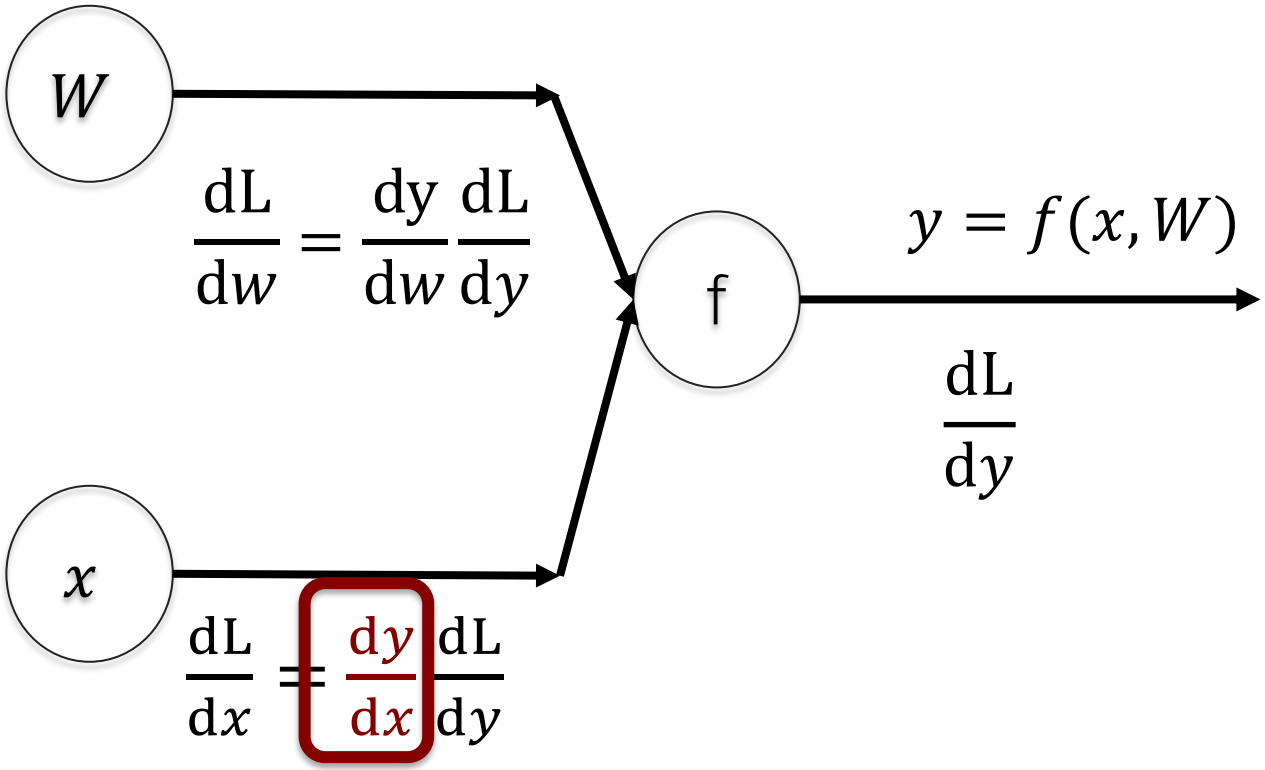
- So, $\mathrm{W} \in \mathbb{R}^{N \times M}$, then $\frac{\mathrm{dL}}{\mathrm{d}W} \in \mathbb{R}^{N \times M}$
- So, what are the dimensions of $\frac{\mathrm{d}y}{\mathrm{d}x}$ and $\frac{\mathrm{d}y}{\mathrm{d}w}$?



Backprop with gradients

- If $x \in \mathbb{R}^M$, then $\frac{\mathrm{dL}}{\mathrm{d}x} \in \mathbb{R}^M$
- Similarly, if $y \in \mathbb{R}^N$, then $\frac{\mathrm{dL}}{\mathrm{d}y} \in \mathbb{R}^N$

- So, $W \in \mathbb{R}^{N \times M}$, then $\frac{\mathrm{dL}}{\mathrm{d}W} \in \mathbb{R}^{N \times M}$
- So, what are the dimensions of $\frac{\mathrm{d}y}{\mathrm{d}x}$ and $\frac{\mathrm{d}y}{\mathrm{d}w}$?



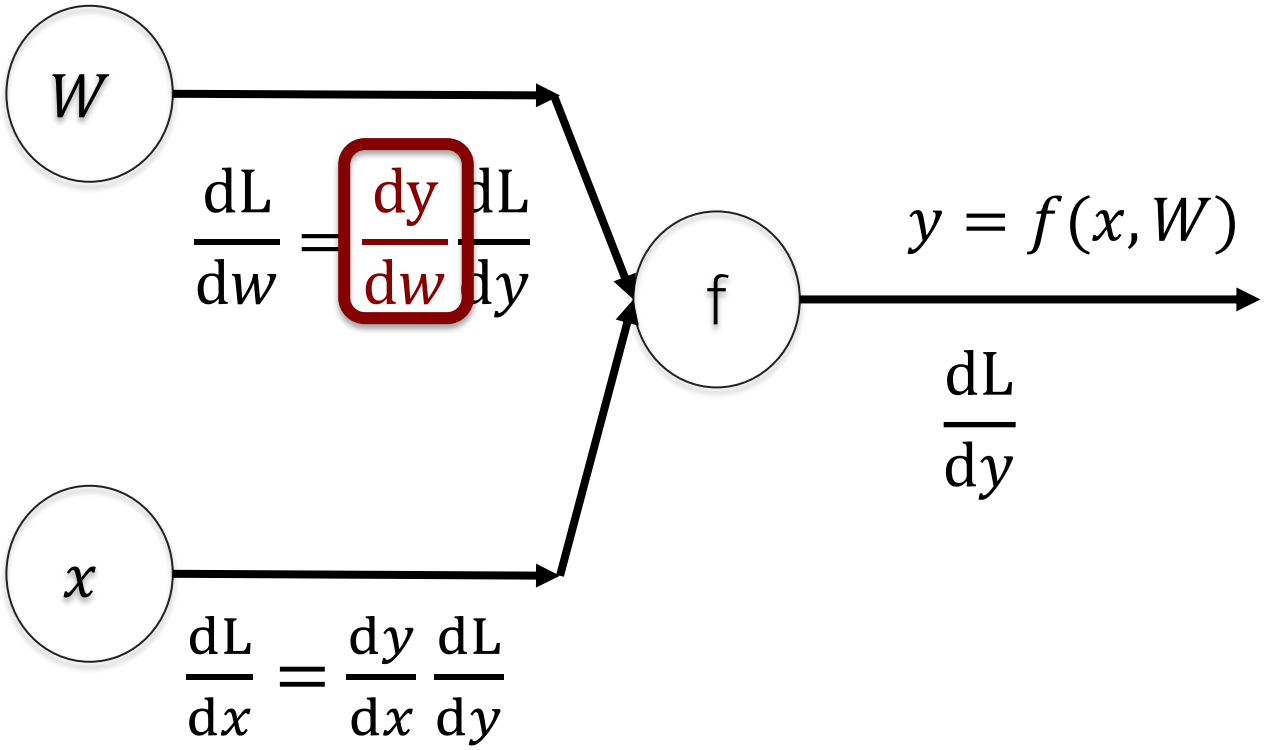
$$\frac{\mathrm{d}y}{\mathrm{d}x} \in \mathbb{R}^{M \times N}$$

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \cdots & \frac{\mathrm{d}y_N}{\mathrm{d}x_1} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}y_1}{\mathrm{d}x_M} & \cdots & \frac{\mathrm{d}y_N}{\mathrm{d}x_M} \end{bmatrix}$$

Backprop with gradients

- If $x \in \mathbb{R}^M$, then $\frac{\mathrm{dL}}{\mathrm{d}x} \in \mathbb{R}^M$
- Similarly, if $y \in \mathbb{R}^N$, then $\frac{\mathrm{dL}}{\mathrm{d}y} \in \mathbb{R}^N$

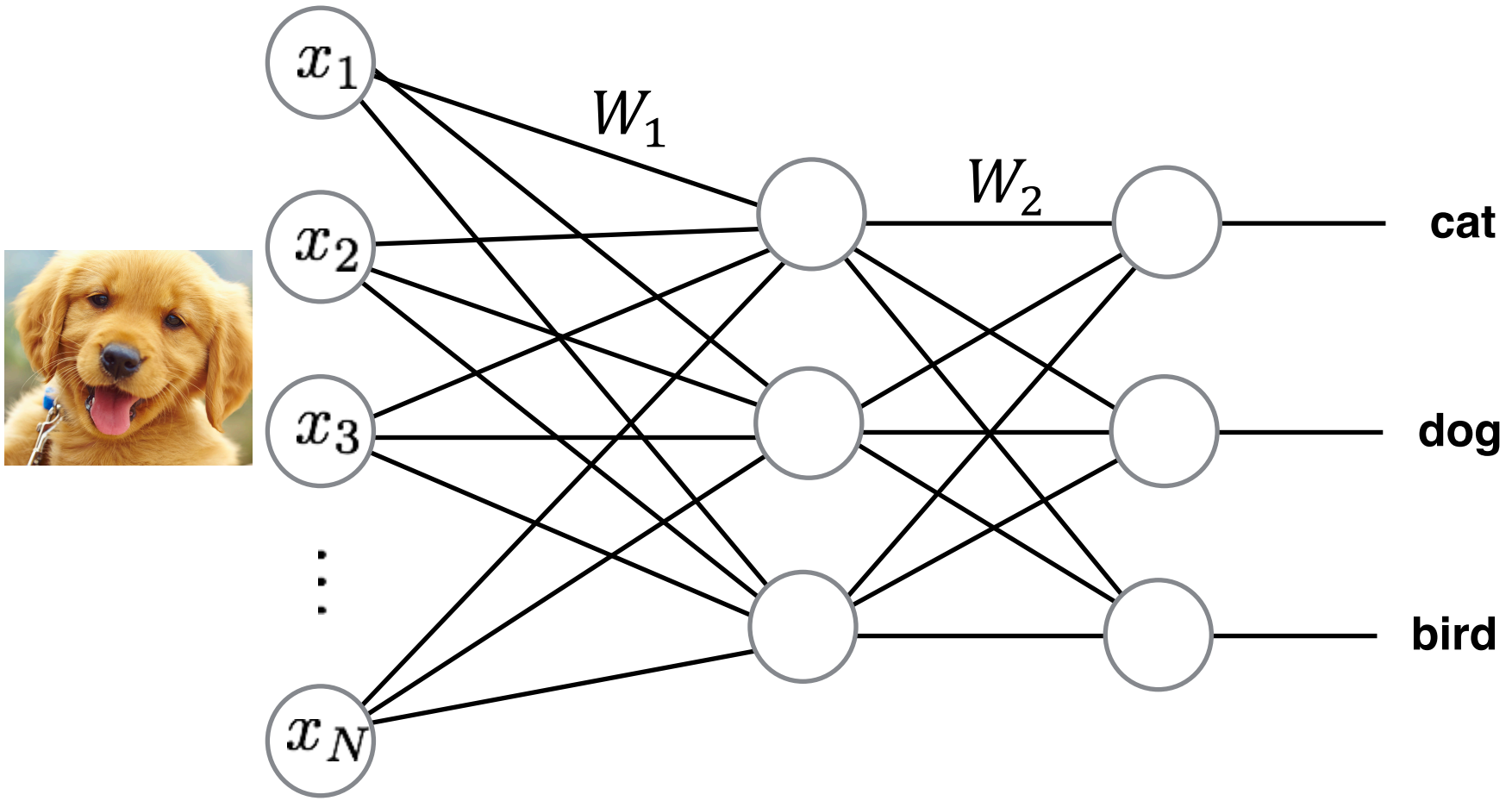
- So, $\mathrm{W} \in \mathbb{R}^{N \times M}$, then $\frac{\mathrm{dL}}{\mathrm{d}W} \in \mathbb{R}^{N \times M}$
- So, what are the dimensions of $\frac{\mathrm{d}y}{\mathrm{d}x}$ and $\frac{\mathrm{d}y}{\mathrm{d}w}$?



$$\frac{\mathrm{d}y}{\mathrm{d}w} \in \mathbb{R}^{(N \times M) \times N}$$

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}w_1} & \cdots & \frac{\mathrm{d}y_N}{\mathrm{d}w_1} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}y_1}{\mathrm{d}w_{NM}} & \cdots & \frac{\mathrm{d}y_N}{\mathrm{d}w_{NM}} \end{bmatrix}$$

Wrap up: Backprop allows us to build arbitrarily large Neural Networks and gradients only need to know local input and output information to calculate gradients.



Today's agenda

- Backprop in neural networks
- Convolutional neural networks
- Architecture design
- Exam and brief class overview
- En fin

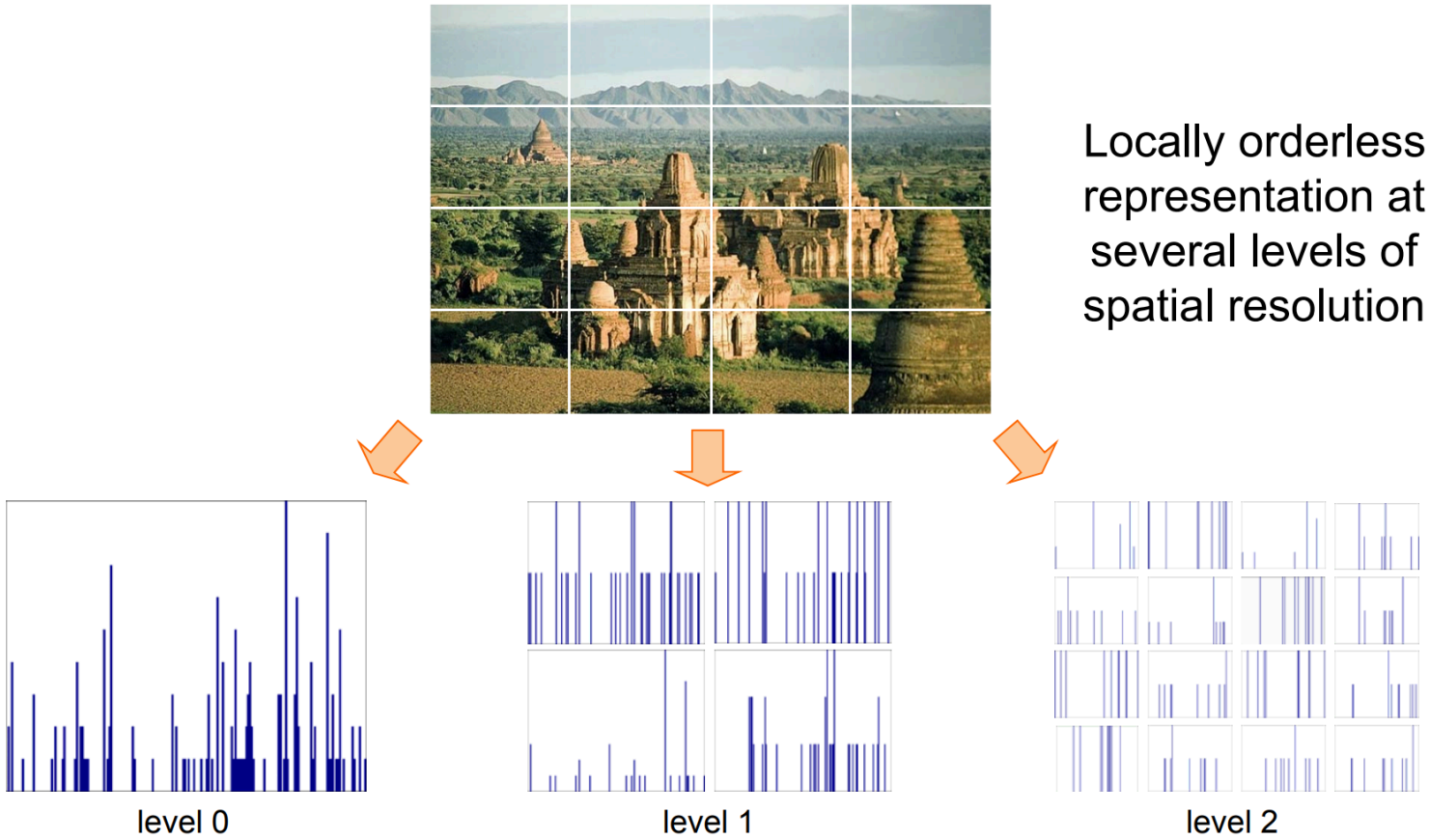
Recall: we can featurize images into a vector

Raw pixels
Raw pixels + (x,y)
PCA
LDA
BoW
BoW + spatial pyramids

Image
Vector

But these representations lose spatial information

Bag of words had the same problem
Solution: pyramids



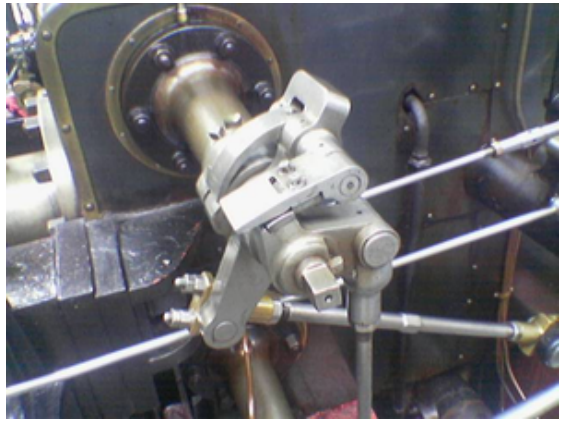
Can we design a neural network to retain spatial information?

- YES

- We have already designed convolutions to be spatially invariant.

Recall: well designed convolutions give important features

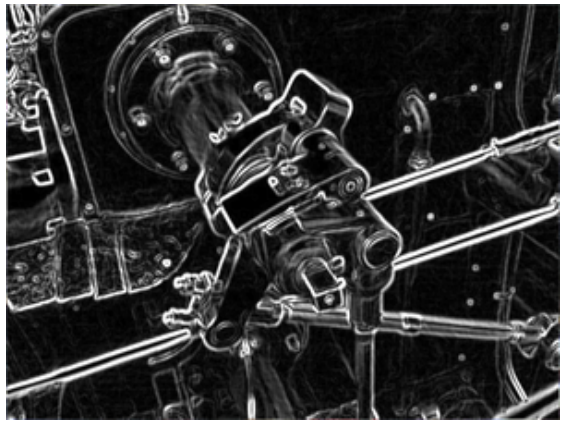
Just like the weights in the neural network, we can learn the weights of a convolution filter as well



*

0	-½	0
0	0	0
0	½	0

=



The convolution layer

- Traditionally we used only 1 2D filter at a time.
- And operated only on black-and-white images

32x32x1

w1	w2	w3
w4	w5	w6
w7	w8	w9

32x32x1

Calculating the size of activation after applying convolution:

- Stride $S = 1$, Padding $P = 2$, Spatial extend $F = 3$ for a 3x3 kernel
- given input size ($W_1 = 32$, $H_1 = 32$), $W_2 = \frac{W_1 - F + 2P}{S} + 1$, $H_2 = \frac{H_1 - F + 2P}{S} + 1$
- So, $W_2 = \frac{W_1 - F + 2P}{S} + 1 = \frac{32 - 3 + 2}{1} + 1 = 32$

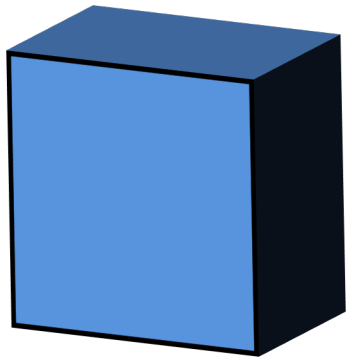
32x32x1

w1	w2	w3
w4	w5	w6
w7	w8	w9

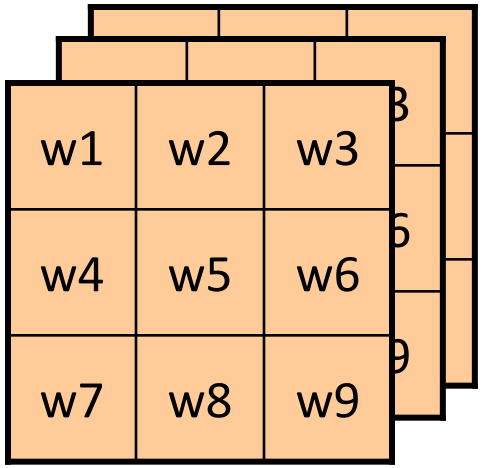
32x32x1

The convolution layer

- We can apply 3 filters to operate over all three color channels



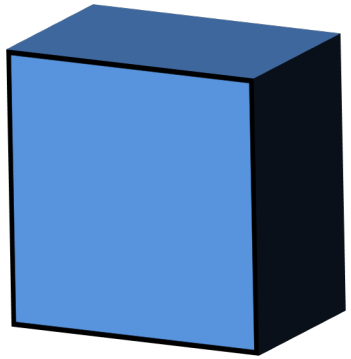
32x32x3



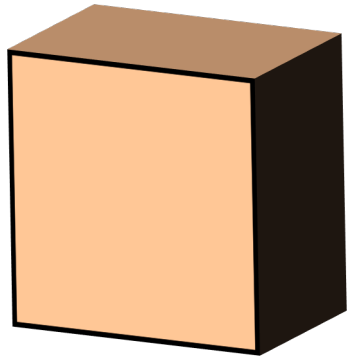
32x32x1

The convolution layer

- Let's represent this new kernel as a tensor as well



32x32x3

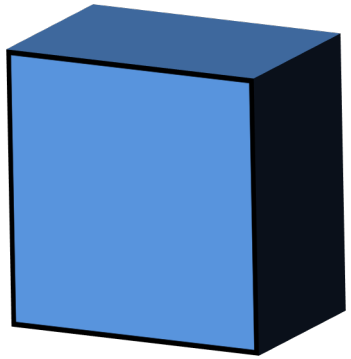


3x3 kernel

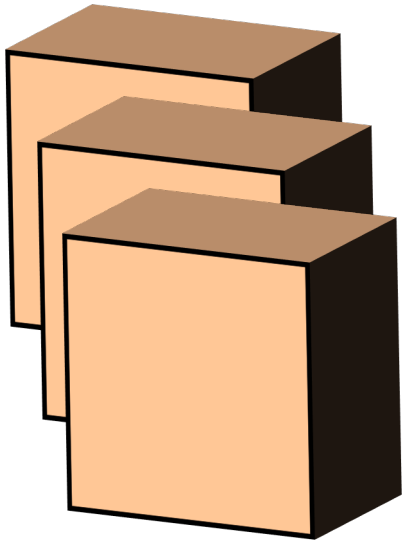
32x32x1

The convolution layer

- We can apply $K = 64$ multiple kernels in parallel to learn different kinds of features



32x32x3

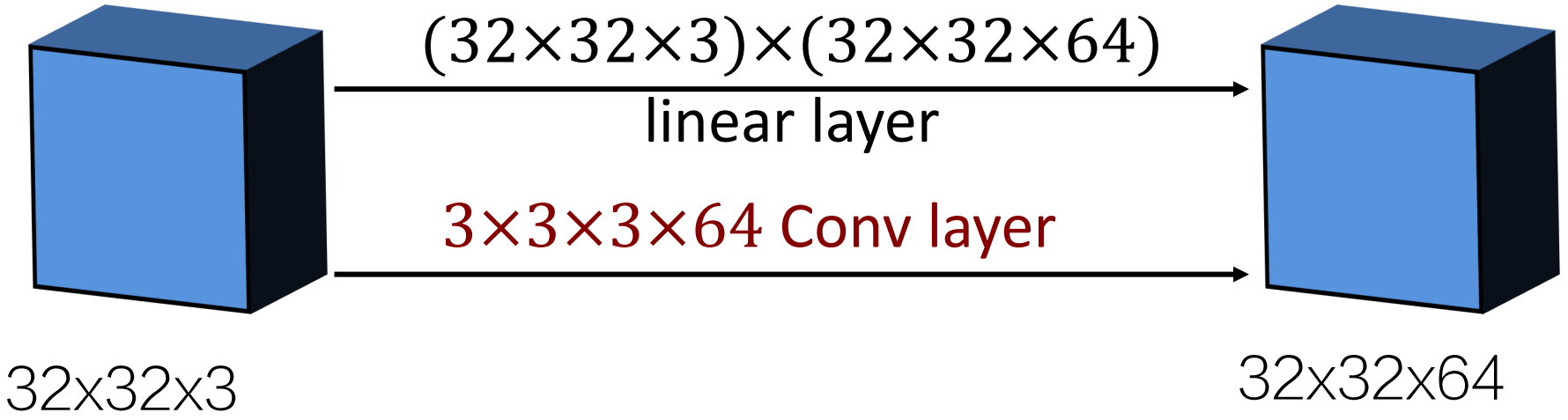


3x3 kernel
64 filters

32x32x64

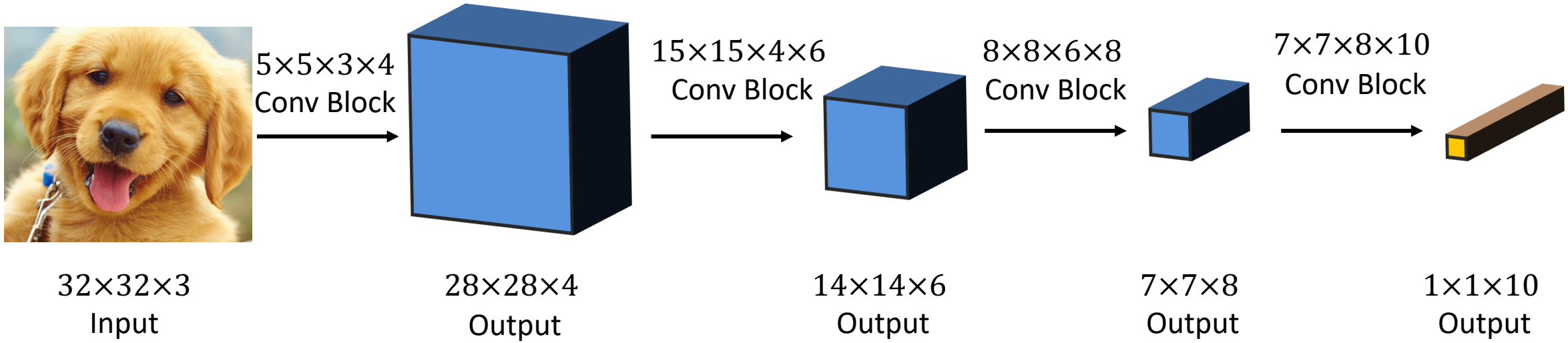
The convolution layer

- How many weights would a linear layer need to convert the input to the following feature?
- How many weights would a convolution filter need?



Stacking Convolutions

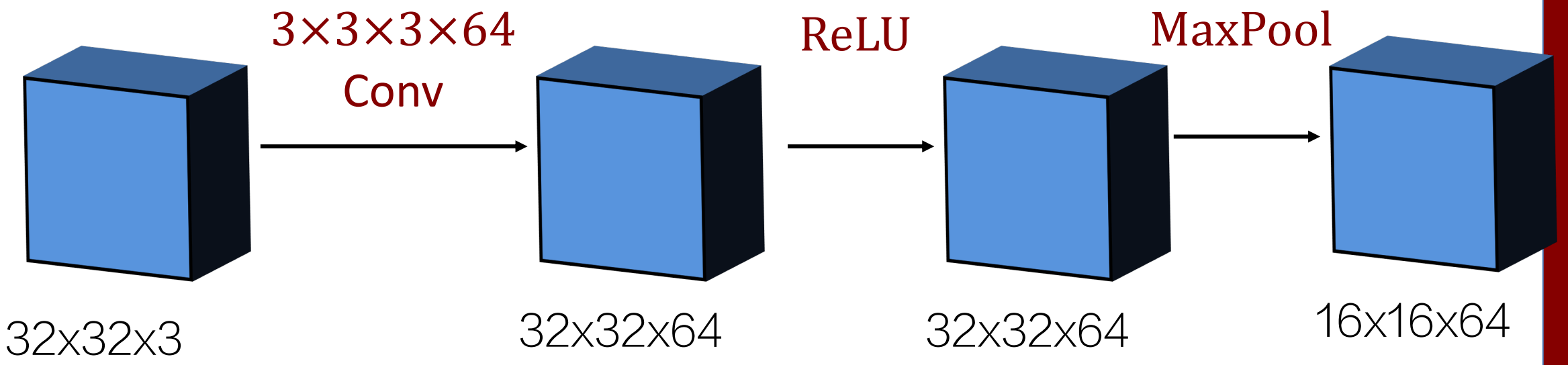
A convolutional neural network architecture



Note that the non-linear ReLU layers between conv layers are not shown

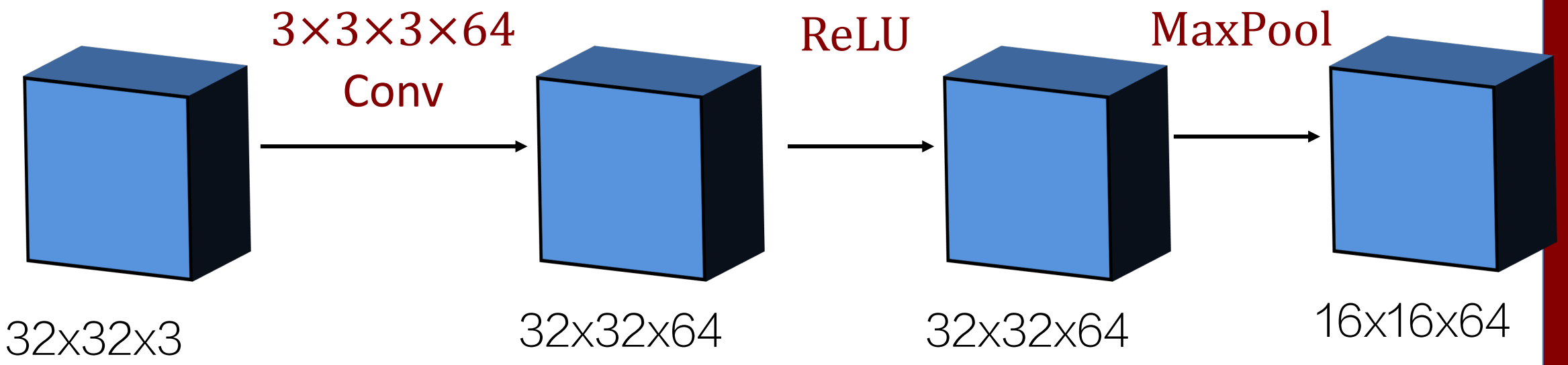
Today's conv networks have pool layers

Pooling layers reduce the spatial dimensions by 2



Today's conv networks have pool layers

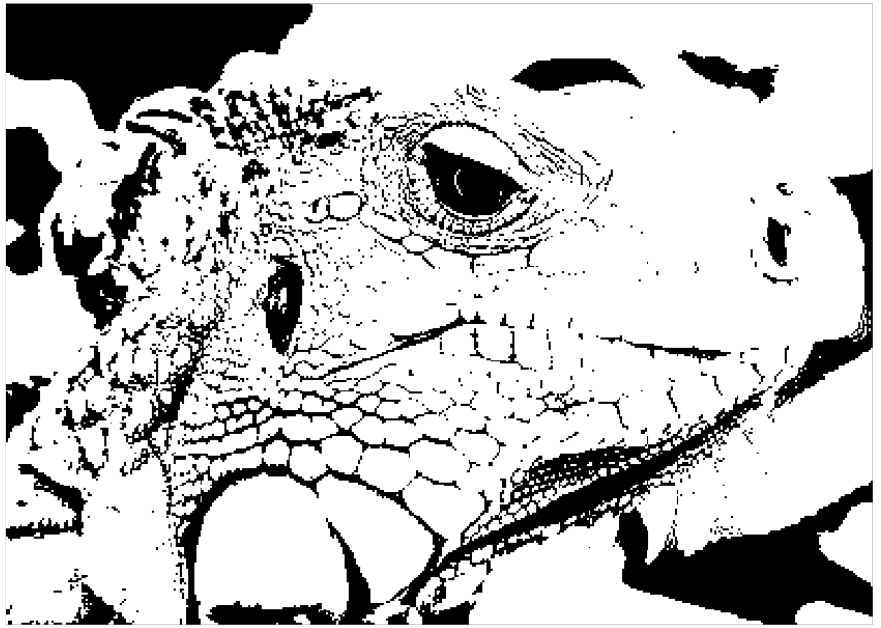
Pooling layers reduce the spatial dimensions by 2



Similar to Filter example #2: Image Segmentation from lecture #2

- Image segmentation based on a simple threshold:

$$g[n,m] = \begin{cases} 255, & f[n,m] > 100 \\ 0, & \text{otherwise.} \end{cases}$$



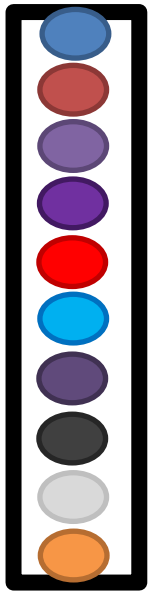
Today's agenda

- Backprop in neural networks
- Convolutional neural networks
- Architecture design
- Exam and brief class overview
- En fin

Convolutional neural network

$32 \times 32 \times 3$
Input

Conv Block → ReLU → Conv Block → ReLU → Pool → Conv Block → ReLU → Conv Block → ReLU → Pool → Conv Block → ReLU → Conv Block → ReLU → Pool

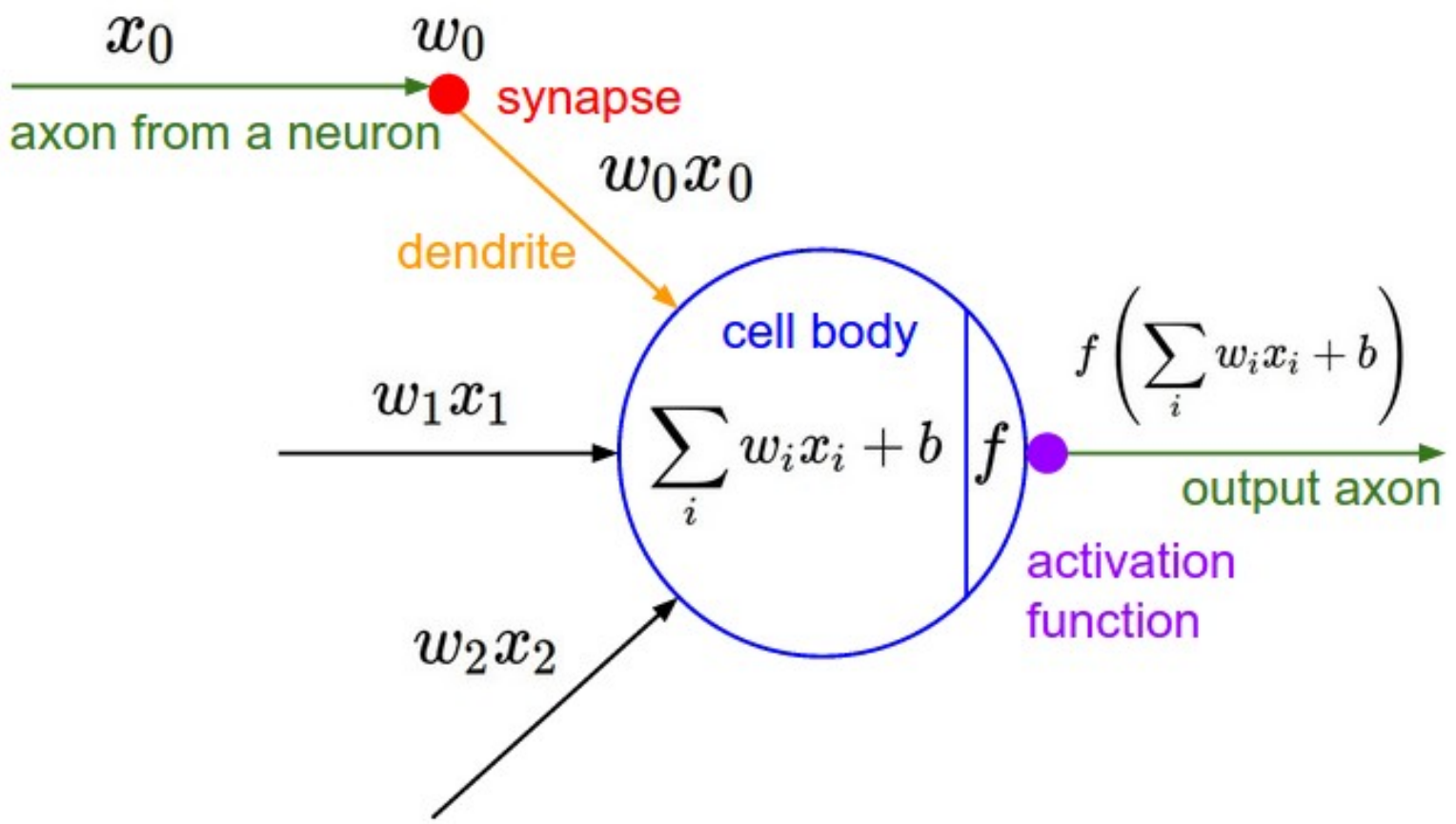


$10x1$
Output

Frank Rosenblatt, ~1957: Perceptron

implemented in circuits / electronics in hardware

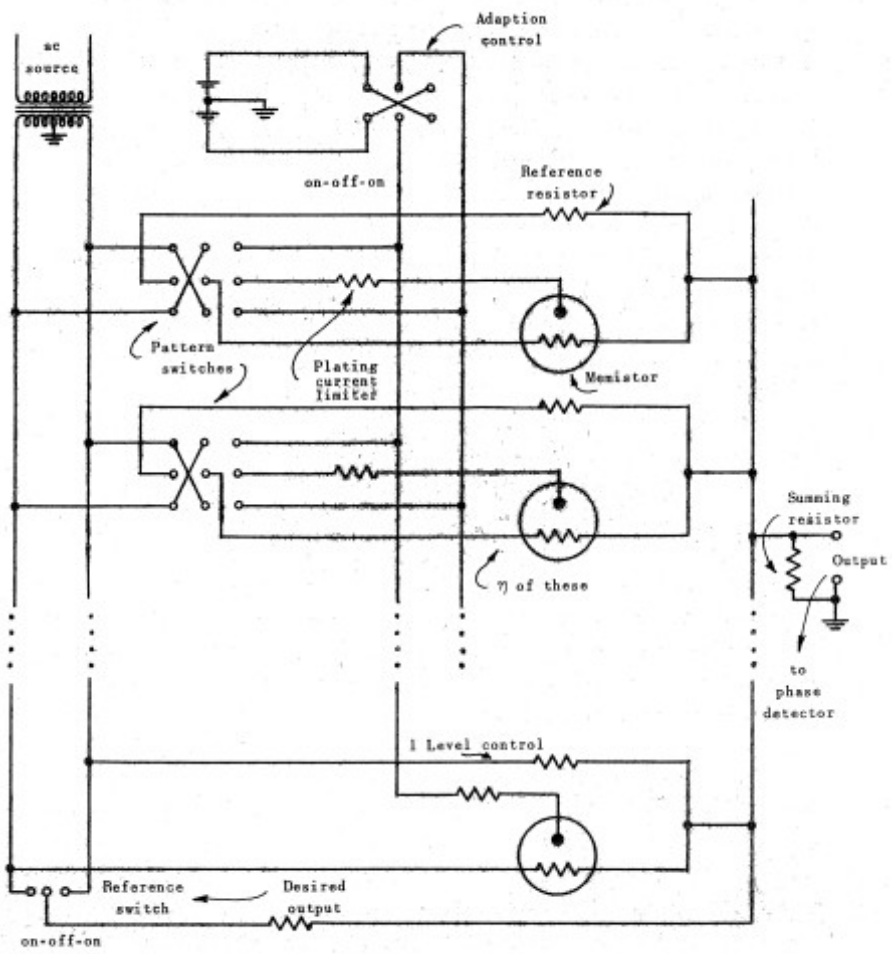
no loss function
no backprop



Widrow and Hoff, ~1960: Adaline/Madaline

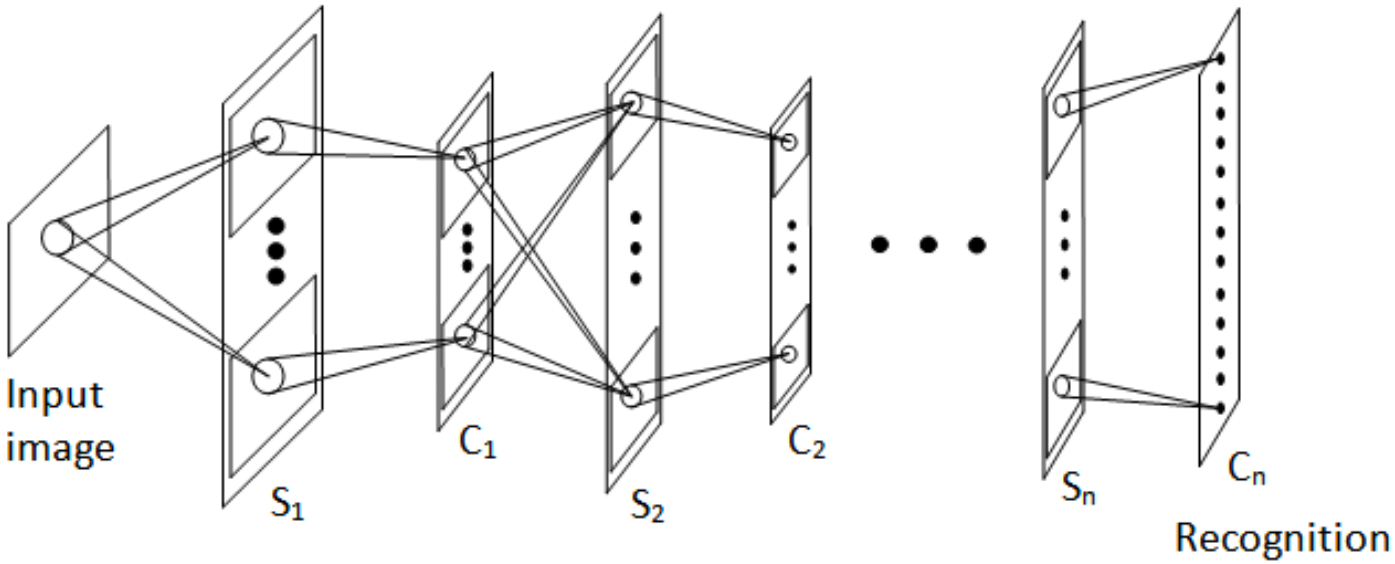
- Introduced the idea of stacking layers of perceptrons

- no loss function
- no backprop



Fukushima 1980 – NeoCognitron

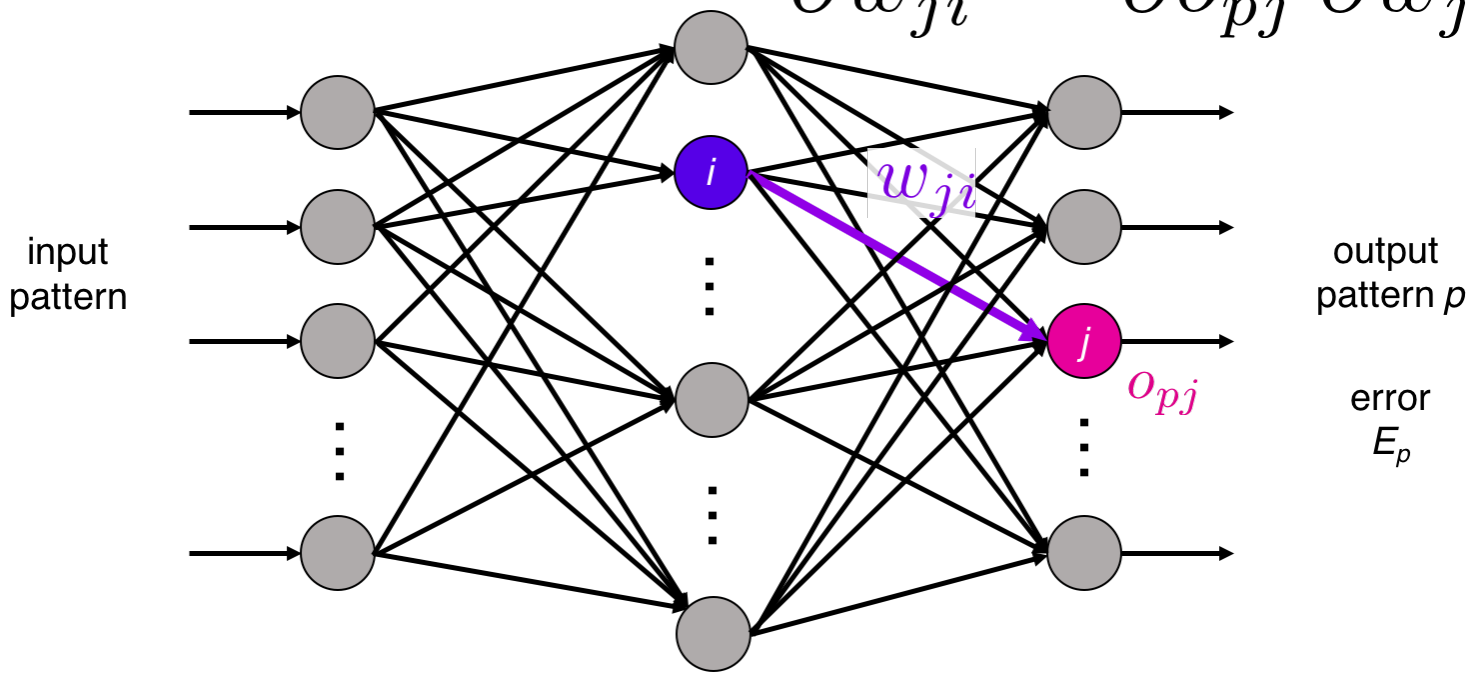
- First network architecture
- Took inspiration from Hubel and Wiesel's Neuroscience research and designed two types of neurons:
  - simple cells: contains weights that need to be learned
  - complex cells: pools activations from previous layers as a way of modeling the hierarchical nature of our visual system



Rumelhart et al., 1986: back-propagation

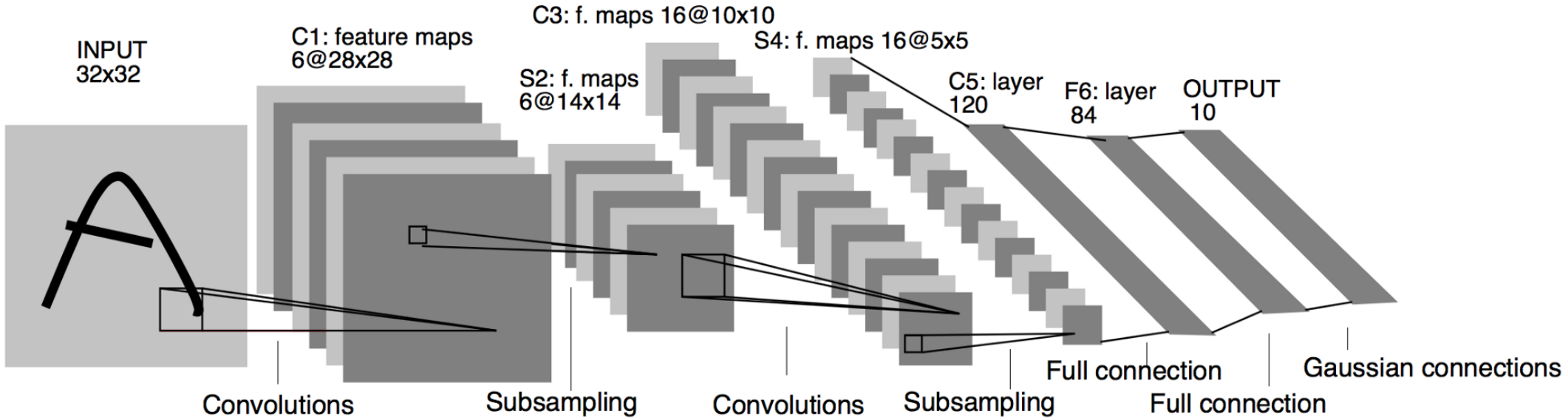
- Chain rule and update rule

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$



LeCun '97 - LeNet architecture

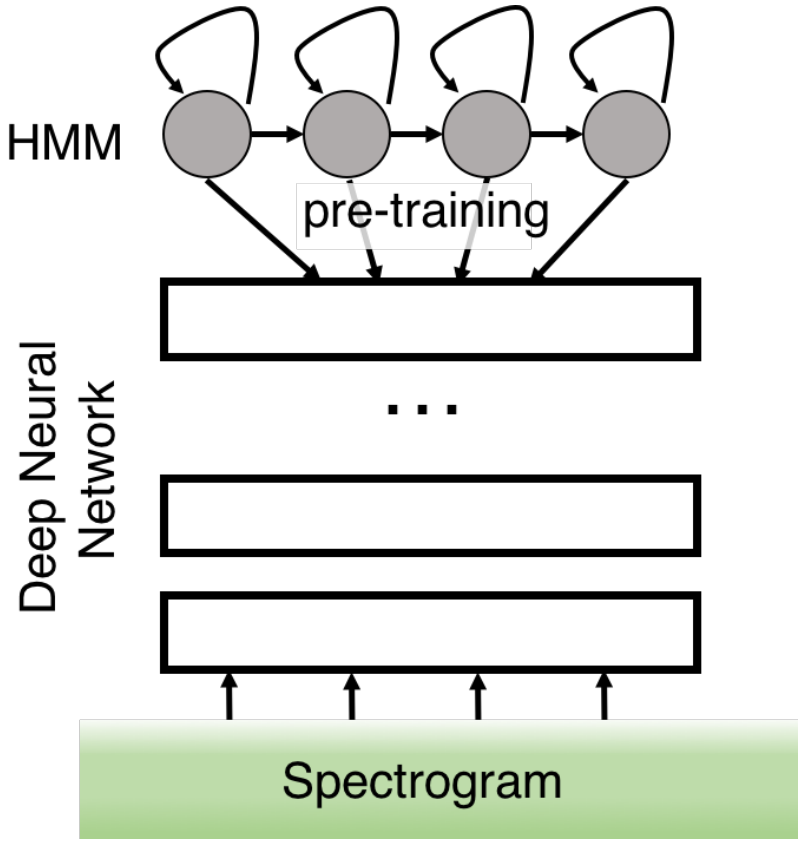
- First real use case: Yann LeCun's digit recognizer
- Used by postoffice to recognize digits
- 70% test accuracy on MNIST (dataset of digits)
- They do average pooling instead of max pooling



First major breakthrough with deep learning

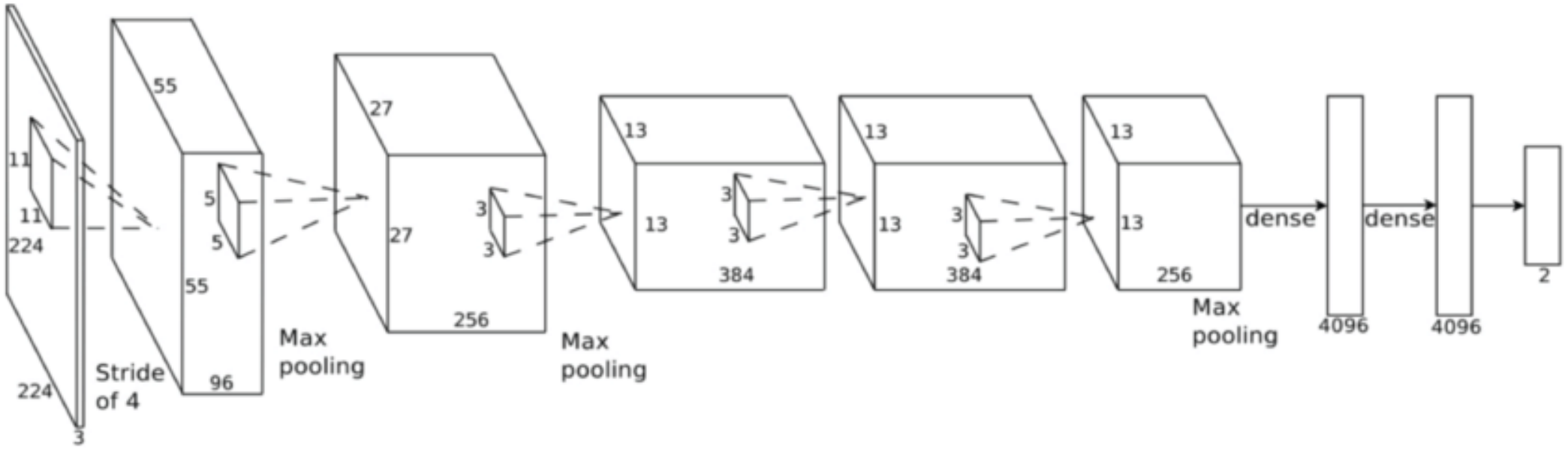
- Mohamed et al. **Acoustic Modeling using Deep Belief Networks,** 2010

- Dahl et al. **Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition**, 2012



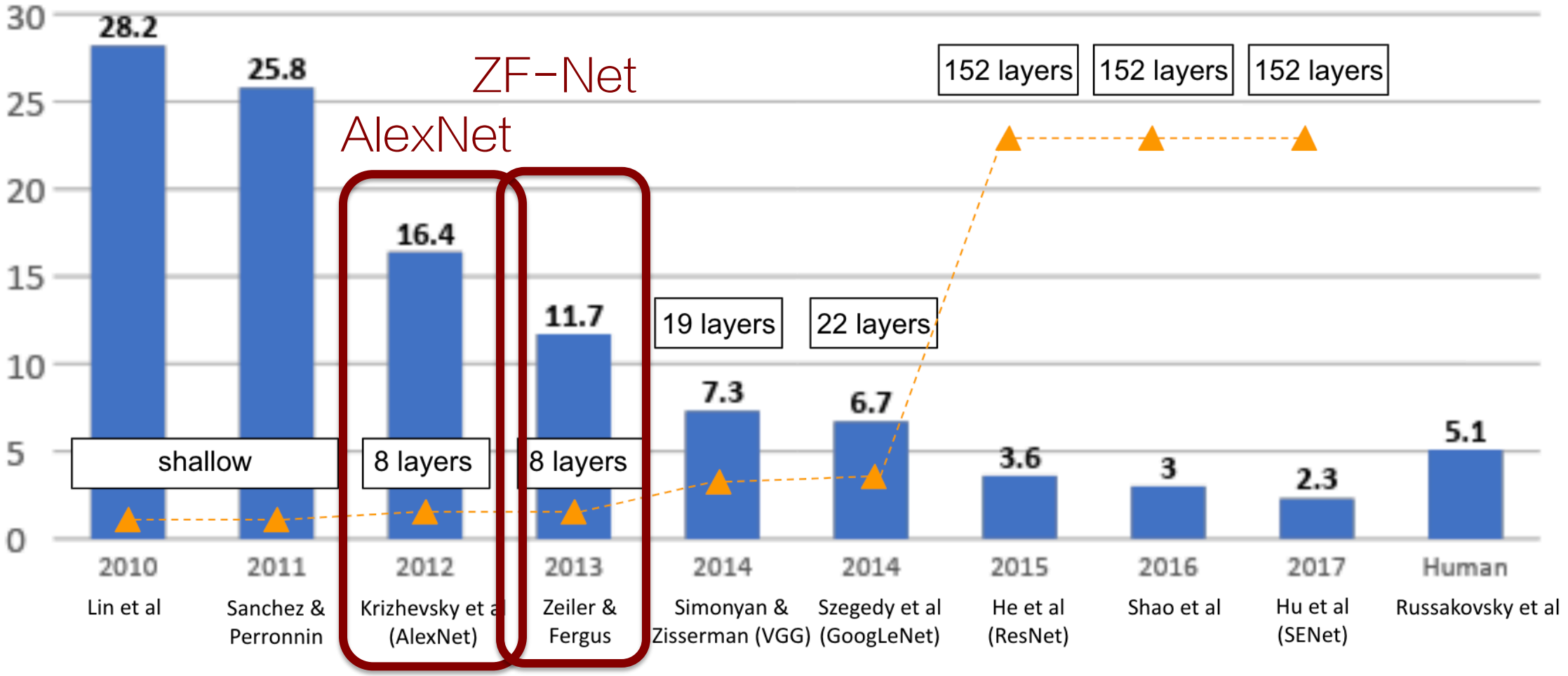
First major deep learning breakthrough in Computer Vision

AlexNet – 2012

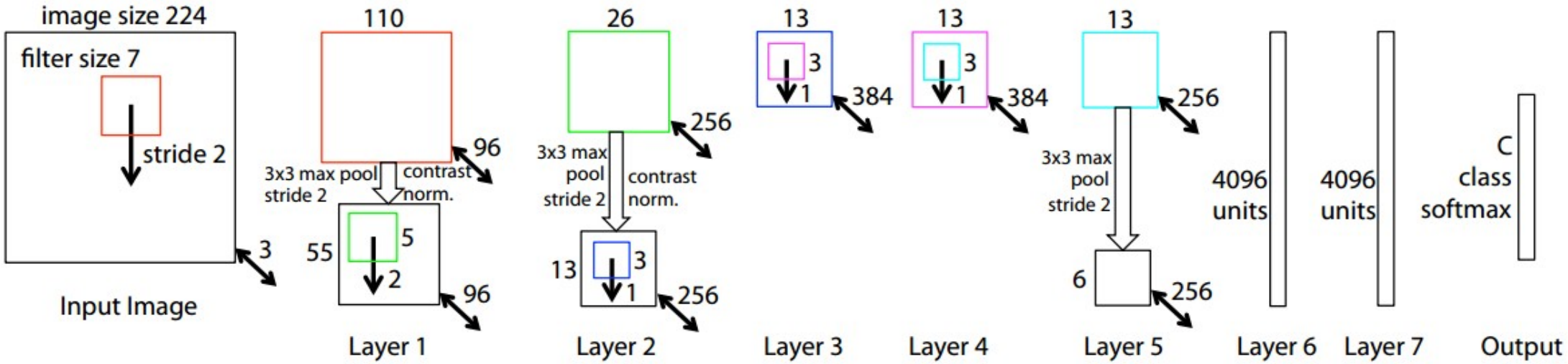


- Reduced error rate in the ImageNet Challenge in half
- very similar overall structure to LeNet architecture, only larger
- needed a GPU to train in a reasonable amount of time
- Incorporated Neurocognitron's pooling layers

ImageNet Challenge error rates over time

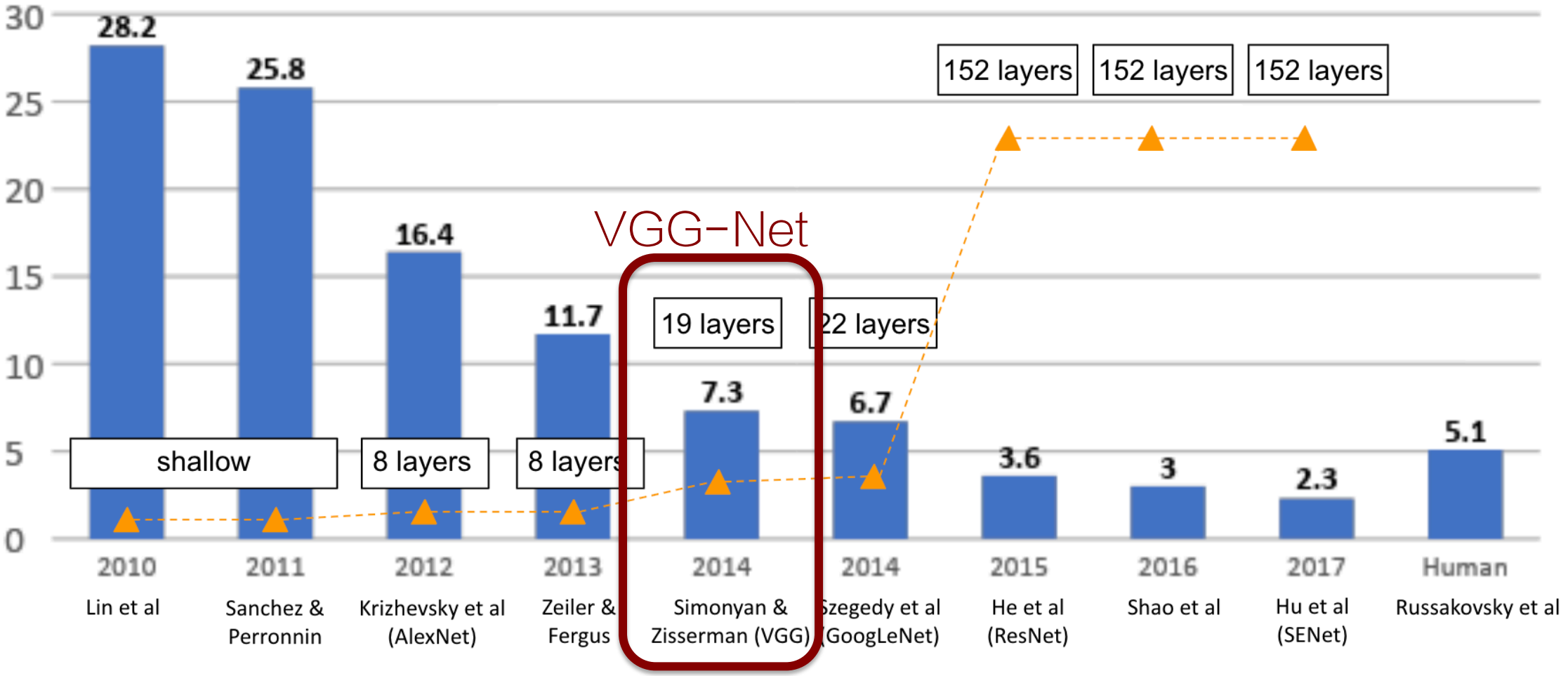


Zeiler 2013 – ZF-Net



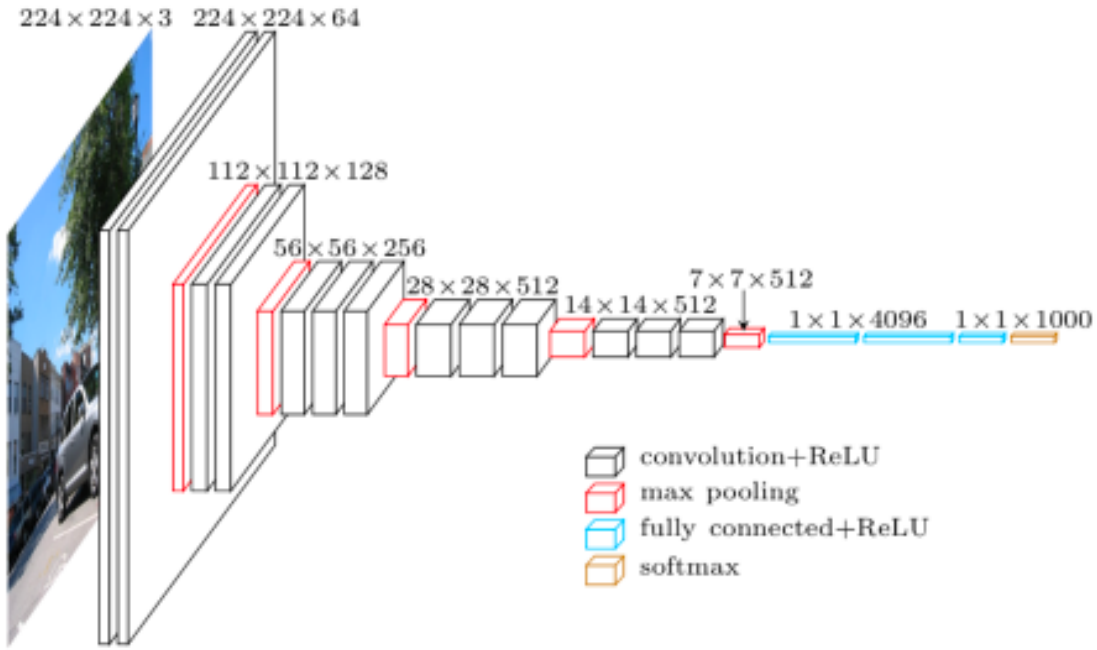
- Changes from convolution sizes of 11 to 7.
- Changes strides from 4 to 2
- Used more filters per layer

ImageNet Challenge error rates over time

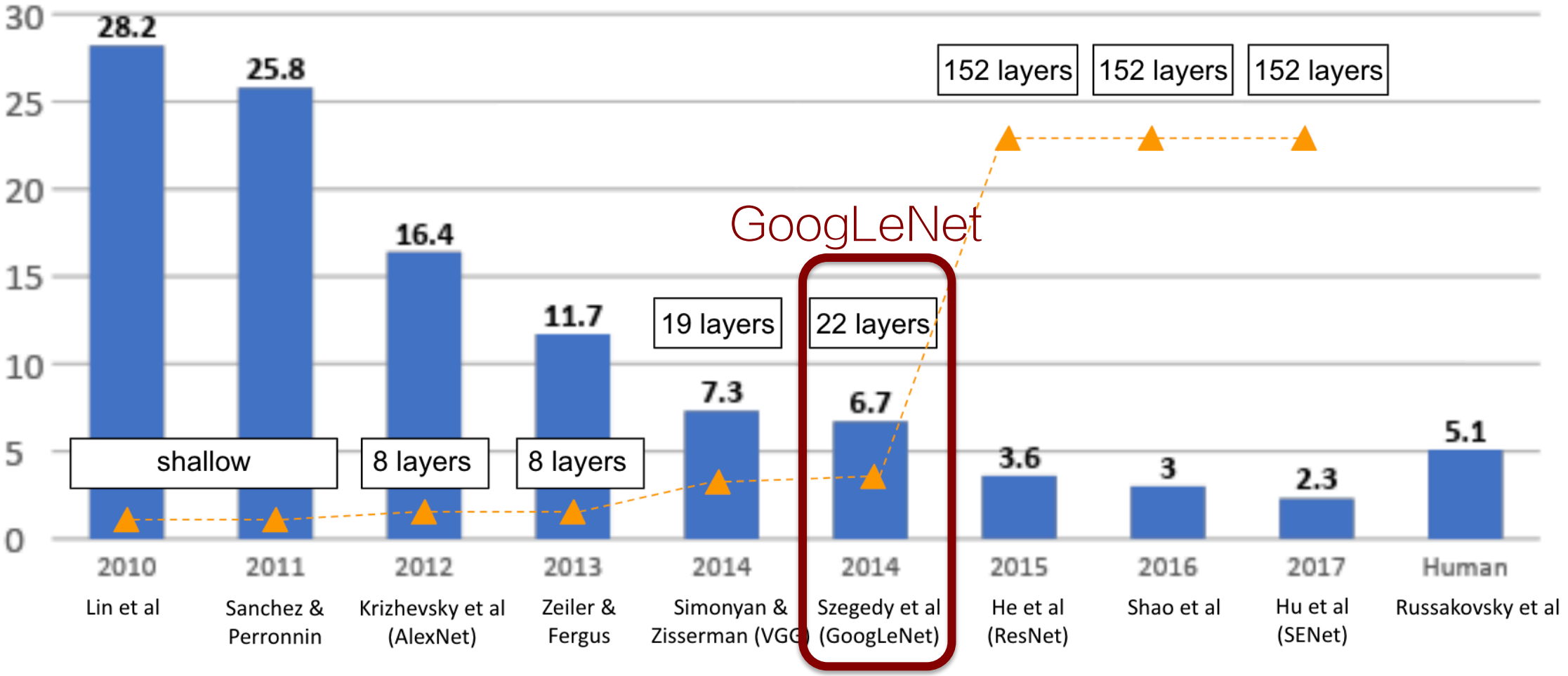


Simmoyan 2014 – VGG-Net

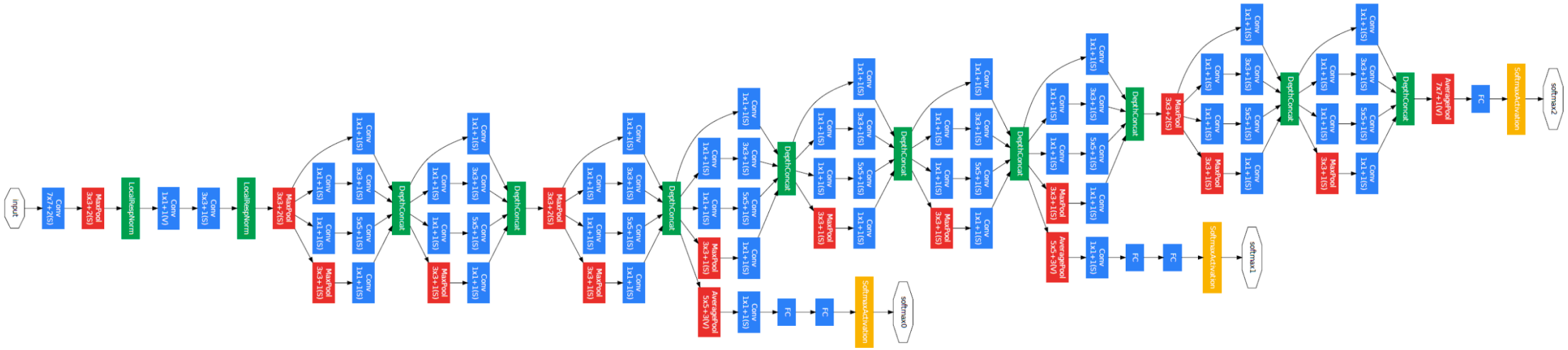
- Deeper than previous layers
  - 16 layers instead of 8
- They used 3x3 conv filters instead of 7x7 used by AlexNet. Why?



ImageNet Challenge error rates over time

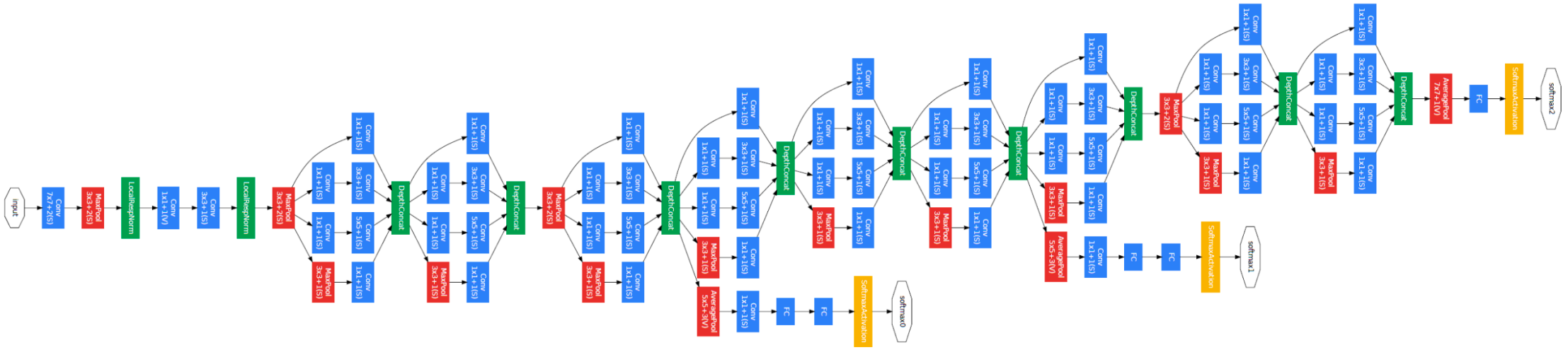


Szegedy et al. 2014 – GoogLeNet



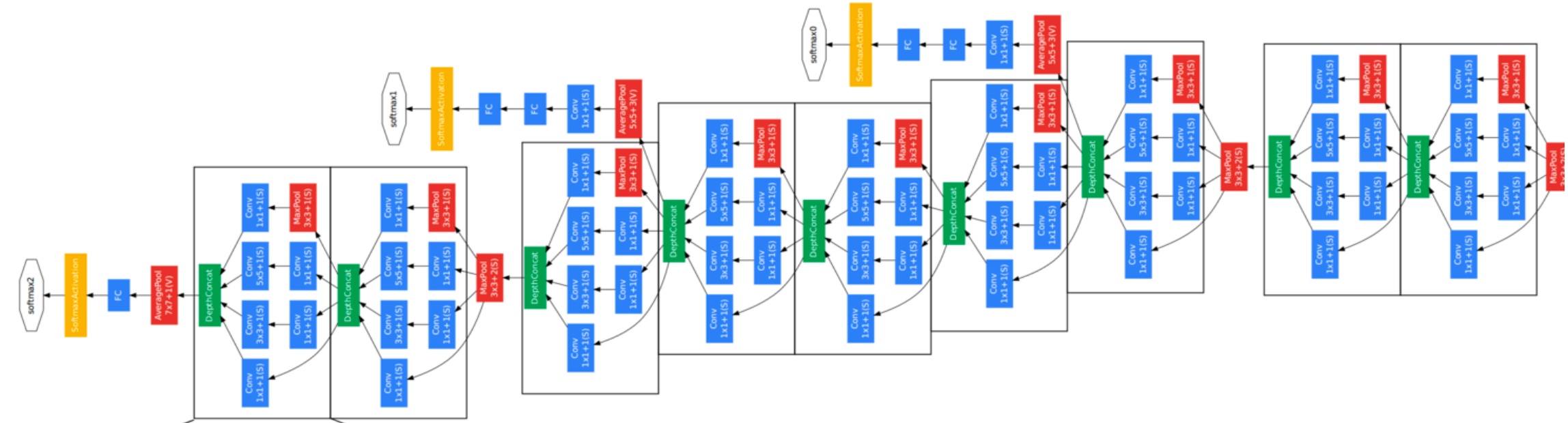
- 22 layers
- Efficient “Inception” module
- No fully connected layers
- Only 5 million parameters!
  - 12x less than AlexNet

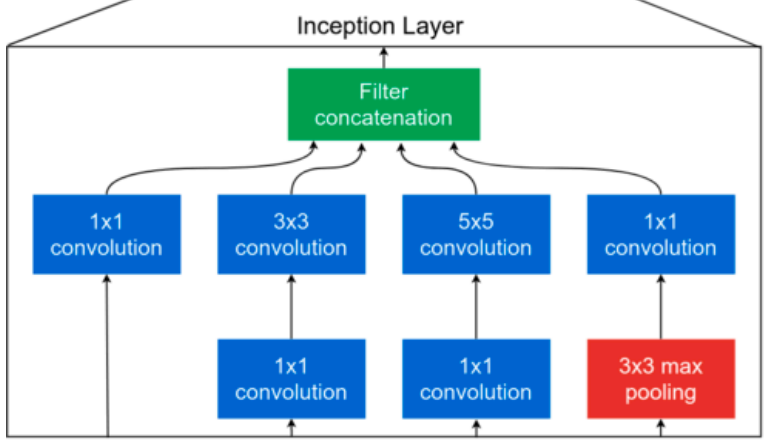
Szegedy et al. 2014 – GoogLeNet



- Multiple output branches to make sure gradients reached the earlier layers.
- As gradients are propagated backwards, they might explore or vanish as we are multiple partial gradients together. Having multiple branches fixes the vanishing gradient problem.

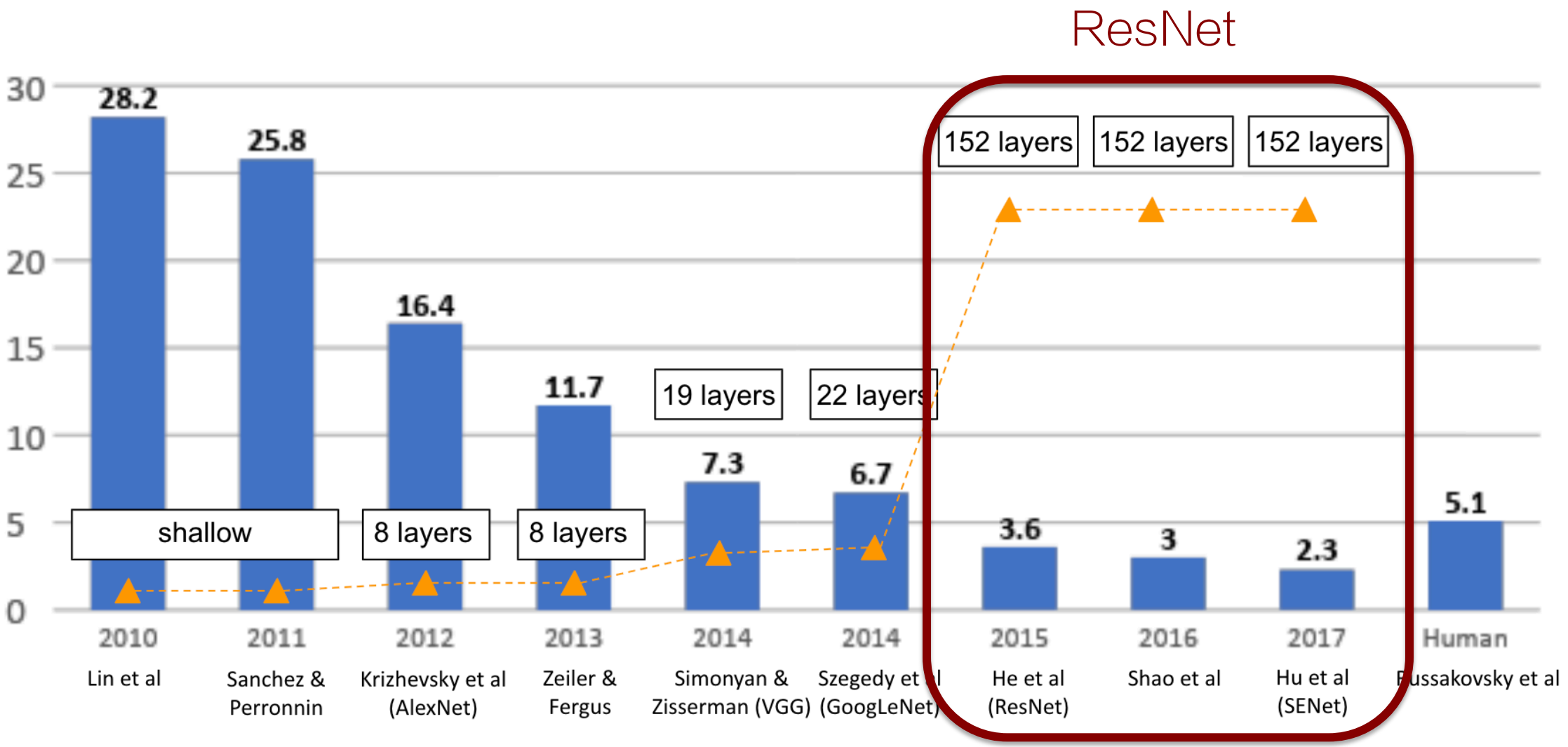
Szegedy et al. 2014 – GoogLeNet



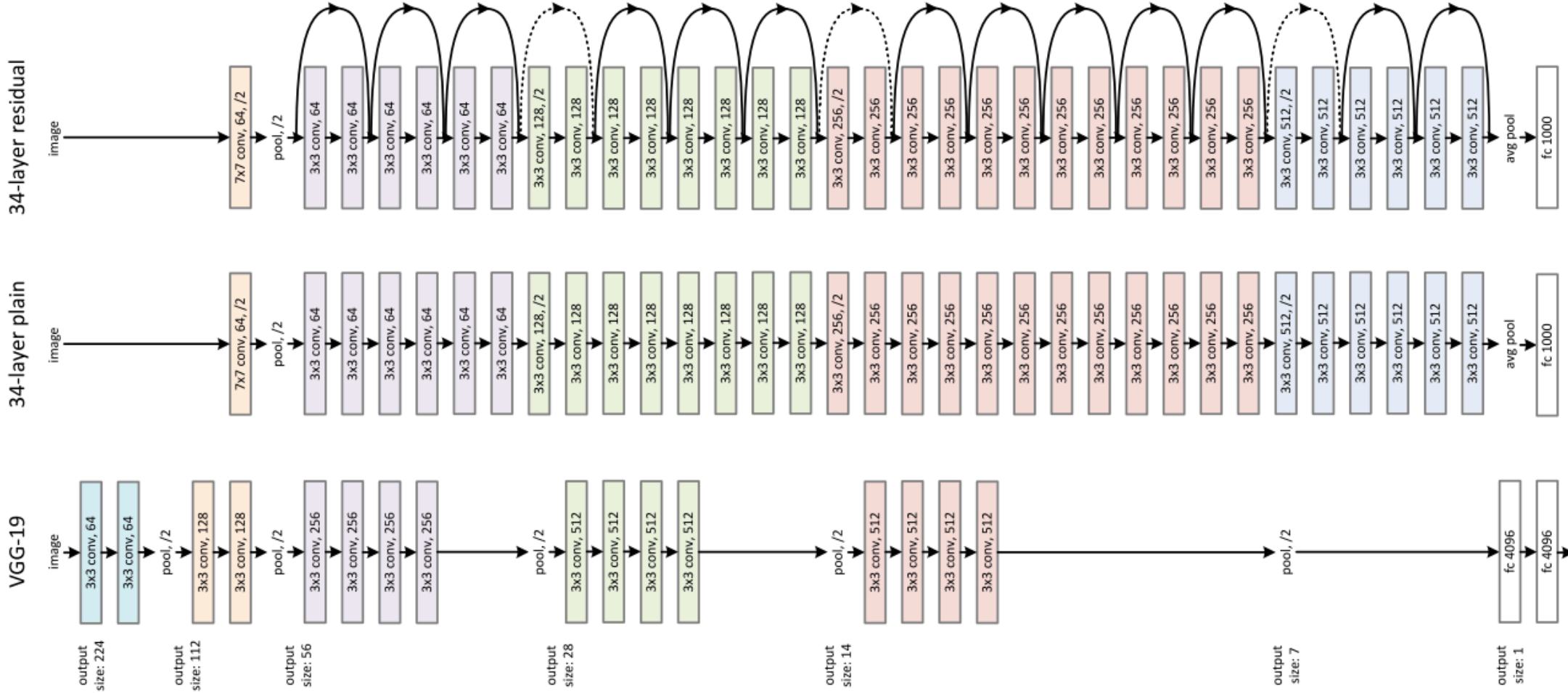


Inception module – hand designed local network topology that can be thought of as a network within a network and stacked on top of each other

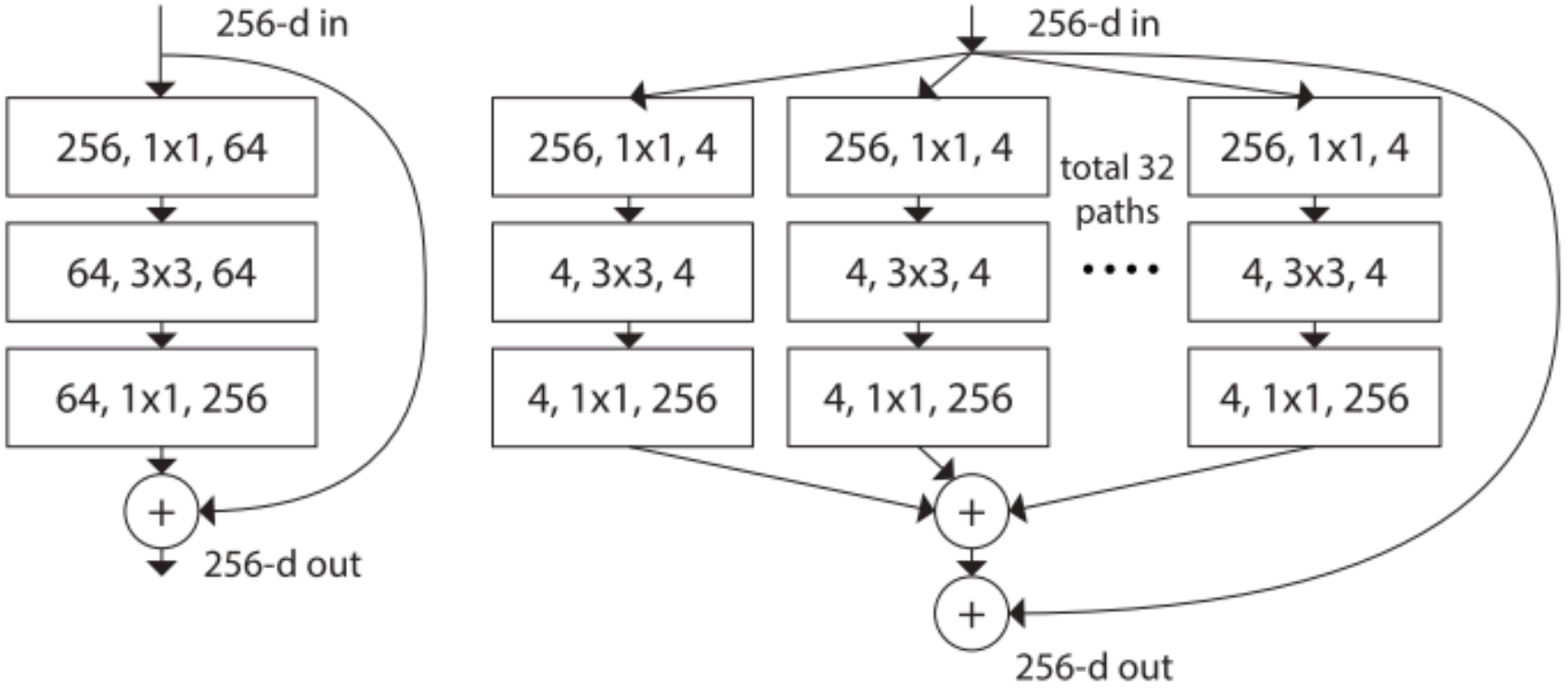
ImageNet Challenge error rates over time



ResNet depth in comparison to VGG



He 2016 – ResNet



- Learning residuals instead of feature transformations
- Also prevented the vanishing gradient problem

But there are open problems with all these architectures

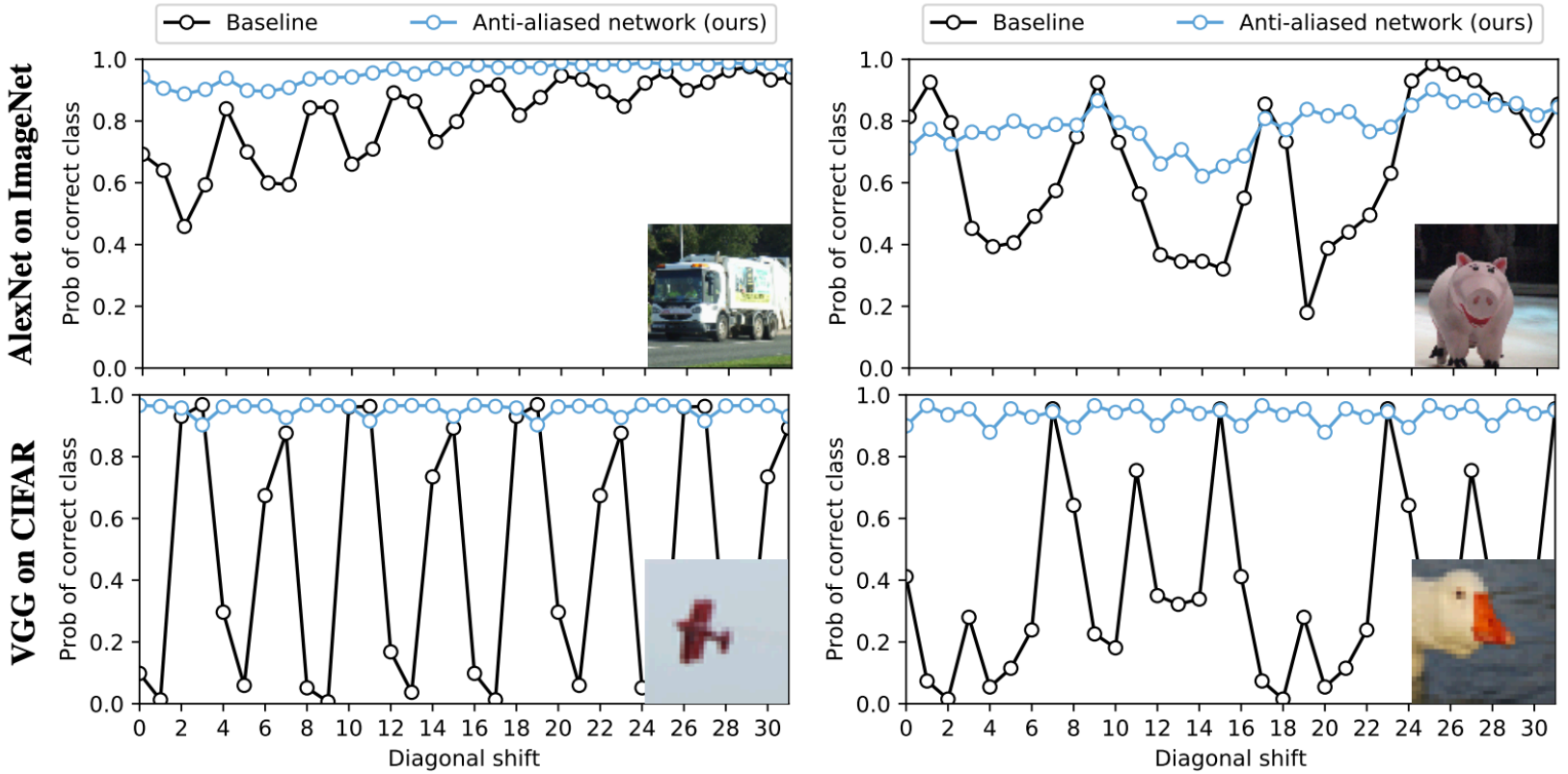
- Let's consider on one specific problem.
- Remember from lecture 2:
  - we derived convolutions to mimic properties of our own visual system
  - One of those properties was shift invariance.

$$f[n - n_0, m - m_0] \xrightarrow{\mathcal{S}} g[n - n_0, m - m_0]$$

- But are these architectures shift invariant systems?

Recent paper found that all these architectures are not shift-invariant

- black line is AlexNet and VGG on ImageNet and CIFAR datasets.
- They proposed a small fix that improves all these models (in blue)



Zhang, Making Convolutions Shift-Invariant Again, ICML 2019

Can anyone guess which component is not shift invariant?

- All the models we spoke of are composed of three building blocks:
  - Convolution layers
  - Pooling Layers
  - ReLU activations

Convolutions were designed to be shift-invariant

$$f[n - n_0, m - m_0] \xrightarrow{\mathcal{S}} g[n - n_0, m - m_0]$$

Let's test ReLU with an example

12	3	19	−1	2
25	10	1	−2	2
9	7	17	−5	4
−2	−5	−1	−4	1
1	3	2	7	7

ReLU

12	3	19	0	2
25	10	1	0	2
9	7	17	0	4
0	0	0	0	1
1	3	2	7	7

shift right

shift right

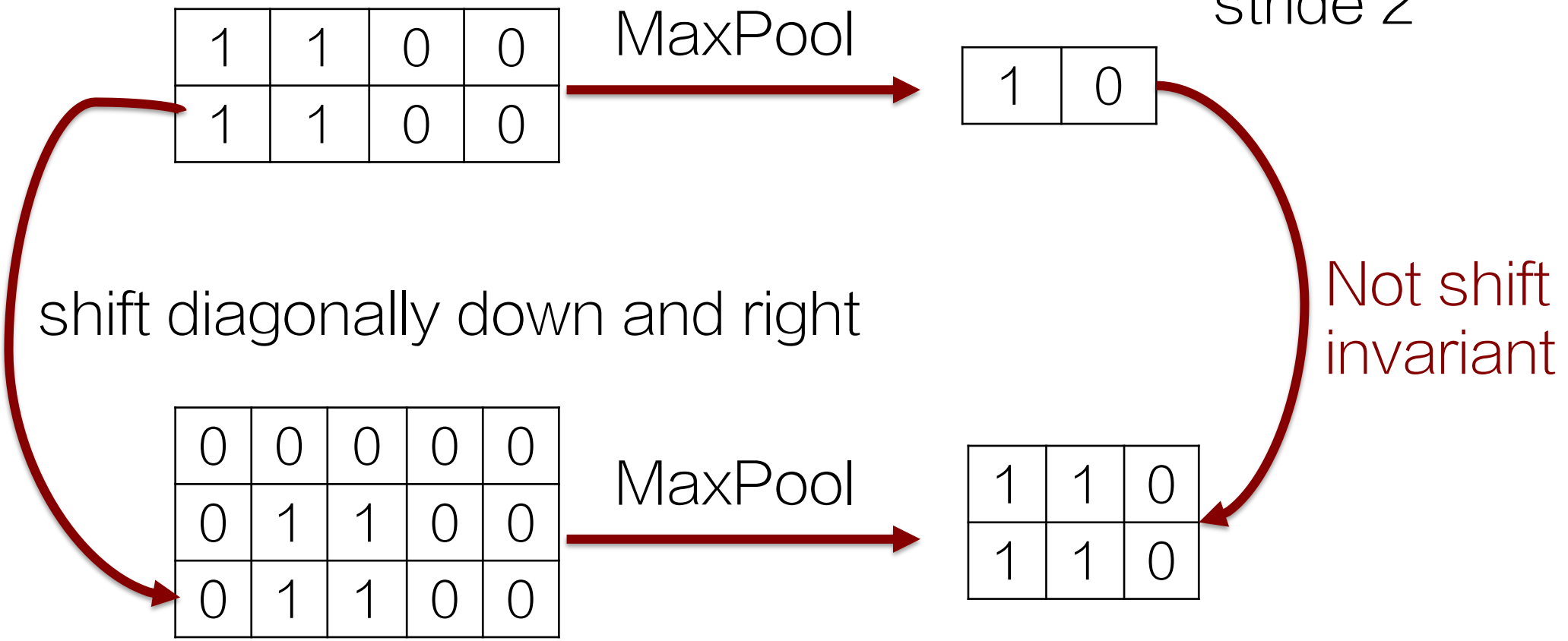
0	12	3	19	−1	2
0	25	10	1	−2	2
0	9	7	17	−5	4
0	−2	−5	−1	−4	1
0	1	3	2	7	7

ReLU

0	12	3	19	0	2
0	25	10	1	0	2
0	9	7	17	0	4
0	0	0	0	0	1
0	1	3	2	7	7

Let's test Pooling

typically all models use filter size of 2 stride 2



L2 loss: $\sqrt{3}$

They proposed a new pooling filter: BlurPool

0	1	0
1	2	1
0	1	0

1	1	0	0
1	1	0	0

Blur →

0.67	0.67	0.17	0
0.67	0.67	0.17	0

MaxPool →

0.67	0.17

shift diagonally

0	0	0	0	0
0	1	1	0	0
0	1	1	0	0

Blur →

0	0.17	0.17	0	0
0	0.67	0.67	0.17	0
0	0.67	0.67	0.17	0

MaxPool →

0.67	0.67	0.17
0.67	0.67	0.17

Zhang, Making Convolutions Shift-Invariant Again, ICML 2019

L2 loss: $\sqrt{1.37}$

Aside

- For an in-depth explanation on aliasing and how it's related to discrete signals (for ex, images), check out this link:
- http://www.rctn.org/bruno/npb261/aliasing.pdf

- This doesn't completely fix the problem as these models are still no shift-invariant. So, we still get unpredictable behavior. This is one of the many open problems left.

Today's agenda

- Backprop in neural networks
- Convolutional neural networks
- Architecture design
- Exam and class brief overview
- En fin

Exam details

- Monday, December 10
- 3:30 to 6:30pm
- Location: Hewlitt teaching center, room 200
- Practice exam available online on piazza.

- Make-up exam for those with conflicts
  - You should have received an email about this.

Class overview

- 80% assignments
- 20% final exam
- +10% extra credit

- Don’t be worried about the final.
- Optimal strategy:
  - Go over all the materials we covered in class.
  - Read through the entire final first and work on the problems you find easy first.

Exam overview

- 100 points in total
- 15 points for multiple choice
- 20 points for true false
- 25 points for filters, edges, corners, RANSAC, Hough transform
- 15 points for segmentation and seam carving
- 15 points for recognition and detection
- 10 points for video and deep learning

The goal of computer vision

- To bridge the gap between pixels and “meaning”

What we see

0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

What a computer sees

Source: S. Narasimhan

Convolution and Cross-correlation

Convolution is an integral that expresses the amount of overlap of one function as it is shifted over another function

$$(f * h)[m,n] = \sum_{k,l} f[k,l]\, h[m-k, n-l]$$

Cross-correlation compares the *similarity of two sets of data*

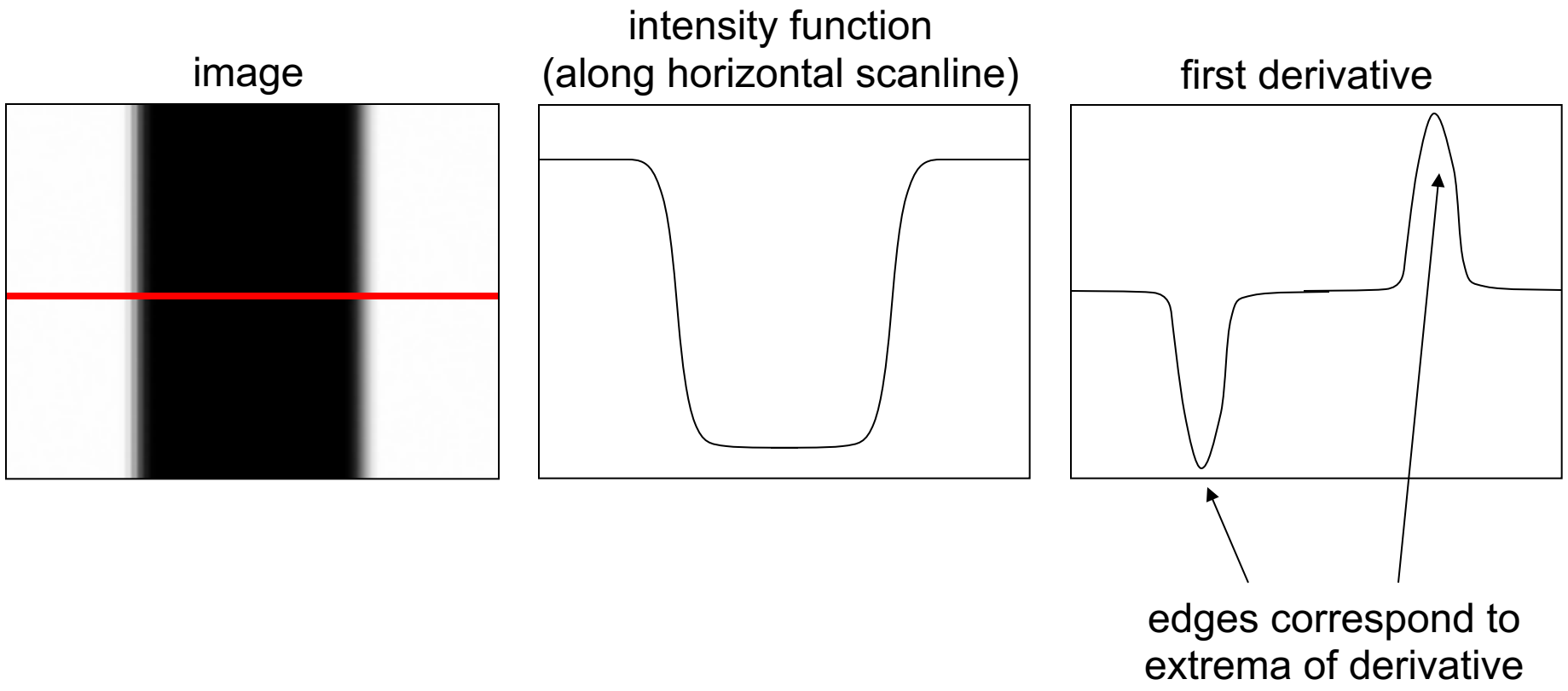
$$(f \star g)[m,\ n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,\ j] \cdot g[i-m,\ j-n]$$

Convolution with Impulse function

$$f[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l]\, \delta_2[n-k, m-l]$$

Characterizing edges

- An edge is a place of rapid change in the image intensity function

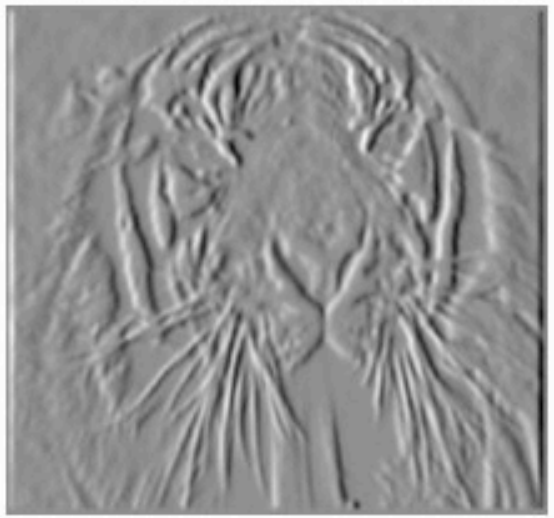


Finite differences: example

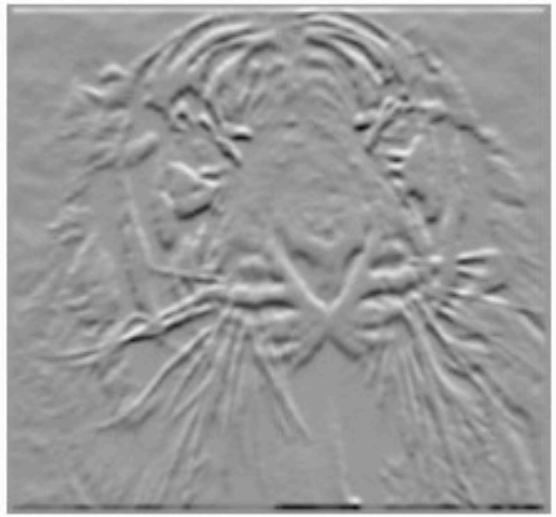
Original Image

Gradient magnitude

x-direction



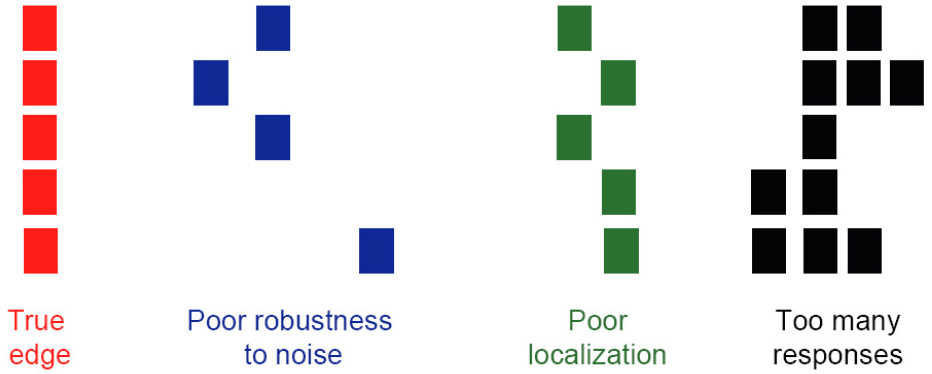
y-direction



- Which one is the gradient in the x-direction? How about y-direction?

Designing an edge detector

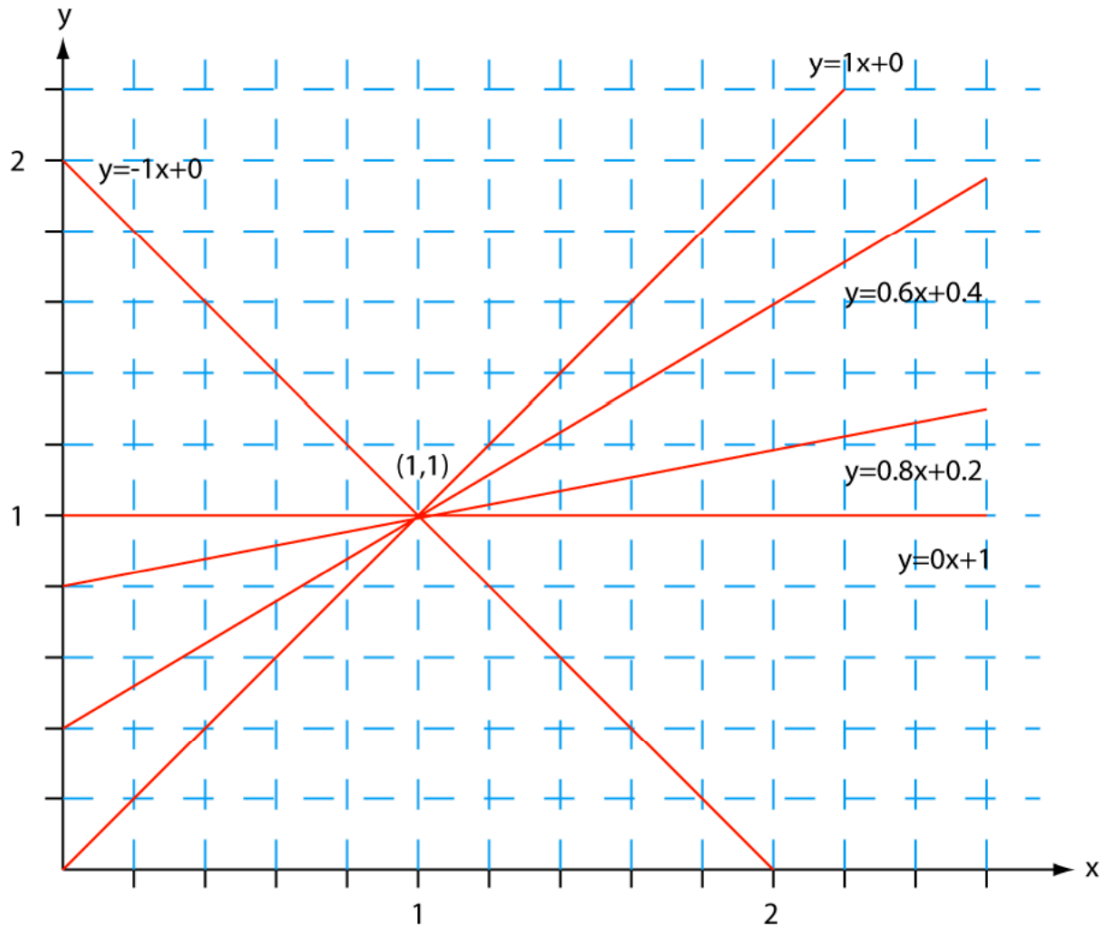
- Criteria for an "optimal" edge detector:
  - **Good detection:** the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)
  - **Good localization:** the edges detected must be as close as possible to the true edges
  - **Single response:** the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge



Canny edge detector

- Suppress Noise
- Compute gradient magnitude and direction
- Apply Non-Maximum Suppression
  - Assures minimal response
- Use hysteresis and connectivity analysis to detect edges

Detecting lines using Hough transform



RANSAC: Pros and Cons

- **Pros:**
  - General method suited for a wide range of model fitting problems
  - Easy to implement and easy to calculate its failure rate
- **Cons:**
  - Only handles a moderate percentage of outliers without cost blowing up
  - Many real problems have high rate of outliers (but sometimes selective choice of random subsets can help)
- A voting strategy, The Hough transform, can handle high percentage of outliers

Requirements for keypoint localization

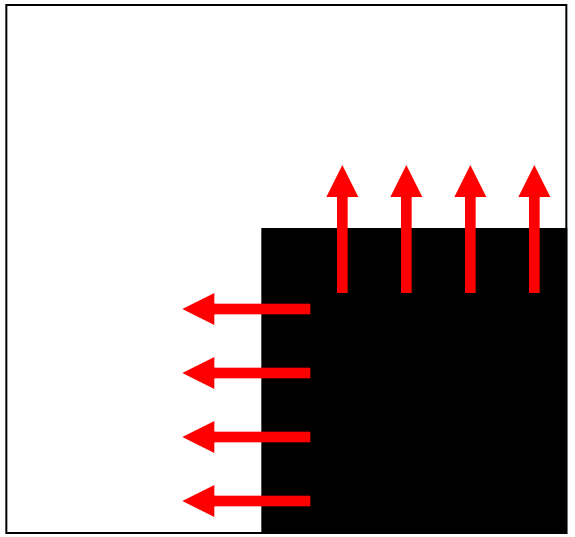
- Region extraction needs to be repeatable and accurate
  - Invariant to translation, rotation, scale changes
  - Robust or covariant to out-of-plane (≈affine) transformations
  - Robust to lighting variations, noise, blur, quantization
- Locality: Features are local, therefore robust to occlusion and clutter.
- Quantity: We need a sufficient number of regions to cover the object.
- Distinctivenes : The regions should contain "interesting" structure.
- Efficiency: Close to real-time performance.

Slide credit: Bastian Leibe

Harris corner detector and second moment matrix

- First, let’s consider an axis-aligned corner:

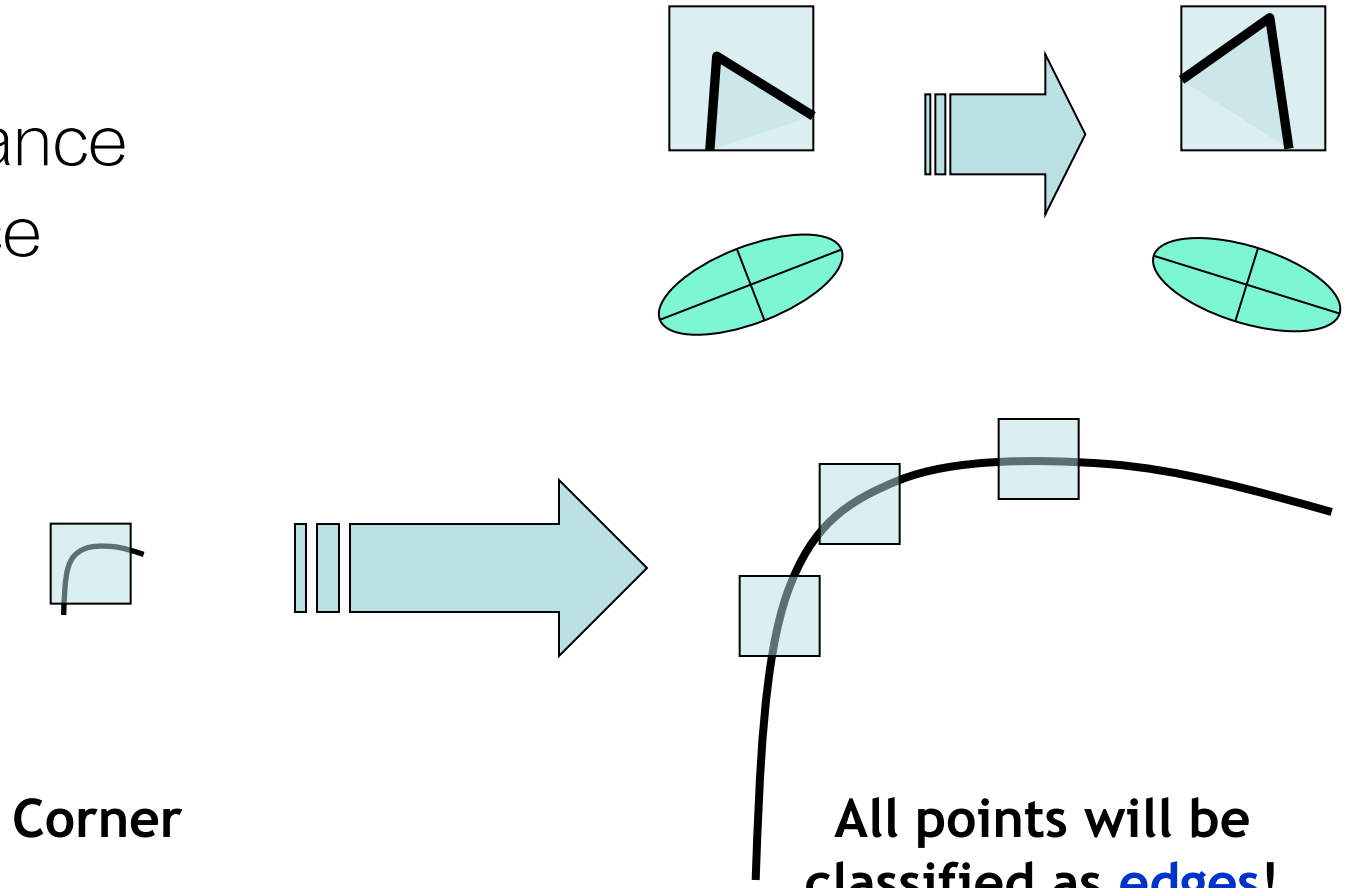
$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$



Slide credit: David Jacobs

Harris Detector: Properties

- Translation invariance
- Rotation invariance
- Scale invariance?

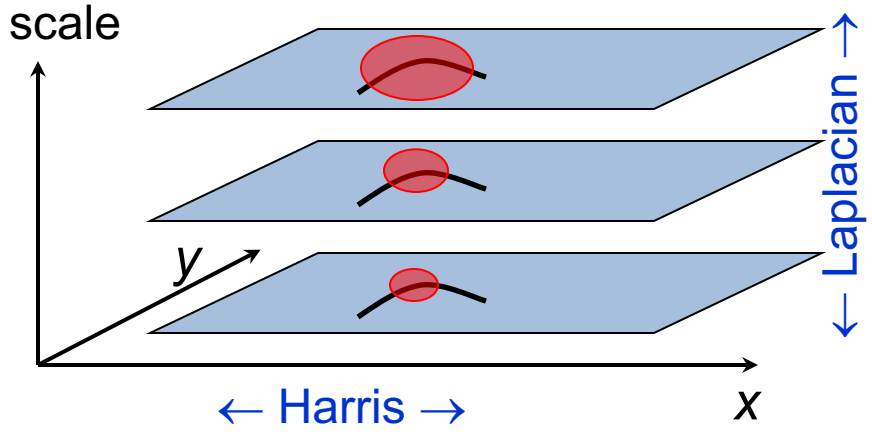


***Not* invariant to image scale!**

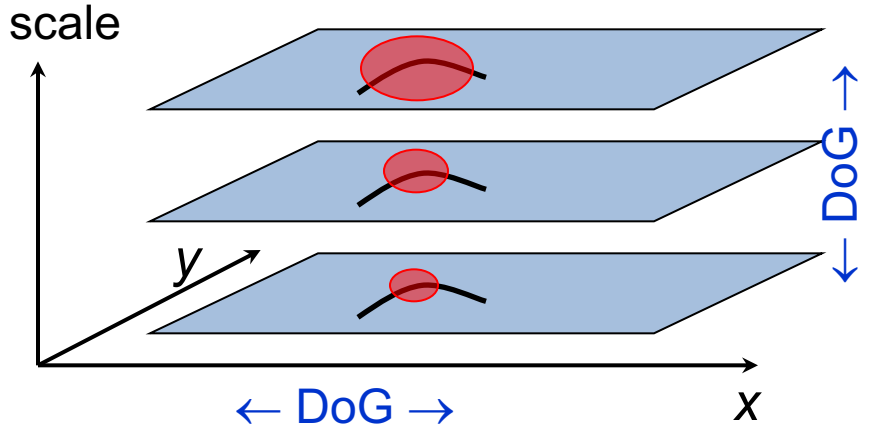
Slide credit: Kristen Grauman

Scale Invariant Detectors

- Harris-Laplacian[1]
  Find local maximum of:
  - Harris corner detector in space (image coordinates)
  - Laplacian in scale

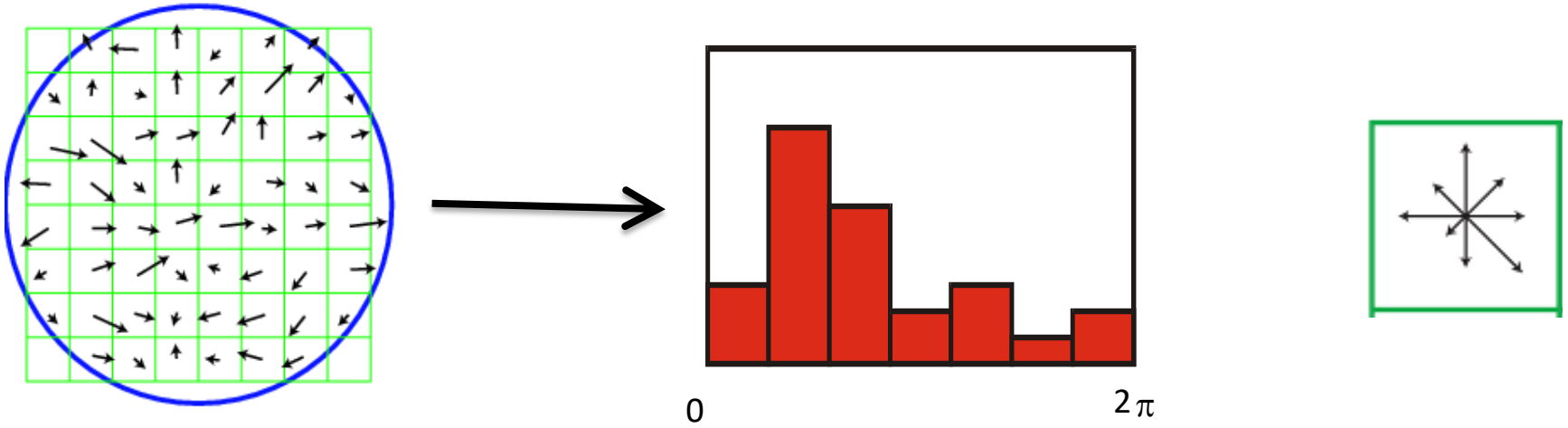


- SIFT (Lowe)[2]
  Find local maximum of:
  - Difference of Gaussians in space and scale



[1] K.Mikolajczyk, C.Schmid. "Indexing Based on Scale Invariant Interest Points". ICCV 2001
[2] D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". IJCV 2004

SIFT descriptor formation



- Using precise gradient locations is fragile. We'd like to allow some "slop" in the image, and still produce a very similar descriptor
- Create array of orientation histograms (a 4x4 array is shown)
- Put the rotated gradients into their local orientation histograms
  - A gradients's contribution is divided among the nearby histograms based on distance. If it's halfway between two histogram locations, it gives a half contribution to both.
  - Also, scale down gradient contributions for gradients far from the center
- The SIFT authors found that best results were with 8 orientation bins per histogram.

Difference between HoG and SIFT

- HoG is usually used to describe entire images. SIFT is used for key point matching
- SIFT histrograms are oriented towards the dominant gradient. HoG is not.
- HoG gradients are normalized using neighborhood bins.
- SIFT descriptors use varying scales to compute multiple descriptors.

Seam Carving

- Assume m x n → m x n', n'<n (summarization)

- Basic Idea: remove unimportant pixels from the image
  - Unimportant = pixels with less "energy"

$$E_1(\mathbf{I}) = |\frac{\partial}{\partial x}\mathbf{I}| + |\frac{\partial}{\partial y}\mathbf{I}|.$$

- Intuition for gradient-based energy:
  - Preserve strong contours
  - Human vision more sensitive to edges - so try remove content from smoother areas
  - Simple enough for producing some nice results

Gestalt Factors

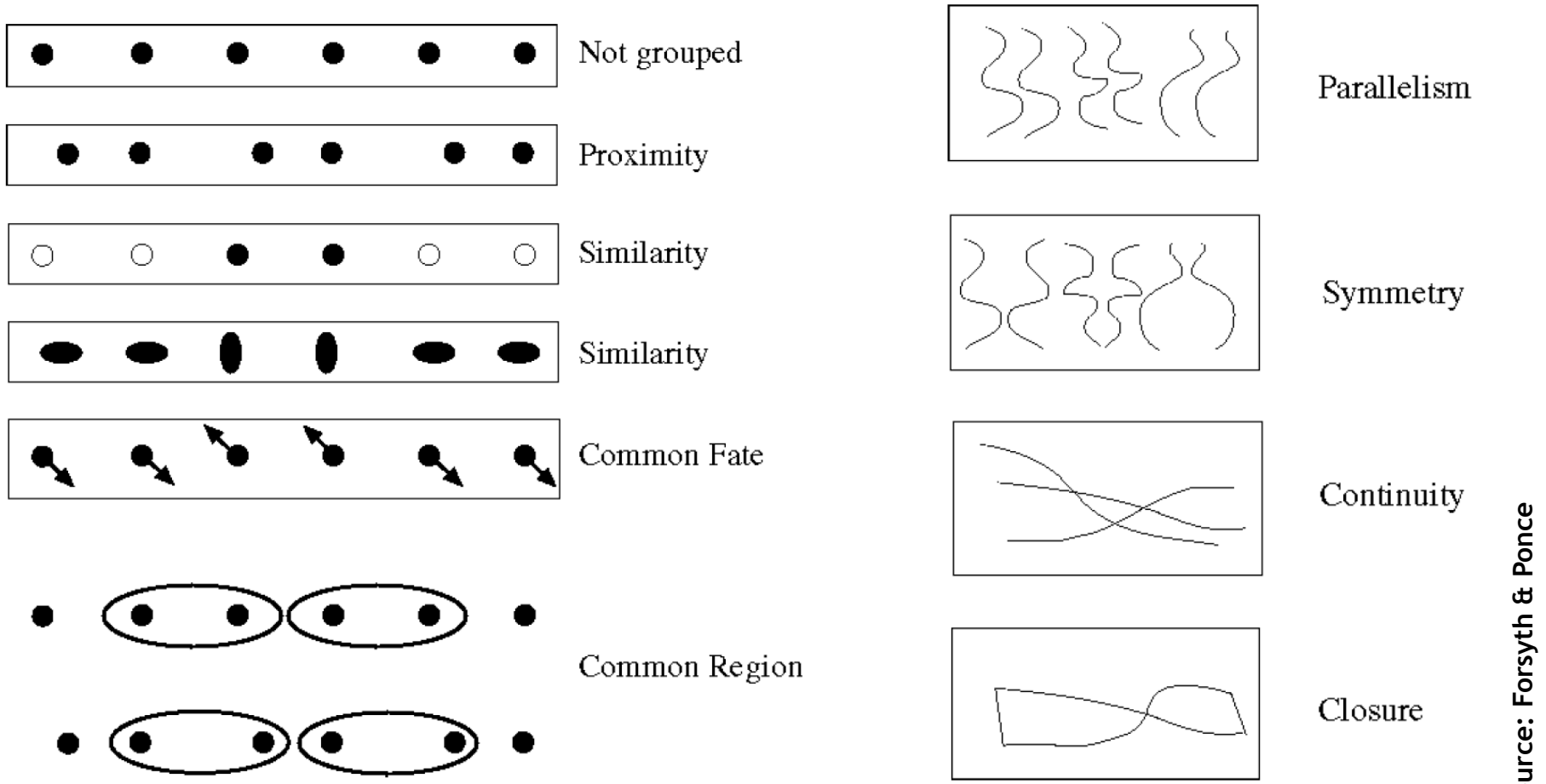


Image source: Forsyth & Ponce

- These factors make intuitive sense, but are very difficult to translate into algorithms

Conclusions: Agglomerative Clustering

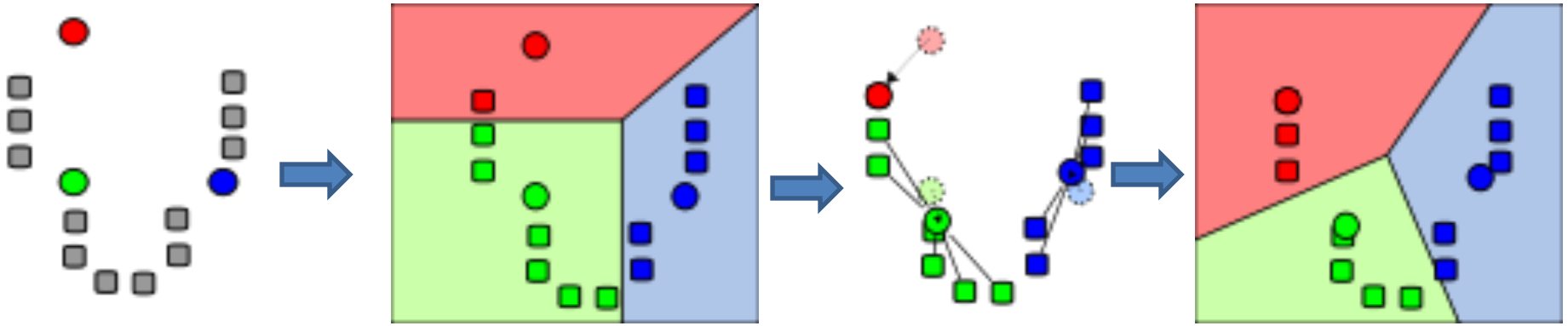
Good

- Simple to implement, widespread application.
- Clusters have adaptive shapes.
- Provides a hierarchy of clusters.
- No need to specify number of clusters in advance.

Bad

- May have imbalanced clusters.
- Still have to choose number of clusters or threshold.
- Does not scale well. Runtime of $O(n^3)$.
- Can get stuck at a local optima.

K-means clustering



1. Initialize Cluster Centers

2. Assign Points to Clusters

3. Re-compute Means

Repeat (2) and (3)

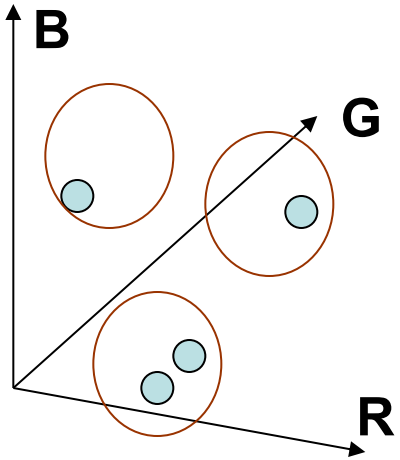
- Java demo:

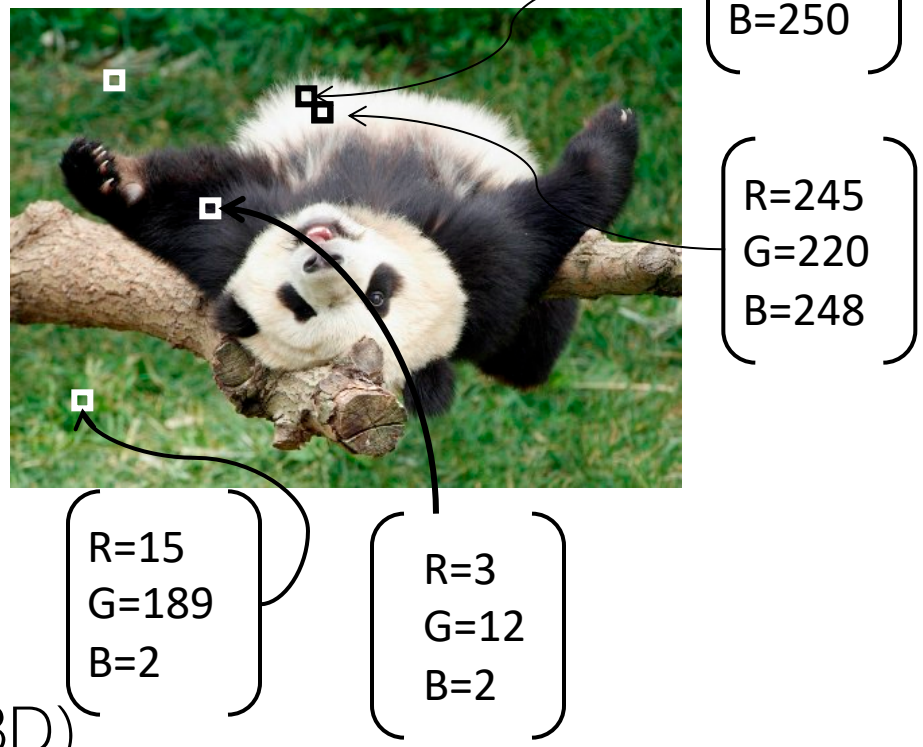
http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

Illustration Source: wikipedia

Feature Space

- Depending on what we choose as the *feature space*, we can group pixels in different ways.
- Grouping pixels based on color similarity



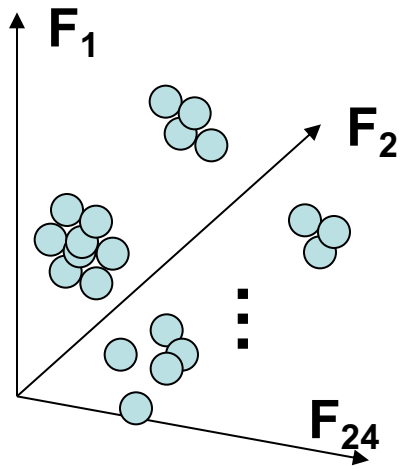


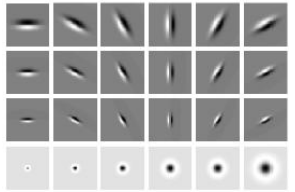
- Feature space: color value (3D)

Slide credit: Kristen Grauman

Feature Space

- Depending on what we choose as the *feature space*, we can group pixels in different ways.
- Grouping pixels based on texture similarity





Filter bank of 24 filters

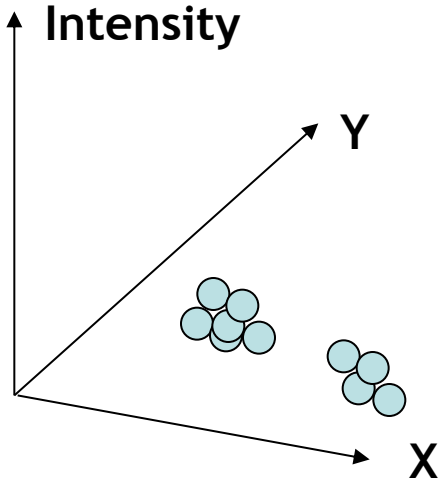
- Feature space: filter bank responses (e.g., 24D)

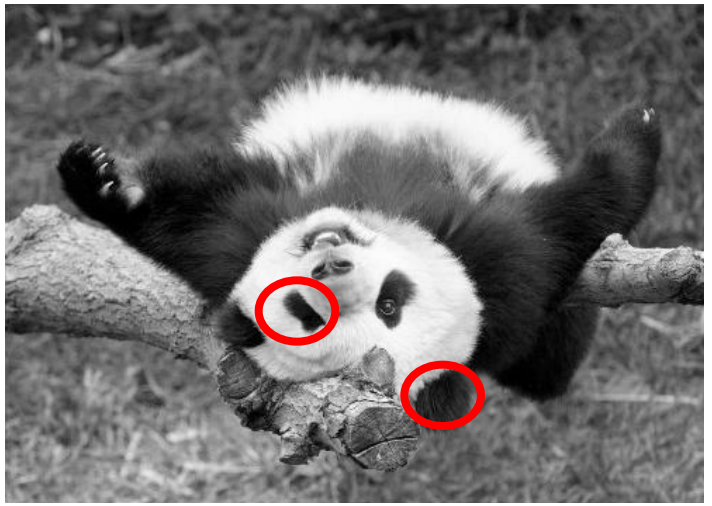
Slide credit: Kristen Grauman

Segmentation as Clustering

- Depending on what we choose as the *feature space*, we can group pixels in different ways.

- Grouping pixels based on *intensity+position* similarity



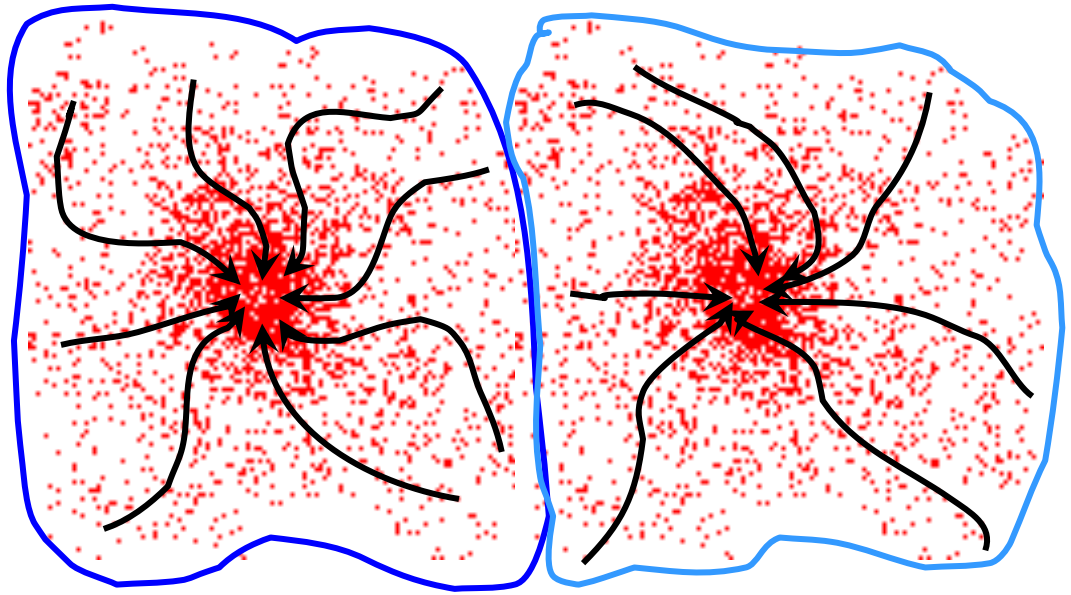


⇒ Way to encode both *similarity* and *proximity.*

Slide credit: Kristen Grauman

Mean-Shift Clustering

- Cluster: all data points in the attraction basin of a mode
- Attraction basin: the region for which all trajectories lead to the same mode



Slide by Y. Ukrainitz & B. Sarel

Summary Mean-Shift

- Pros
  - General, application-independent tool
  - Model-free, does not assume any prior shape (spherical, elliptical, etc.) on data clusters
  - Just a single parameter (window size h)
    - h has a physical meaning (unlike k-means)
  - Finds variable number of modes
  - Robust to outliers
- Cons
  - Output depends on window size
  - Window size (bandwidth) selection is not trivial
  - Computationally (relatively) expensive (~2s/image)
  - Does not scale well with dimension of feature space

The machine learning framework

$$y = f(\mathbf{x})$$

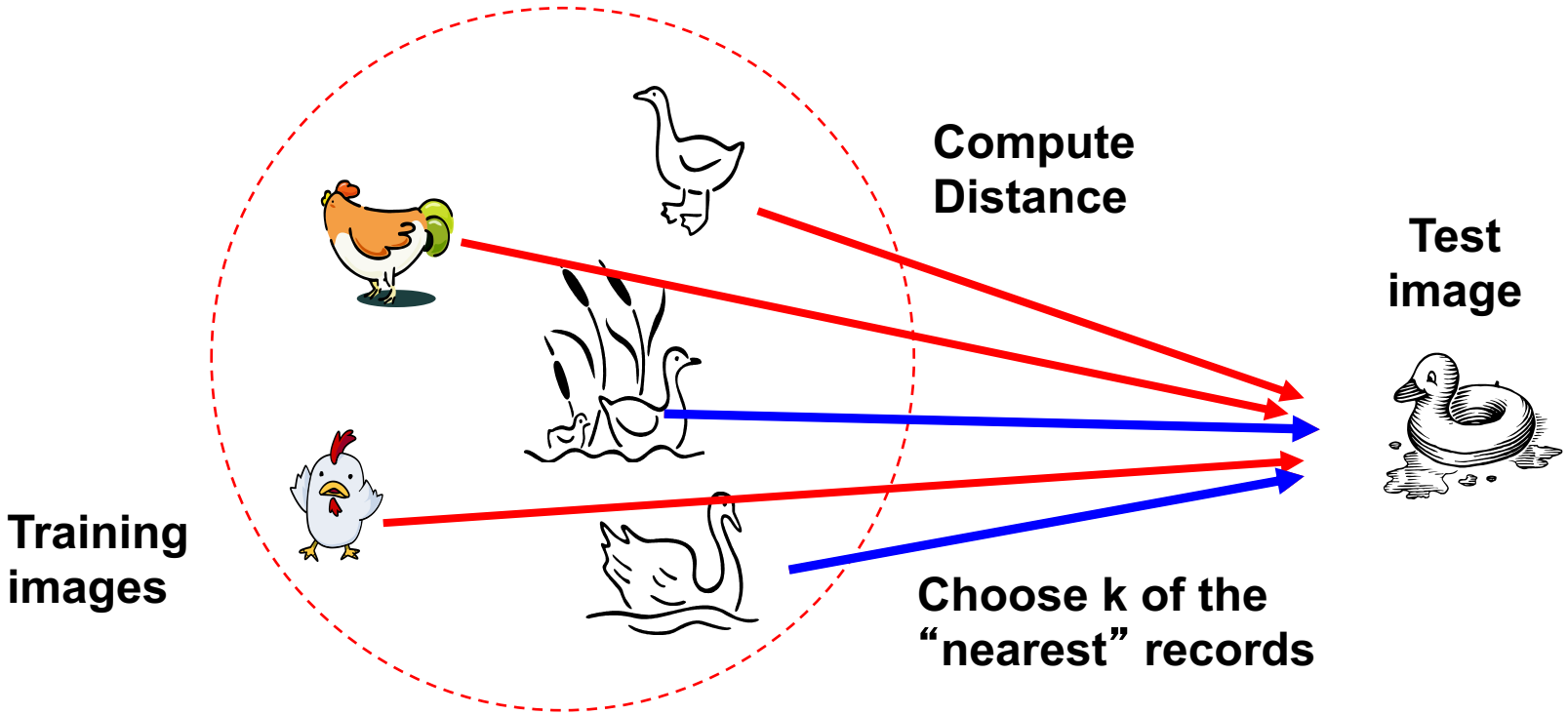
output | prediction function | Image feature

- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$

Slide credit: L. Lazebnik

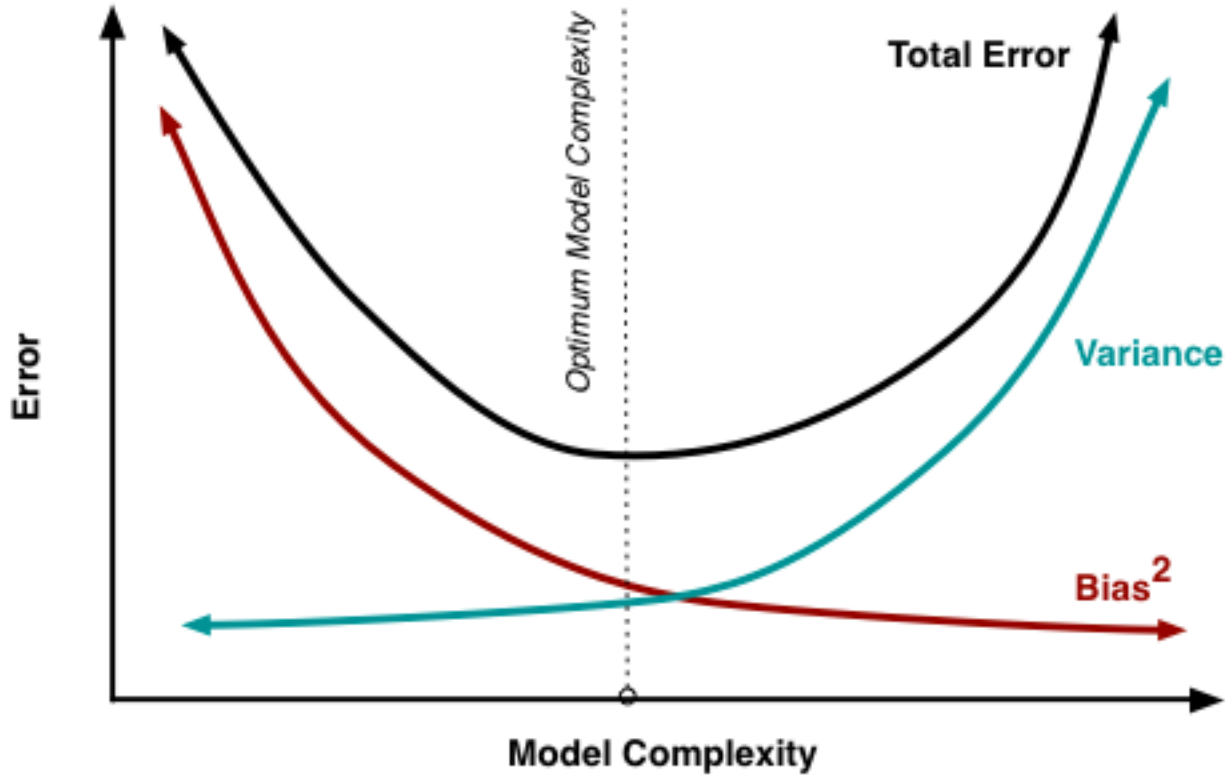
Nearest Neighbor Classifier

- Assign label of nearest training data point to each test data point



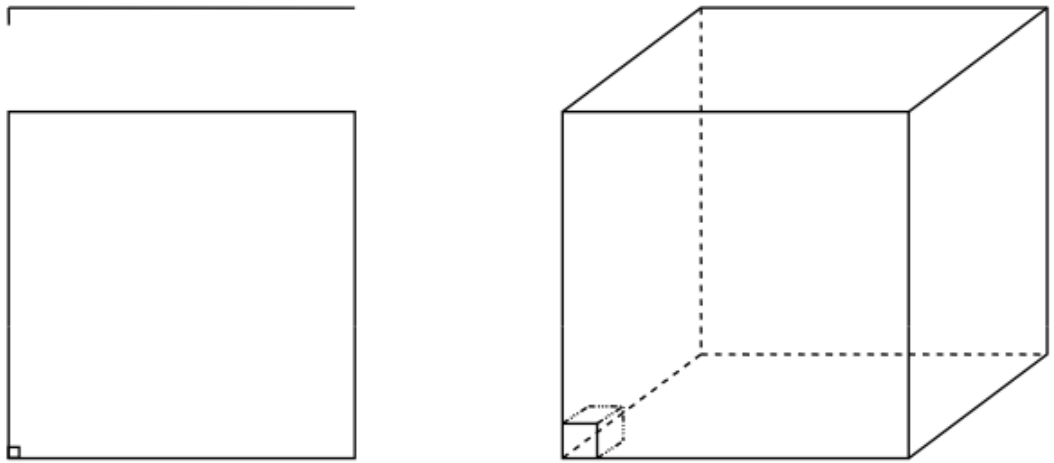
Source: N. Goyal

Bias versus variance trade off

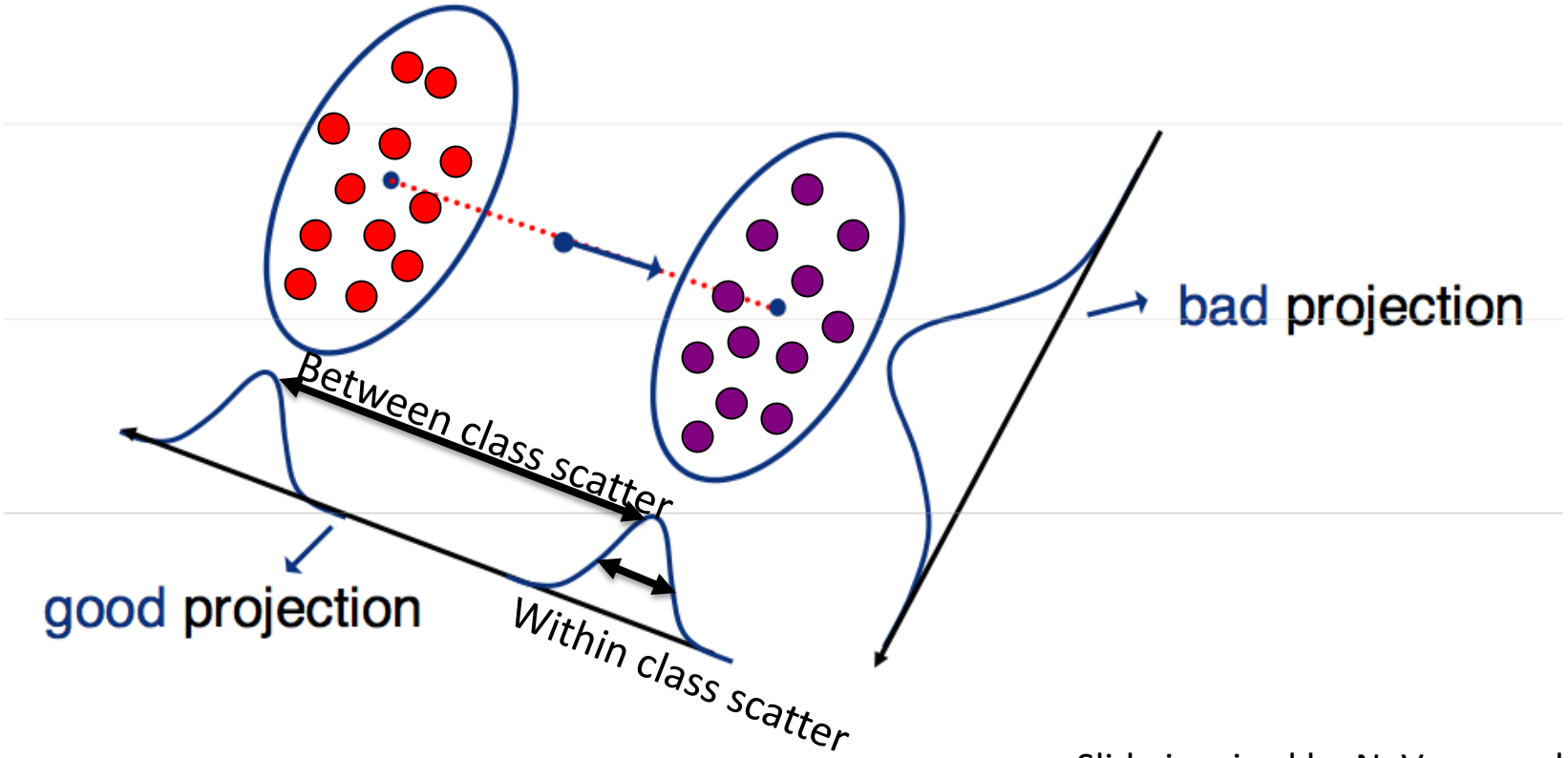


Curse of dimensionality

- Assume 5000 points uniformly distributed in the unit hypercube and we want to apply 5−NN. Suppose our query point is at the origin.
  - In 1−dimension, we must go a distance of 5/5000=0.001 on the average to capture 5 nearest neighbors.
  - In 2 dimensions, we must go $\sqrt{0.001}$ to get a square that contains 0.001 of the volume.
  - In d dimensions, we must go $(0.001)^{1/d}$



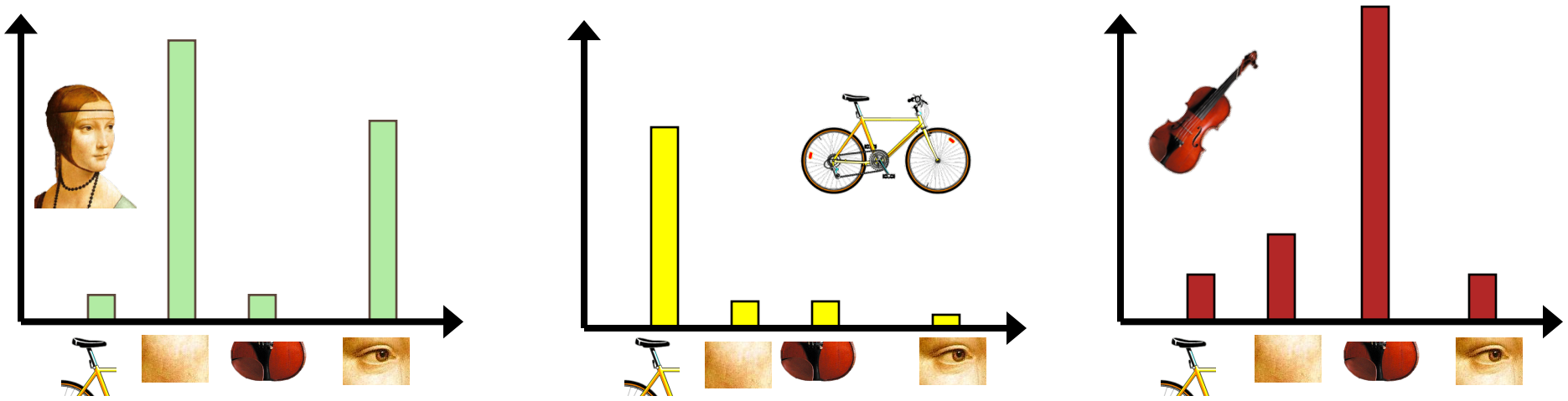
Fischer's Linear Discriminant Analysis



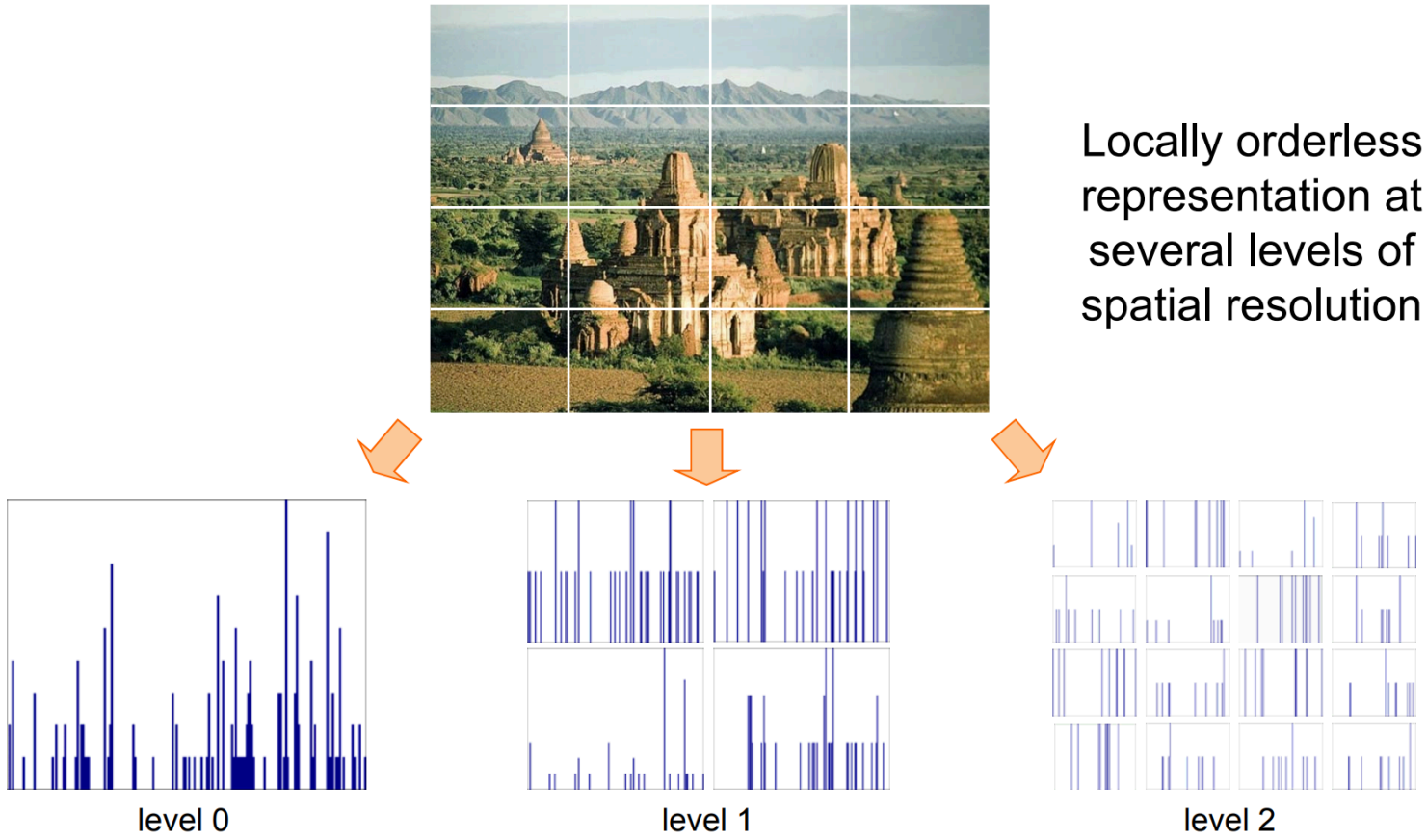
Slide inspired by N. Vasconcelos

Bag of features: outline

1. Extract features
2. Learn “visual vocabulary”
3. Quantize features using visual vocabulary
4. Represent images by frequencies of “visual words”

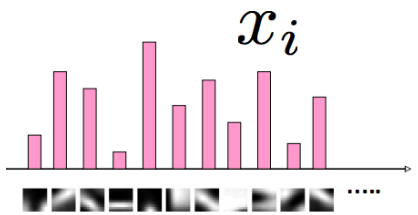


Bag of words + pyramids



Naïve Bayes

- Classify image using histograms of occurrences on visual words:

$$\mathbf{x} =$$ 

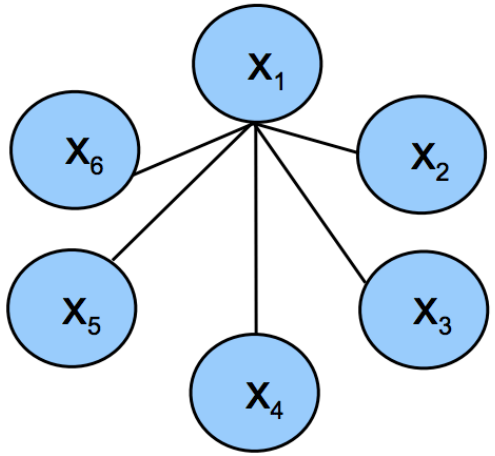
- if only present/absence of a word is taken into account:
- Naïve Bayes classifier assumes that visual words are conditionally independent given object class

$x_i \in \{0, 1\}$

Csurka Bray, Dance & Fan, 2004

Detecting a person with their parts

- For example, a person can be modelled as having a head, left arm, right arm, etc.
- All parts can be modelled relative to the global person detector

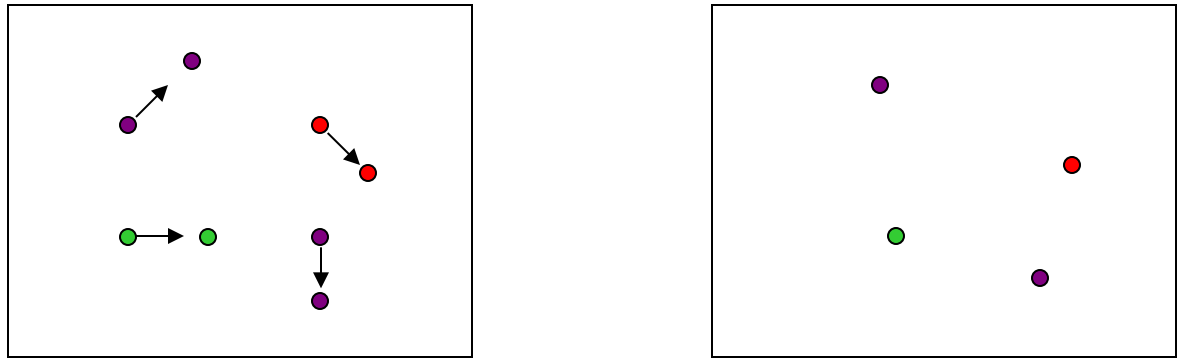


Fine-Grained Recognition

What breed is this dog?

Key: Find the right features.

Estimating optical flow



$I(x,y,t-1)$ $I(x,y,t)$

- Given two subsequent frames, estimate the apparent motion field u(x,y), v(x,y) between them

- Key assumptions
  - **Brightness constancy:** projection of the same point looks the same in every frame
  - **Small motion:** points do not move very far
  - **Spatial coherence:** points move like their neighbors

Source: Silvio Savarese

Lucas Kande optical flow

– Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A$ $\qquad$ $A^T b$

When is This Solvable?

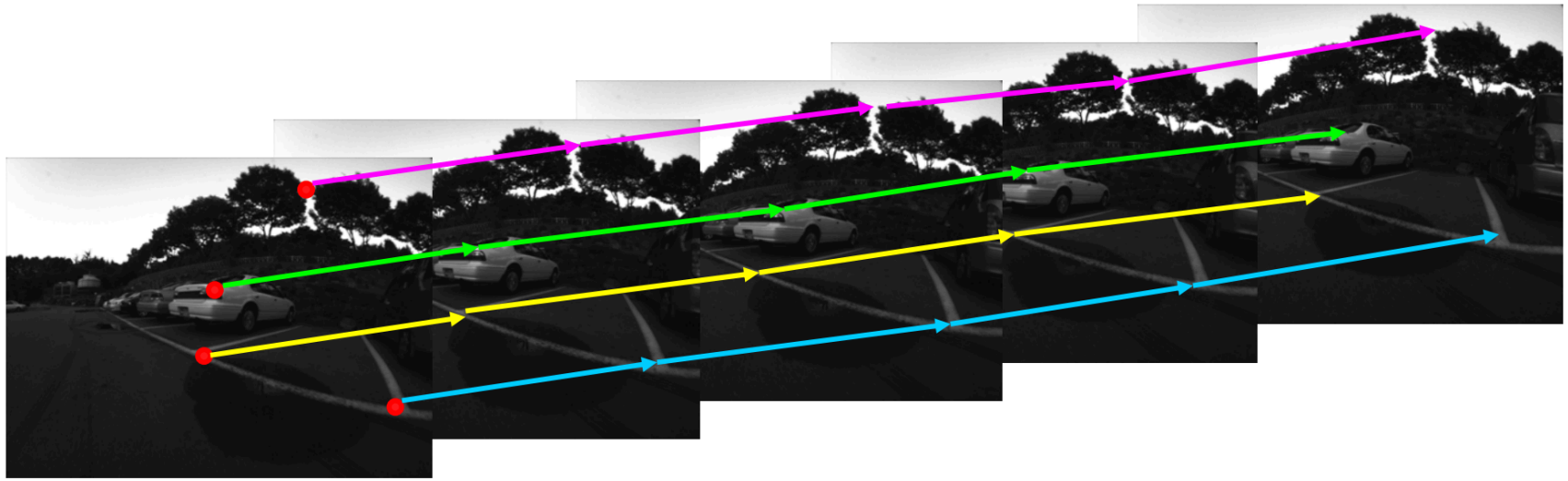
- **A^TA** should be invertible
- **A^TA** should not be too small due to noise
  - eigenvalues λ_1 and λ_2 of **A^TA** should not be too small
- **A^TA** should be well-conditioned
  - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

Does this remind anything to you?

Source: Silvio Savarese

Tracking

Feature point tracking



Slide credit: Yonsei Univ.

Today's agenda

- Backprop in neural networks
- Convolutional neural networks
- Architecture design
- Exam and brief class overview
- En fin

What should you take away from this class?

- A broad understanding of computer vision as a field.
- Learning to use common software packages: github, jupyter, numpy, scipy.
- Converting ideas into mathematical equations.
- Converting mathematical equations into code.
- Learning to communicate ideas and algorithms in formal writing.

What should you take away from this class?

Pixels	Segments	Images	Videos	Web
Convolutions Edges Descriptors	Resizing Segmentation Clustering	Recognition Detection Ontology	Motion Tracking	Neural networks Convolutional neural networks

Feedback – it matters a lot

- Feel free to email me with any personal feedback.
  - I am always trying to improve.

- Fill out anonymous class evaluations on axess.stanford.edu

Feedback – it matters a lot!

- Assignments
  - Which assignments were too easy? Too hard?
- Topics
  - Which topics did you enjoy the most and would recommend we keep and which ones did we not cover that you would like to include?
- Class notes and extra credit
  - Did you find them to be educational?
- Lectures
  - Were they engaging? How can we improve them?

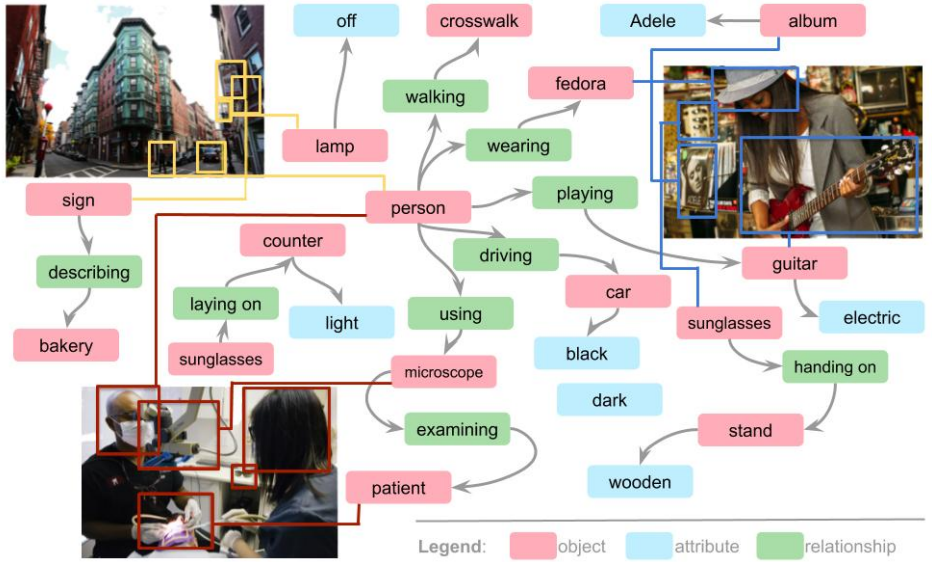
How to stay involved in Computer Vision?

- Join us to do research!!
  - We expect 20 hours of research a week
  - Positions are open for credit starting winter quarter (if you are undergrad student)
  - Positions are open both for credit and assistantship starting winter quarter (if you are a graduate student)
  - Paid summer research internship opportunities are also available

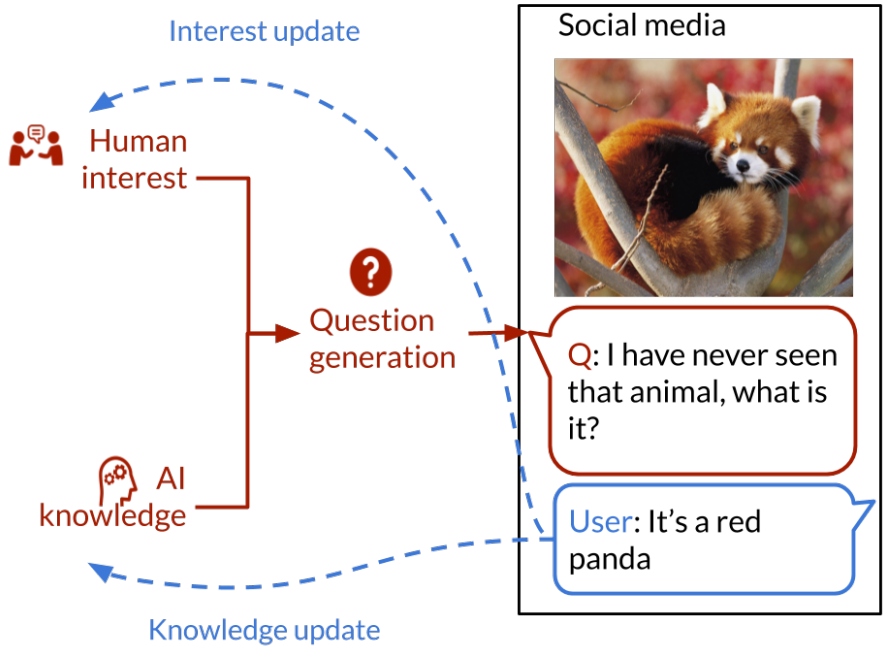
- **Juan Carlos** focuses on visual recognition and understanding of human actions and activities, objects, scenes, and events.
  - Check out his one hour talk on his research here: http://tv.vera.com.uy/video/55276

How to stay involved in Computer Vision?

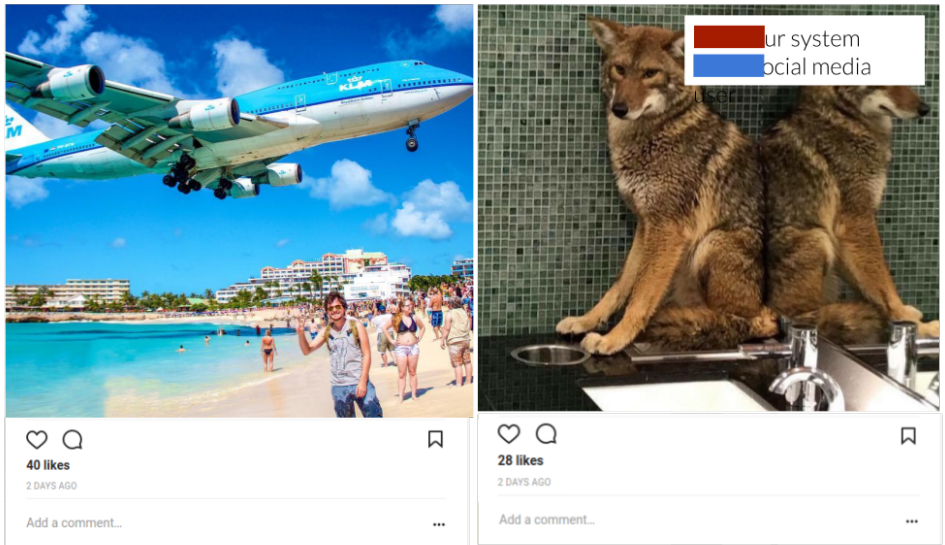
- **Ranjay** focuses on building Intelligence systems that grow their visual knowledge by interacting with and learning directly from people.



High level learning framework



Example interactions with real social media users



Where to go from here?

Still lots of open questions!

- Machine Learning (CS 229) (to learn the fundamentals of ML)
- 3D Computer Vision (CS 231A) (Silvio is on sabbatical this year)
- Convolutional Neural Networks (CS 231n) (I will be teaching it in the spring with Fei-Fei and Danfei)

- Mobile Computer Vision (CS 231M) (not offered every year)
- Representation Learning (CS 331B) (not offered every year)
- Advanced Computer Vision (CS 331A) (not offered every year)

CS 131 Computer Vision: Foundations and Applications

Juan Carlos Niebles and Ranjay Krishna

Sasha Harrison, Maxime Voisin, Brent Yi, Boxiao Pan

Stanford Vision and Learning Lab