Lecture 11: Recognition

Instructor: Roni Sengupta

ULA: Andrea Dunn, William Li, Liujie Zheng

Course Website: Scan Me!

What is “Recognition”?

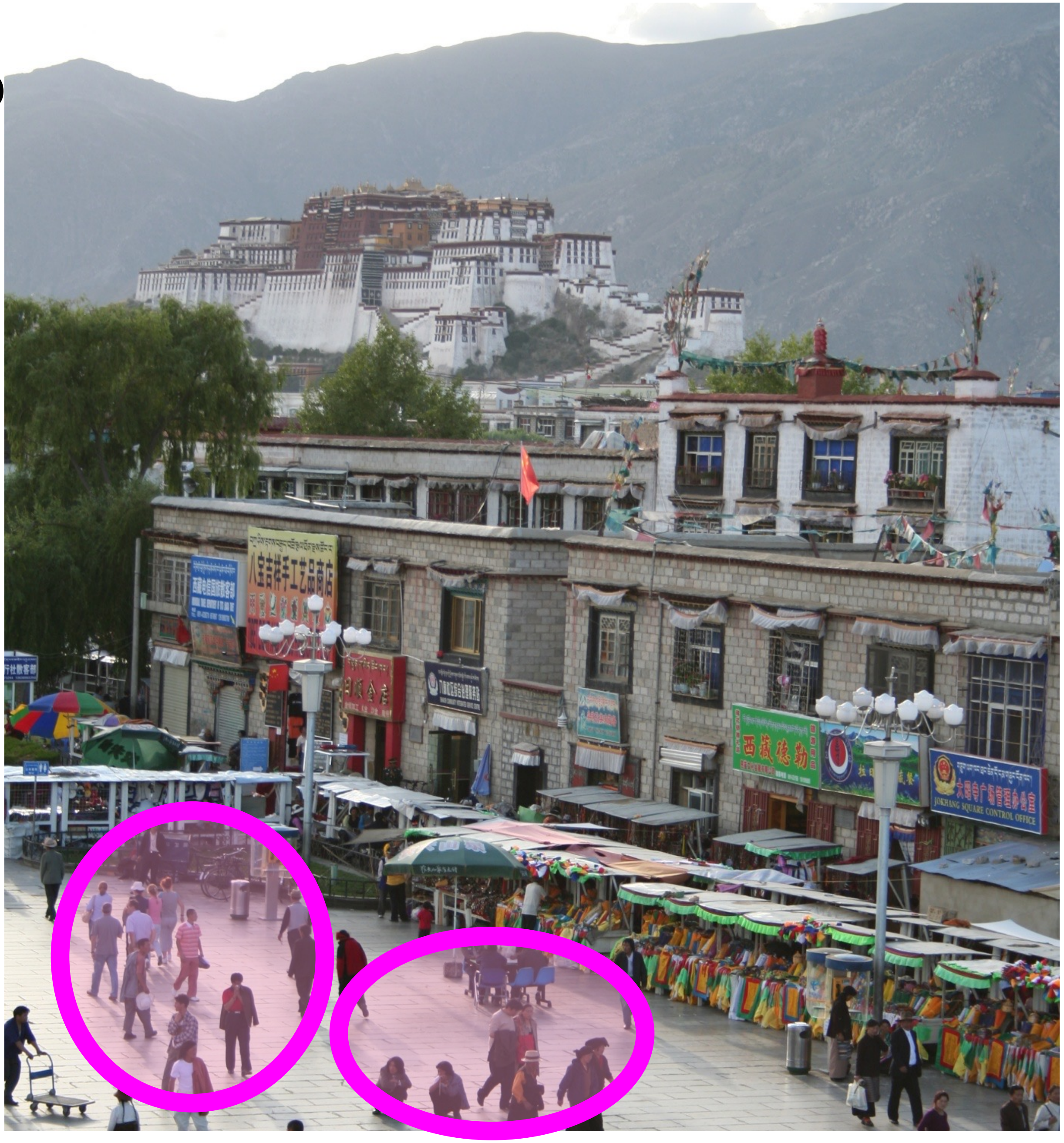
Next few slides adapted from Li, Fergus, & Torralba’s excellent short course on category and object recognition

What is “Recognition”?

- Verification: is that a lamp?

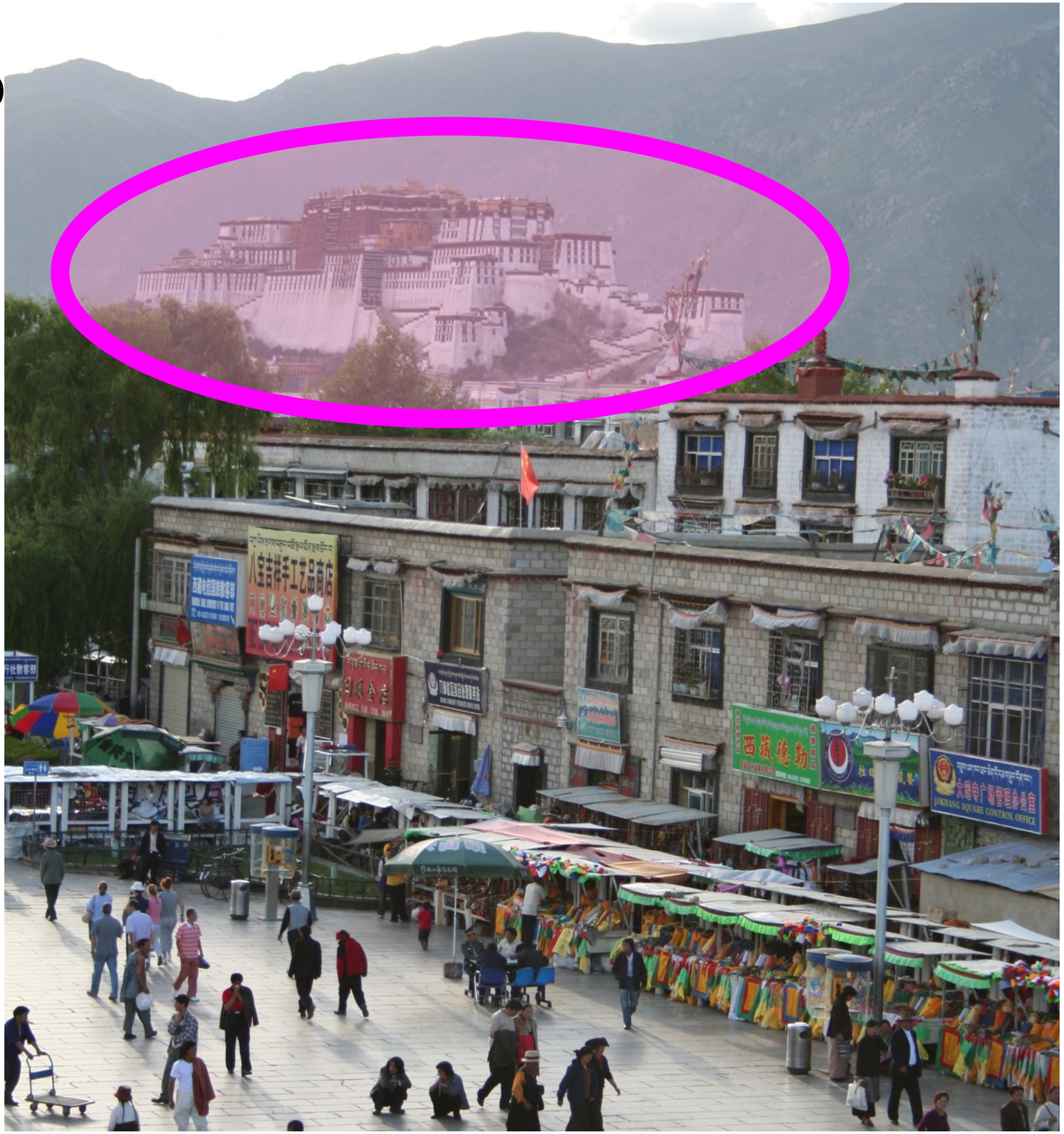
What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?



What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?

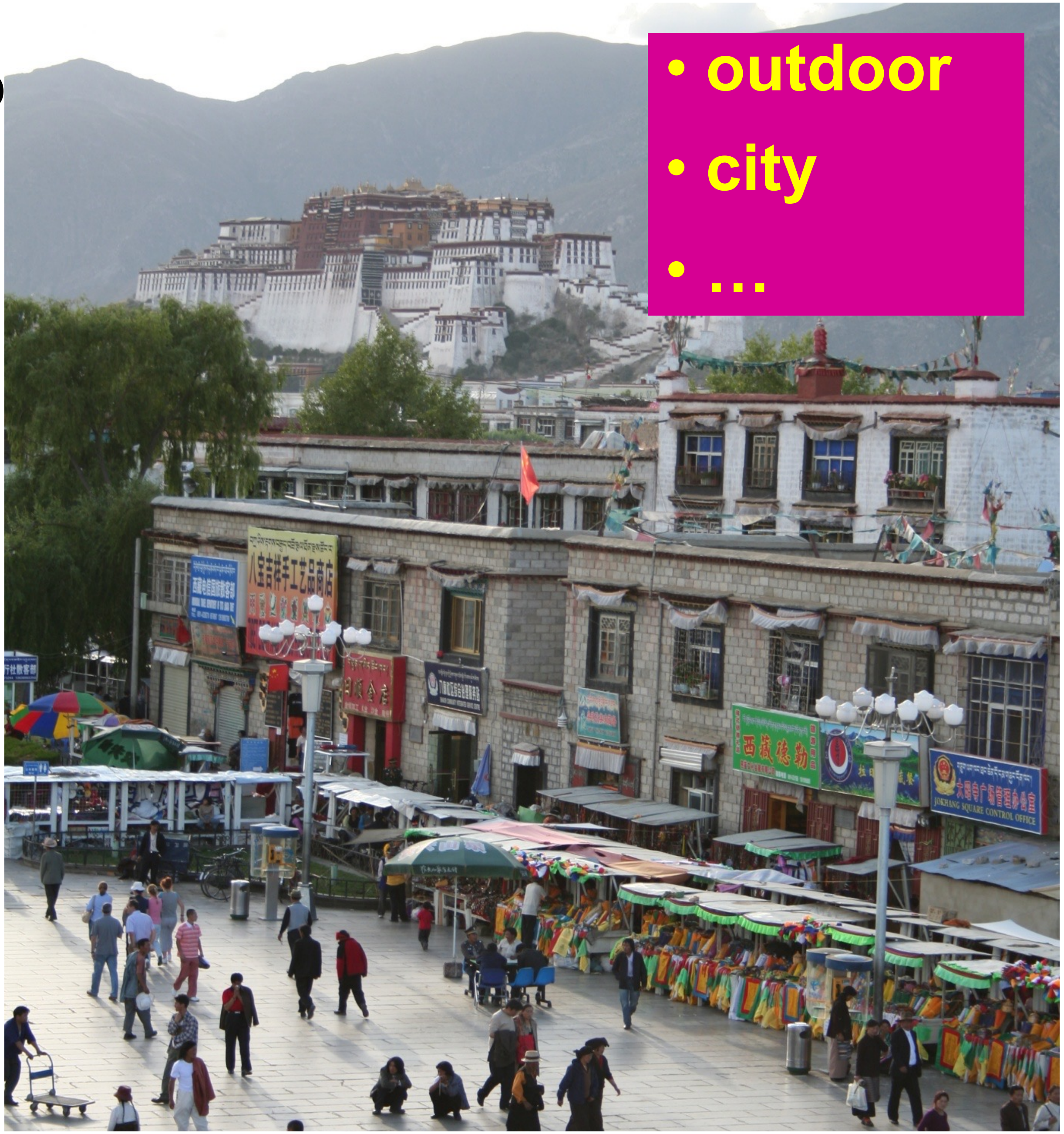


What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?
- Object categorization

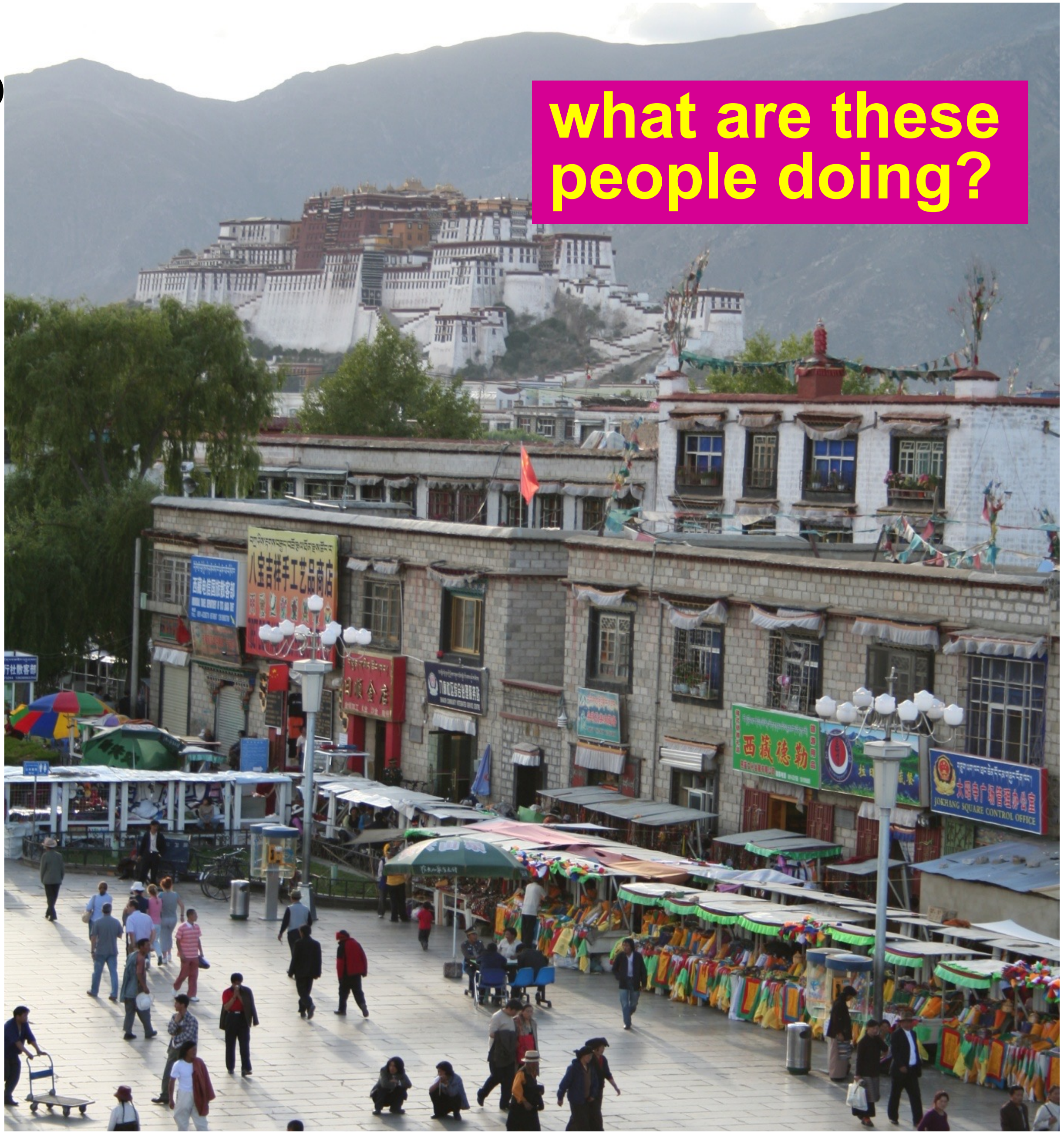
What is "Recognition"?

- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?
- Object categorization
- Scene and context categorization



What is “Recognition”?

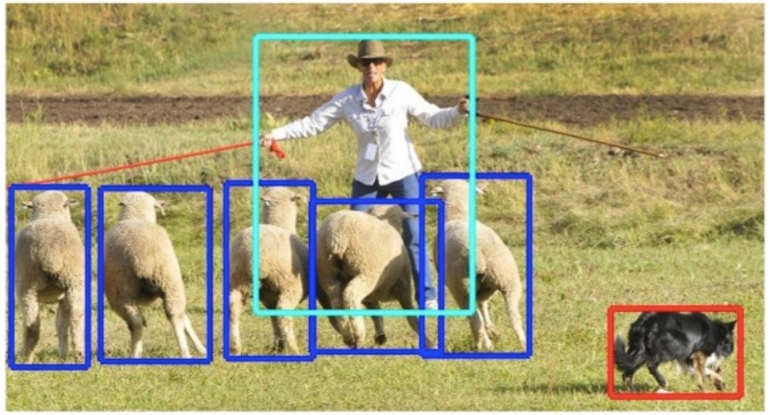
- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?
- Object categorization
- Scene and context categorization
- Activity / Event Recognition



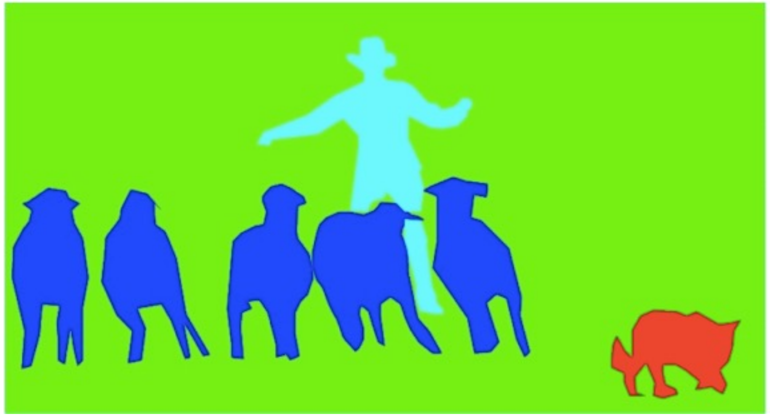
Recognition: What type of output?

Image classification

Object detection



Semantic segmentation



Instance segmentation

Image Classification:
a core task in computer vision

- Assume given set of discrete labels, e.g.

  {cat, dog, cow, apple, tomato, truck, … }

f() = "apple"

f() = "tomato"

f() = "cow"

Dataset: ETH-80, by B. Leibe   Slide credit: L. Lazebnik

What Matters in Recognition?

- Data
  - More is always better (as long as it is good data)
  - Annotation is the hard part

- Representation
  - Low level: SIFT, HoG, GIST, edges
  - Mid level: Bag of words, sliding window, deformable model
  - High level: Contextual dependence
  - Deep learned features

- Learning Techniques
  - E.g. choice of classifier or inference method

What Matters in Recognition?

- Data
  - More is always better (as long as it is good data)
  - Annotation is the hard part

- Representation
  - Low level: SIFT, HoG, GIST, edges
  - Mid level: Bag of words, sliding window, deformable model
  - High level: Contextual dependence
  - Deep learned features

- Learning Techniques
  - E.g. choice of classifier or inference method

Data Sets

- PASCAL VOC
  - *Not* Crowdsourced, bounding boxes, 20 categories
- **ImageNet**
  - **Huge, Crowdsourced, Hierarchical, *Iconic* objects**
- SUN Scene Database, Places
  - *Not* Crowdsourced, 397 (or 720) scene categories
- LabelMe (Overlaps with SUN)
  - Sort of Crowdsourced, Segmentations, Open ended
- SUN *Attribute* database (Overlaps with SUN)
  - Crowdsourced, 102 attributes for every scene
- OpenSurfaces
  - Crowdsourced, materials
- **Microsoft COCO**
  - **Crowdsourced, large-scale objects**

… and many more https://paperswithcode.com/datasets?task=image-classification

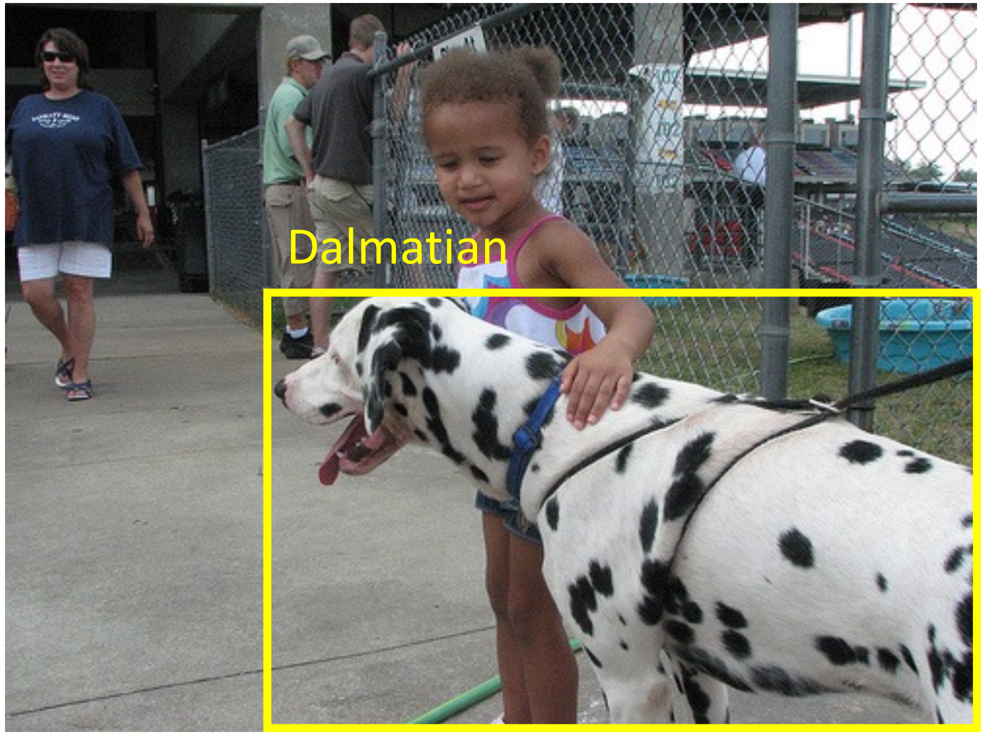
Large Scale Visual Recognition Challenge (ILSVRC)

IMAGENET

2010-2017

~~20 object classes~~ ~~22,591 images~~

1000 object classes **1,431,167 images**



http://image-net.org/challenges/LSVRC/{2010,2011,2012}

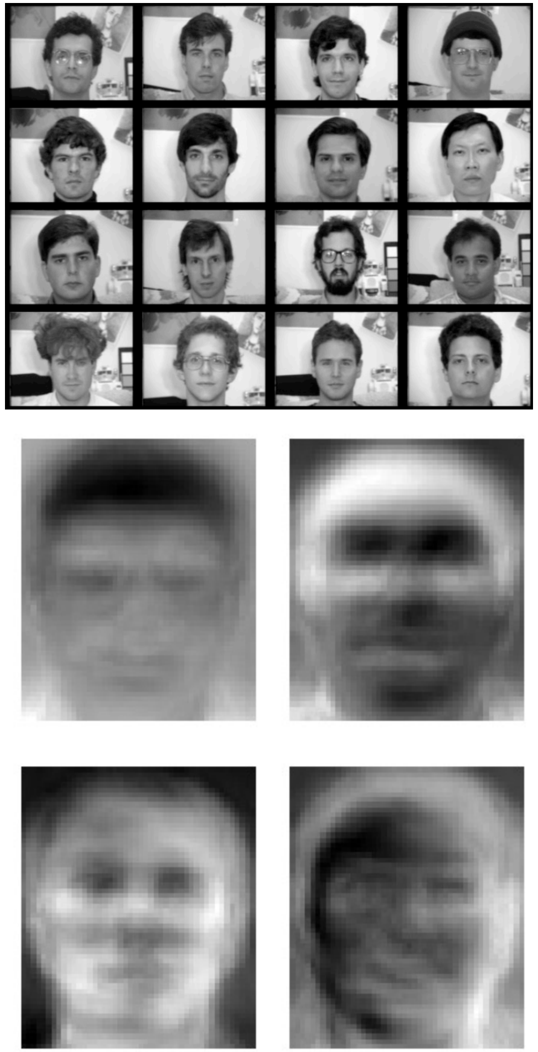
What Matters in Recognition?

- Data
  - More is always better (as long as it is good data)
  - Annotation is the hard part

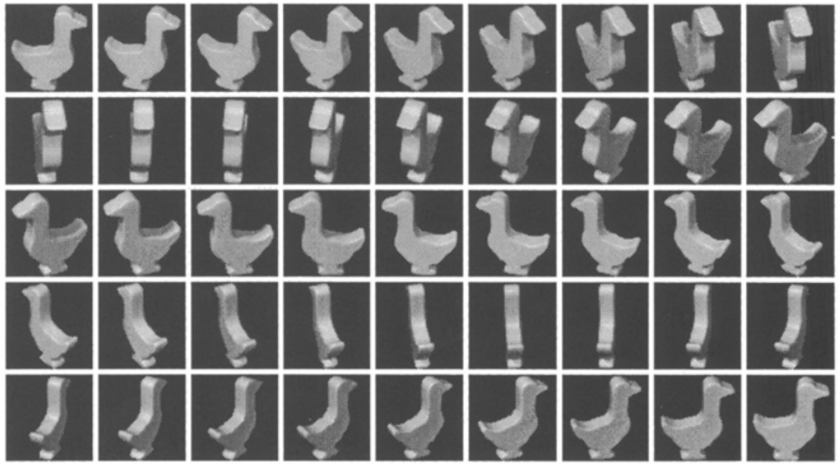
- Representation
  - **Low level: SIFT, HoG, GIST, edges**
  - **Mid level: Bag of words, sliding window, deformable model**
  - **High level: Contextual dependence**
  - Deep learned features

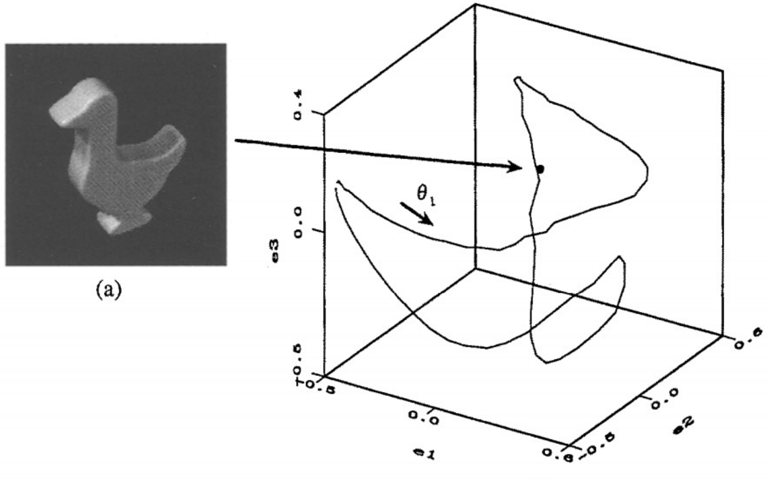
- Learning Techniques
  - E.g. choice of classifier or inference method

History of recognition: Appearance-based models



M. Turk and A. Pentland, [Face recognition using eigenfaces](), CVPR 1991



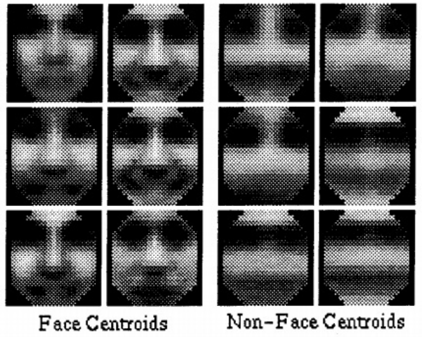


(a) (b)

H. Murase and S. Nayar, [Visual learning and recognition of 3-d objects from appearance](), IJCV 1995

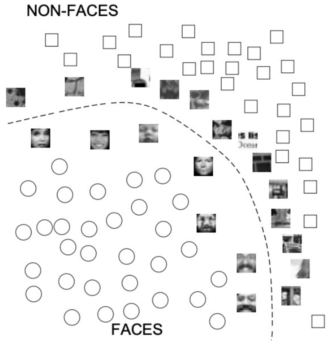
History of recognition: Features and classifiers

Appearance manifolds
+ neural network



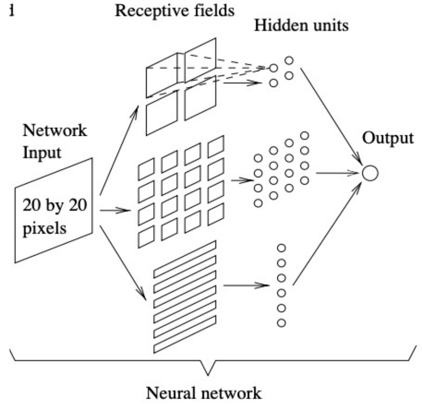
Sung & Poggio (1994)

Support vector machines



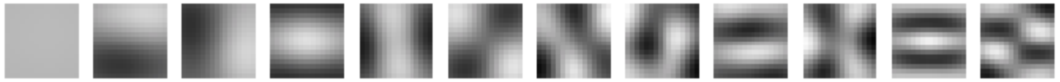
Osuna, Freund, Girosi (1997)

Neural network



Rowley, Baluja, Kanade (1998)

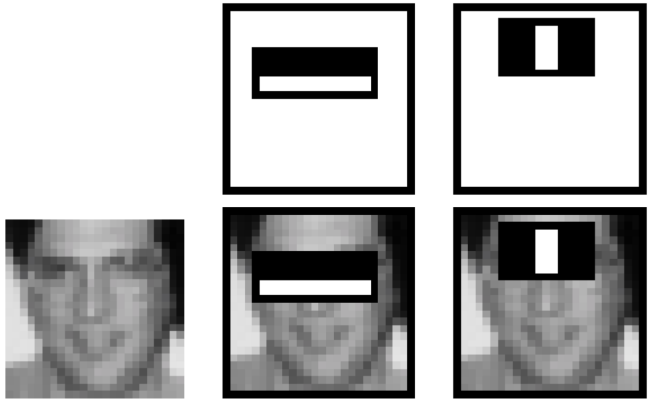
Statistics of feature responses, probabilistic classifier



$$\prod_{j=1}^{n_{magn}}\prod_{i=1}^{n_{subs}} \frac{P(q1_i^j|\text{object})P(pos_i^{j'}|q2_i,\text{object})}{\frac{P(q1_i^j|\overline{\text{object}})}{n_{subs}}} \underset{\overline{\text{object}}}{\overset{\text{object}}{\gtrless}} \lambda = \frac{P(\overline{\text{object}})}{P(\text{object})}$$

Schneiderman & Kanade (1998)

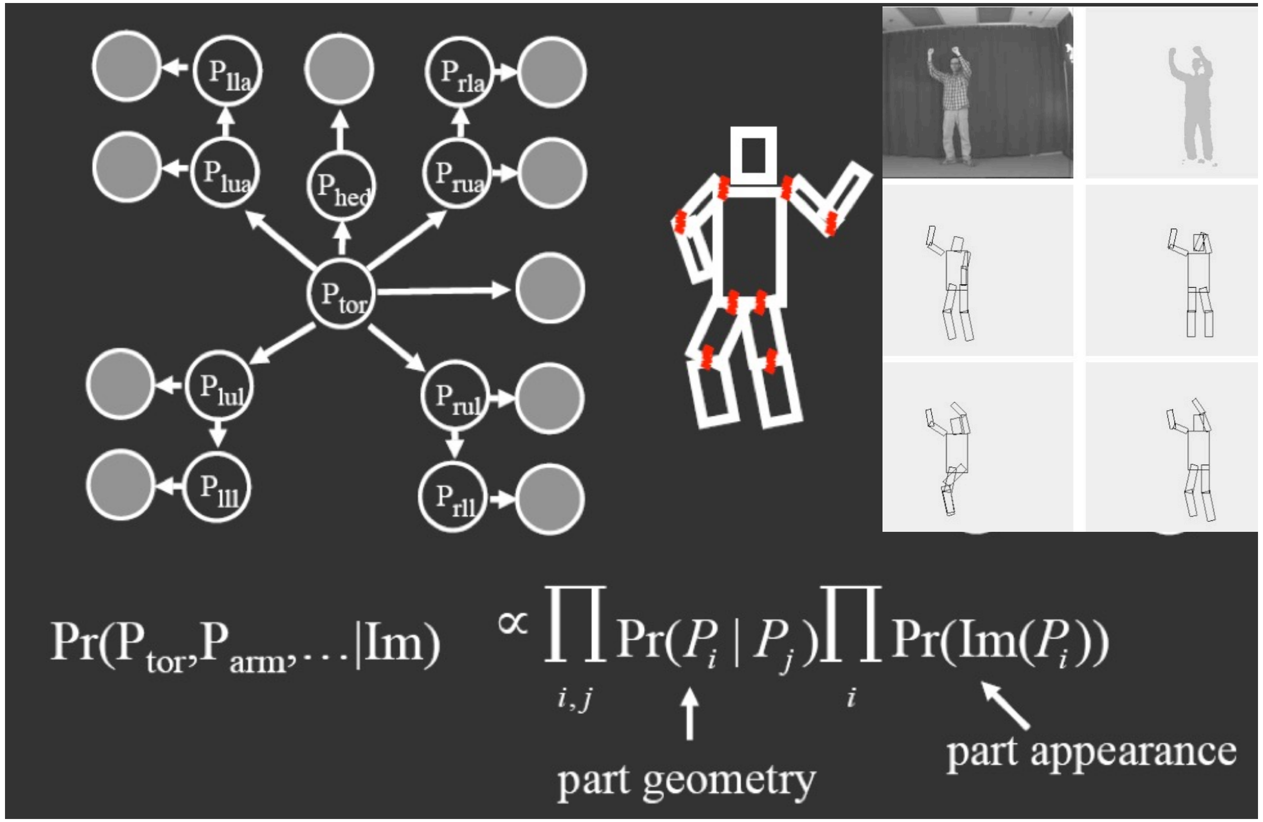
Rectangle features, boosting



Viola & Jones (2001)

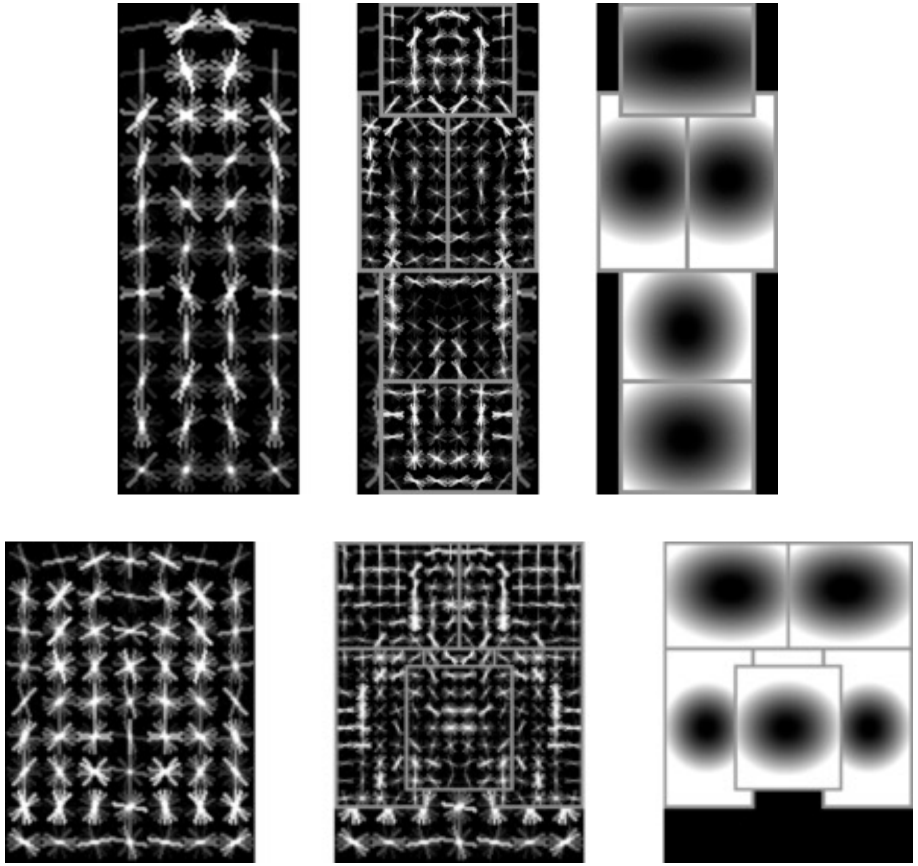
History of recognition: Deformable templates

Pictorial structures revisited



Felzenszwalb & Huttenlocher (2000)

Discriminatively trained deformable part-based models



Felzenszwalb et al. (2008)

What Matters in Recognition?

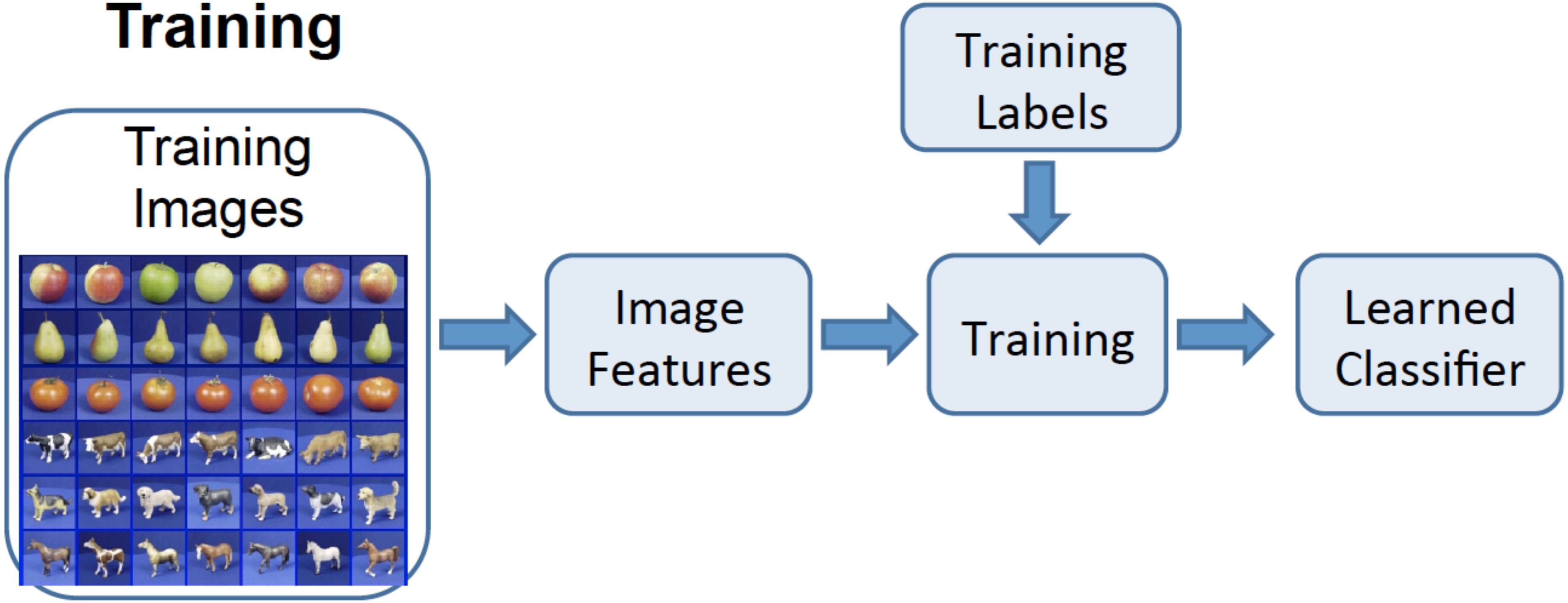
- Data
  - More is always better (as long as it is good data)
  - Annotation is the hard part

- Representation
  - Low level: SIFT, HoG, GIST, edges
  - Mid level: Bag of words, sliding window, deformable model
  - High level: Contextual dependence
  - Deep learned features

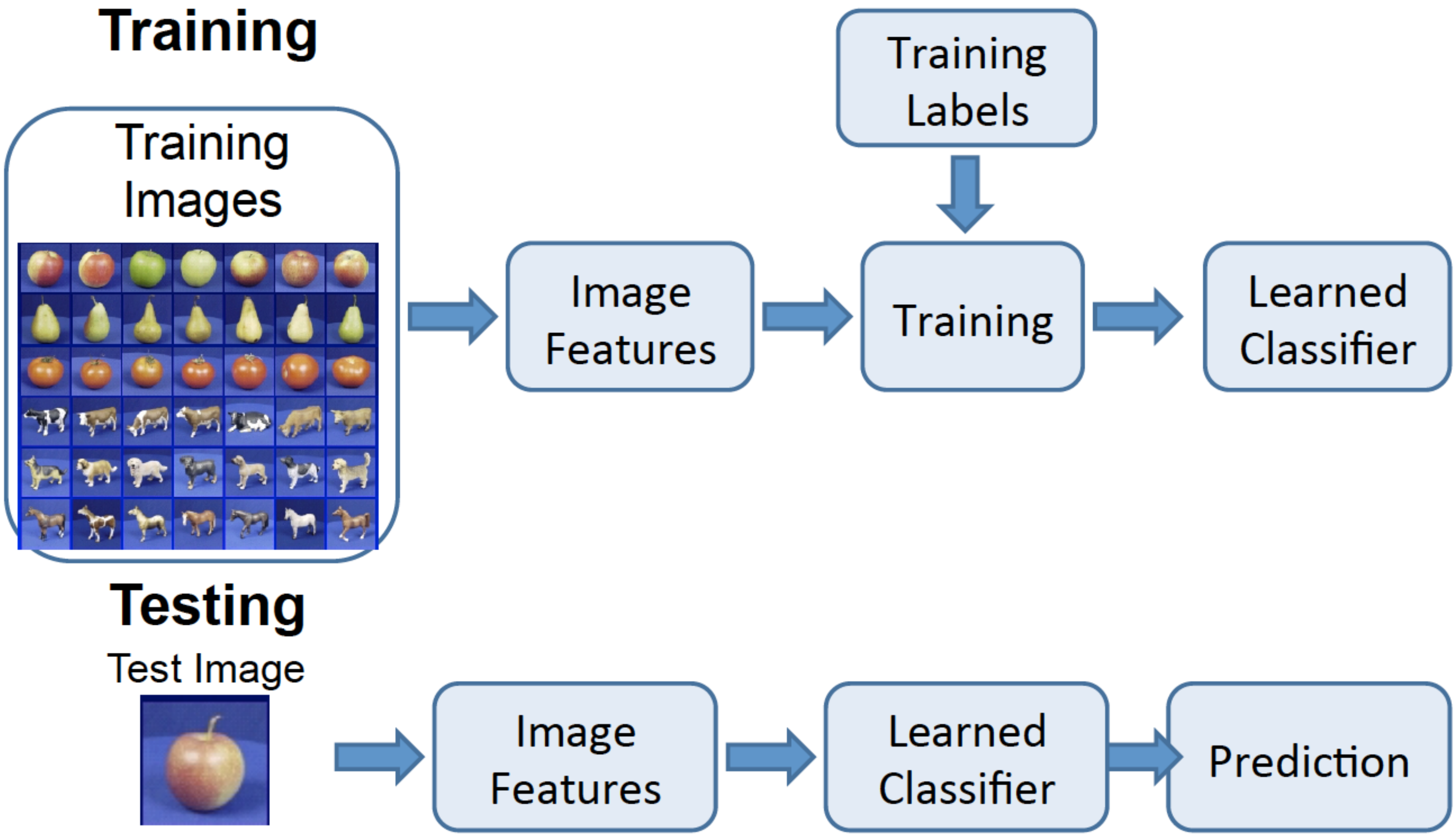
- Learning Techniques
  - E.g. choice of classifier or inference method

Training & Testing a Classifier

Training



Training & Testing a Classifier

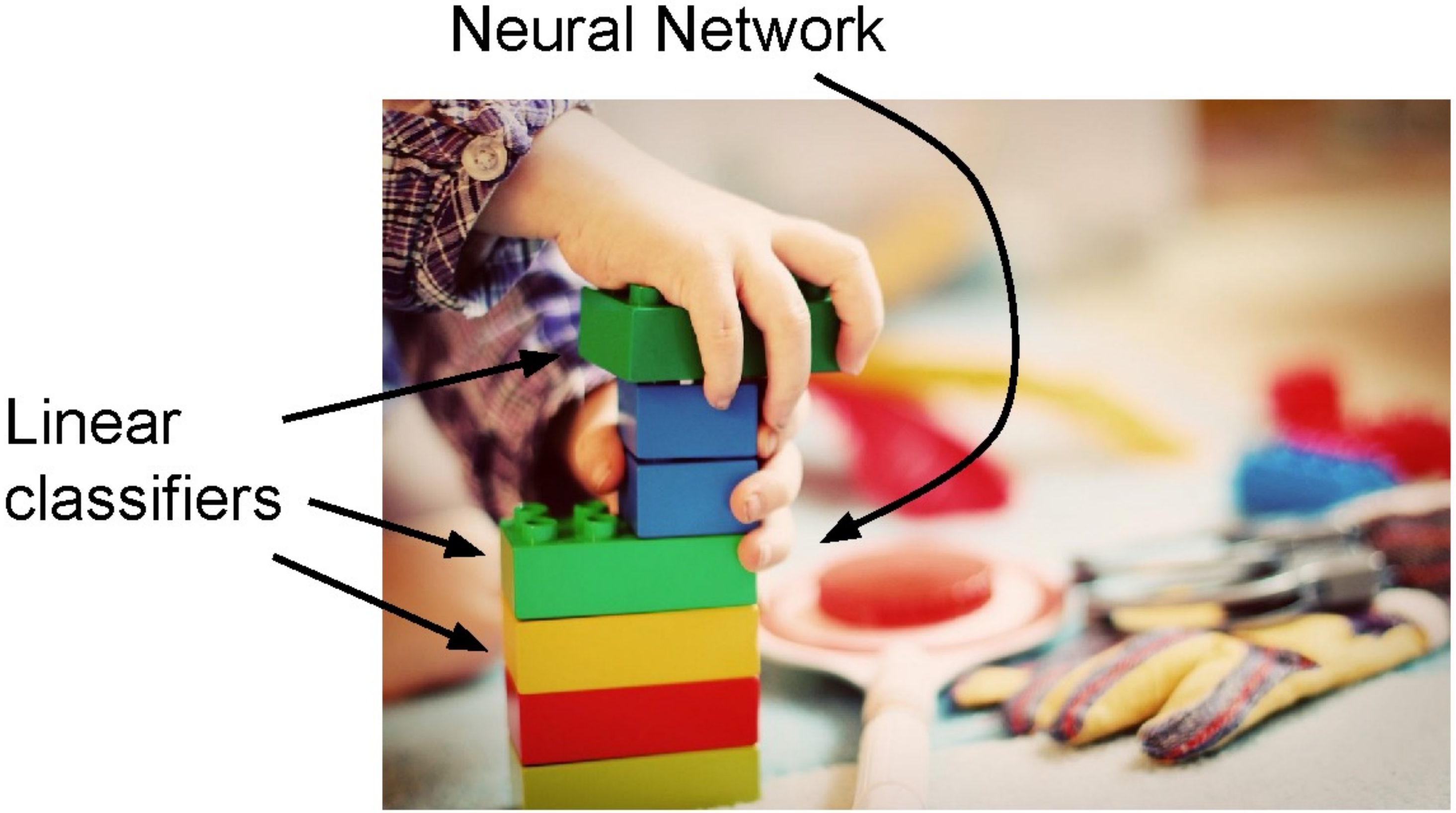


Dataset: ETH-80, by B. Leibe   Slide credit: D. Hoiem, L. Lazebnik

Classifiers

- Nearest Neighbor
- kNN (“k-Nearest Neighbors”)
- Linear Classifier
- Neural Network
- Deep Neural Network
- …

Linear Classifiers



Score functions

class scores

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Parametric Approach

image parameters

$f(\mathbf{x},\mathbf{W})$

10 numbers, indicating class scores

[32x32x3] $= 3072$
array of numbers 0...1
(3072 numbers total)

Parametric Approach: Linear Classifier

$$\boxed{f(x, W)} = \boxed{W}\boxed{x}$$

3072x1

10x1 **10x3072**

[32x32x3] $= 3072$
array of numbers 0...1

10 numbers,
indicating class
scores

parameters, or "weights"

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Parametric Approach: Linear Classifier

$$\underbrace{\boxed{f(x,W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\,\overbrace{\boxed{x}}^{3072\text{x}1} \quad \boxed{(+b)}\ 10\text{x}1$$

[32x32x3] $= 3072$
array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”

Linear Classifier

define a **score function**

data (image)

$$f(x_i, W, b) = Wx_i + b$$

class scores

"weights"

"bias vector"

"parameters"

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Interpretation: Algebraic

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

stretch pixels into single column


input image

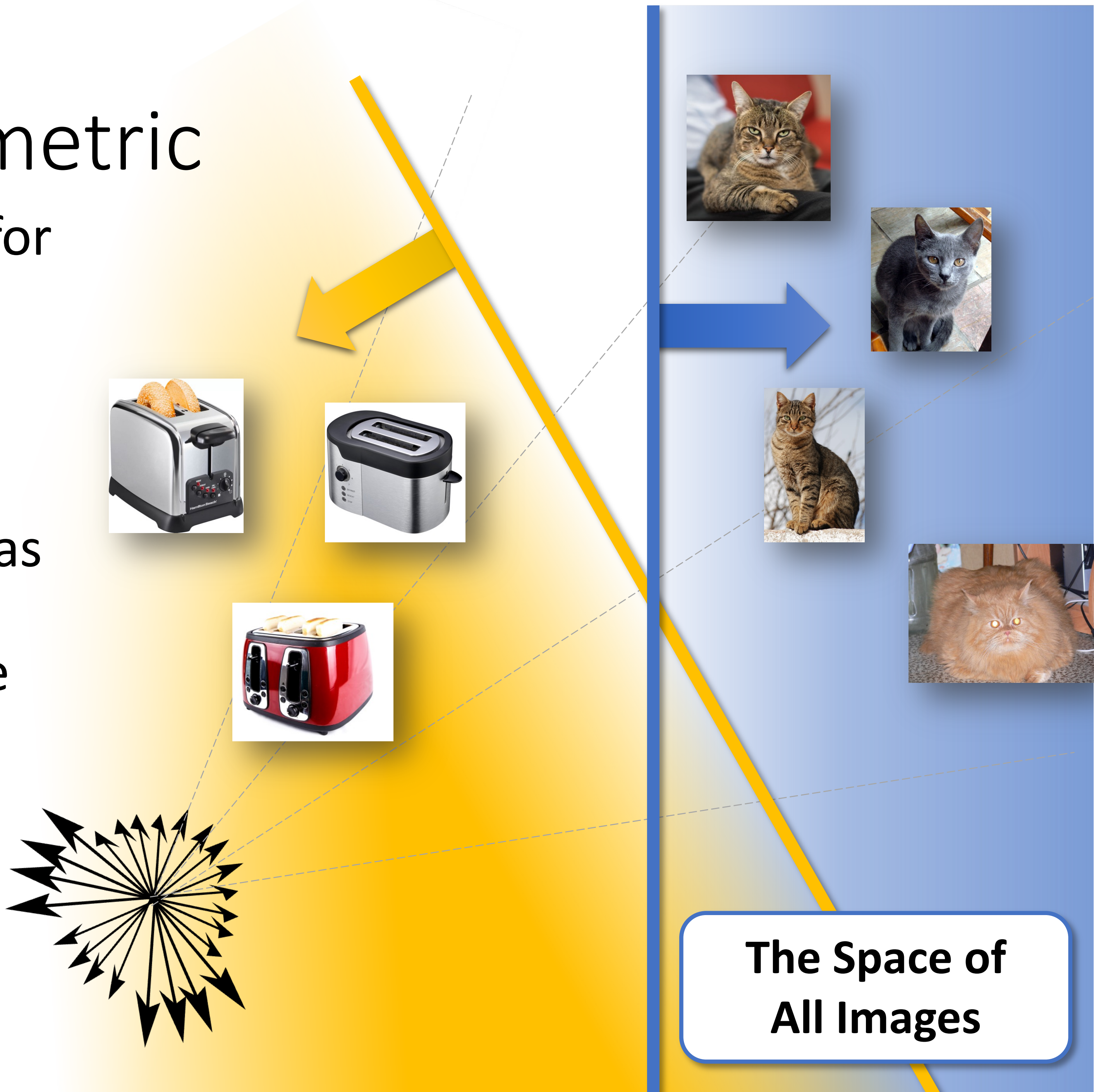
$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} \longrightarrow \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{f(x_i; W, b)} \begin{array}{l} \text{cat score} \\ \text{dog score} \\ \text{ship score} \end{array}$$

Interpretation: Geometric

- Parameters define a hyperplane for each class:

$$f(x_i, W, b) = Wx_i + b$$

- We can think of each class score as defining a distribution that is proportional to distance from the corresponding hyperplane



Simpler example: binary classification

- Two classes (e.g., “cat” and “not cat”)
  - AKA “positive” and “negative” classes

cat

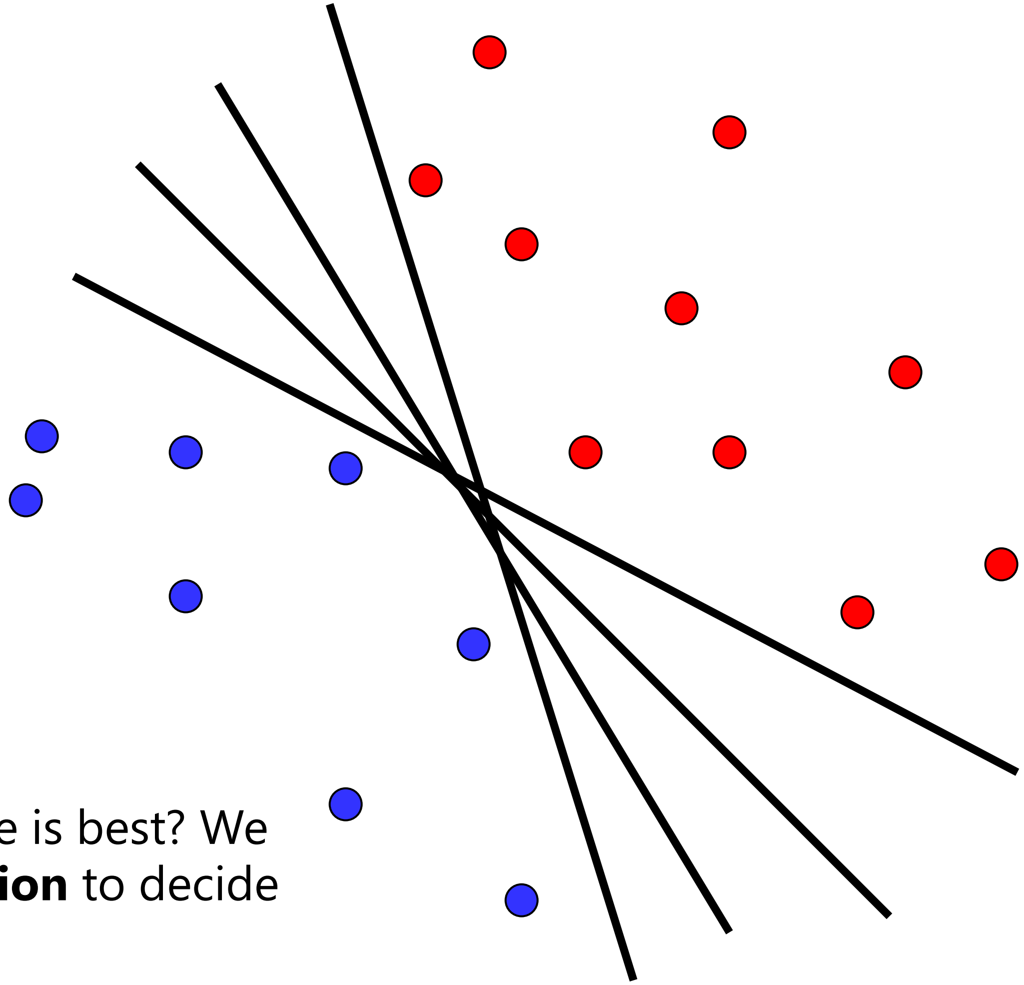
not cat

Simpler example: binary classification

- Find linear function (*hyperplane*) to separate positive and negative examples

$$\mathbf{x}_i \text{ positive}: \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$$

$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$



Which hyperplane is best? We need a **loss function** to decide

Linear classification

airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Output scores

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

Loss functions

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Softmax classifier

- Interpret Scores as unnormalized log probabilities of classes

$f(x_i, W) = Wx_i$ (score function)

$\boxed{\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}}$ softmax function

Squashes values into *probabilities* ranging from 0 to 1

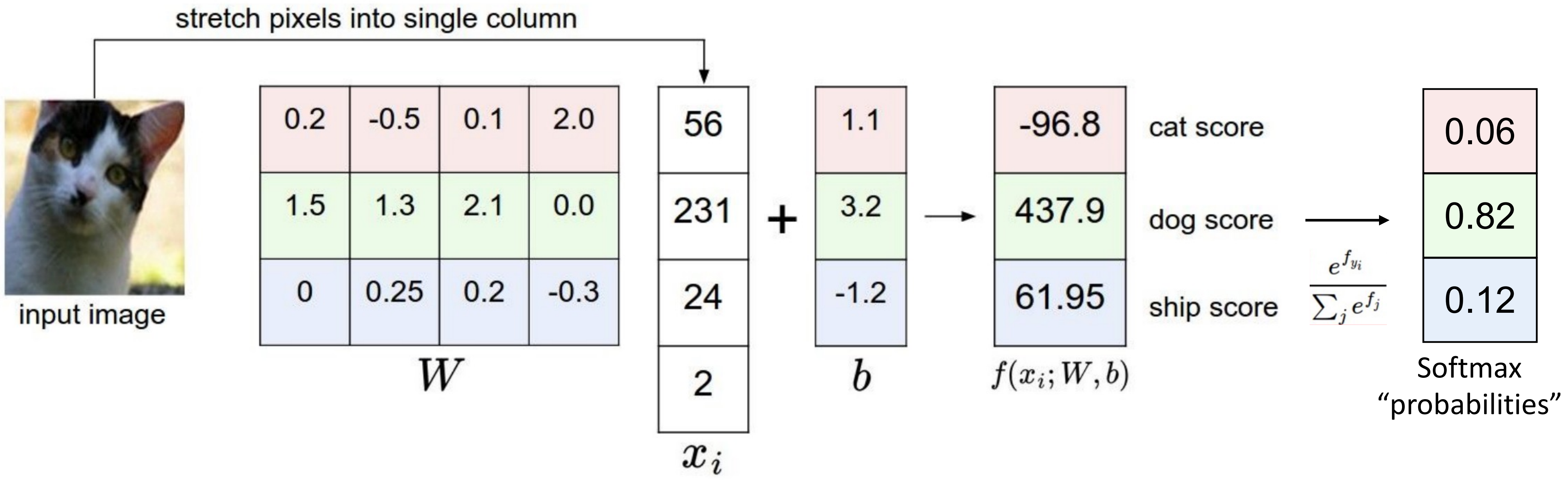
$\boxed{P(y_i \mid x_i; W)}$

Example with three classes:

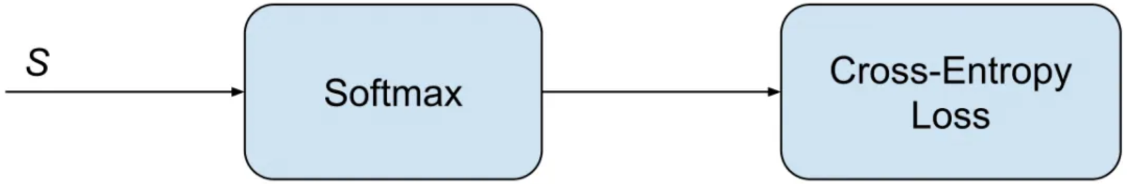
$$[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]$$

Softmax classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Cross-entropy loss



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

t_i = label for class, usually 1 for right class, else 0.

$$f(x_i, W) = W x_i$$ (score function)

Summary

- Have score function and loss function
  - Currently, score function is based on linear classifier
  - Next, will generalize to convolutional neural networks
- Find W and b to minimize loss

$$L = \underbrace{\frac{1}{N}\sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)}_{\text{Average of cross-entropy loss over all training examples}} + \lambda \sum_k \sum_l W_{k,l}^2$$

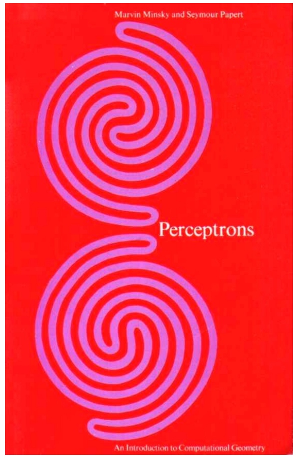
Regularization term

(Deep) Neural Networks

History of recognition: Neural networks

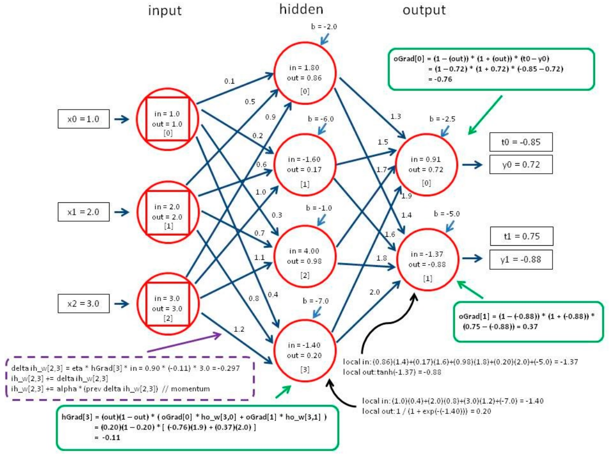
Perceptrons

Rosenblatt (1958)



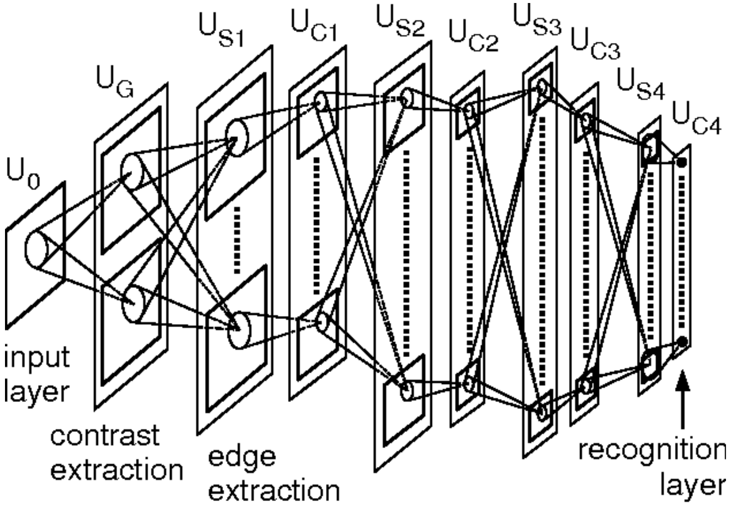
Minsky & Papert (1969)

Back-propagation



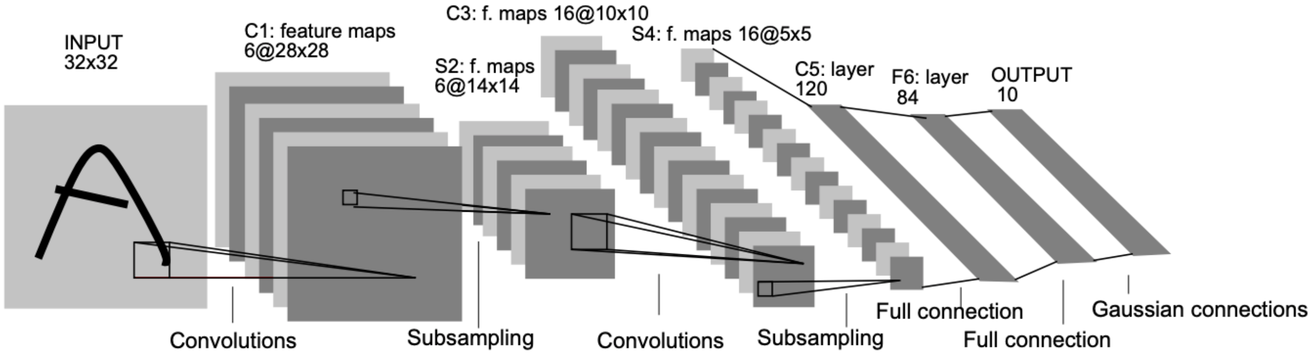
Rumelhart, Hinton & Williams (1986)

Neocognitron



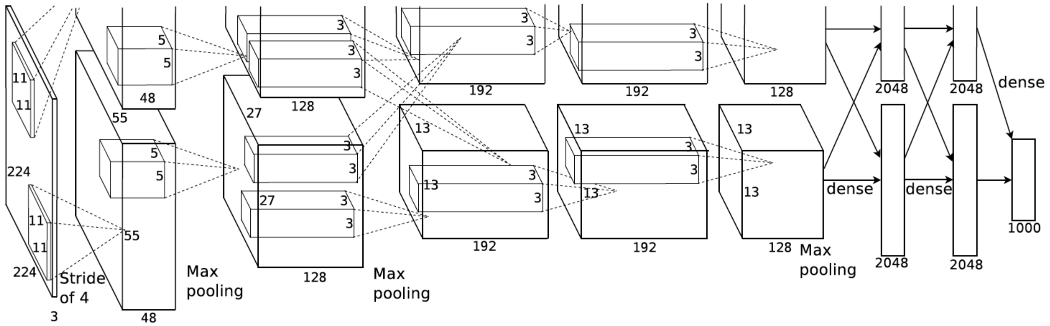
Fukushima (1980)

LeNet-5



LeCun et al. (1998)

AlexNet



Krizhevsky et al. (2012)

Neural networks

(**Before**) Linear score function: $f = Wx$

Neural networks

(**Before**) Linear score function: $f = Wx$

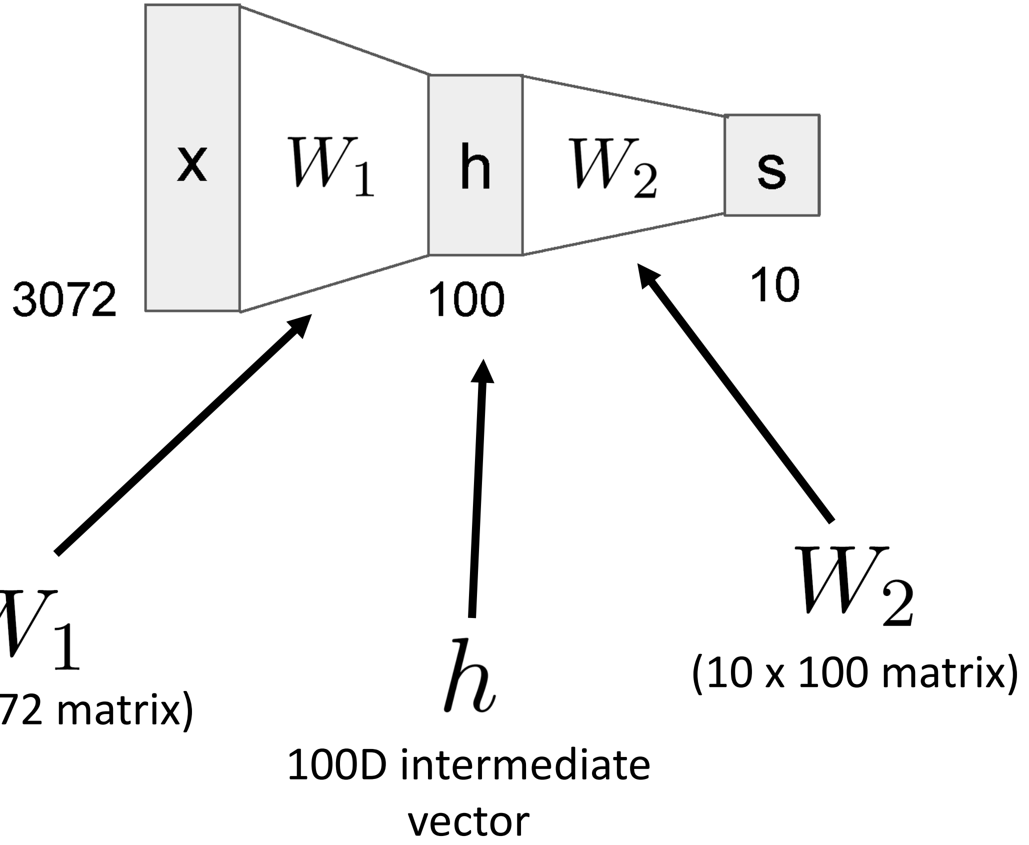
(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

Non-linear Activation Function
(many other choices exist)

Neural networks

(**Before**) Linear score function: $f = Wx$

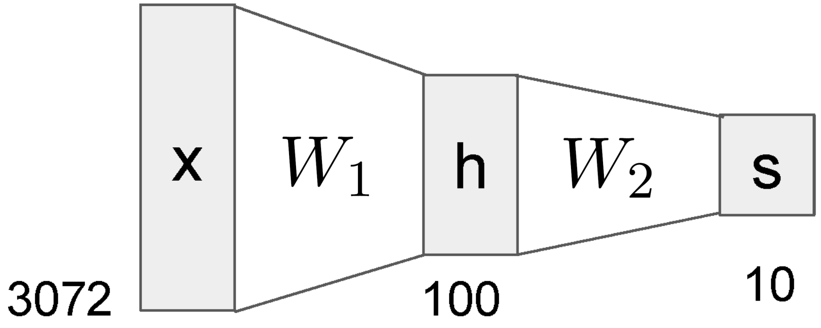
(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



Neural networks

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



- Total number of weights to learn:

  3,072 x 100 + 100 x 10 = 308,200

Neural networks

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

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

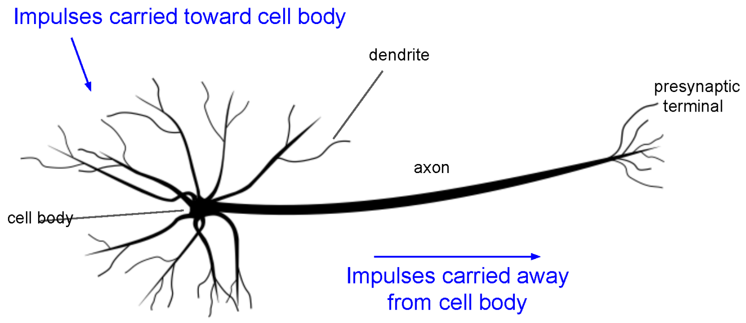
also called "Multi-Layer Perceptrons" (MLPs)

Neural networks

- Very coarse generalization of neural networks:
    - Linear functions chained together and separated by non-linearities (*activation functions*), e.g. “max”

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

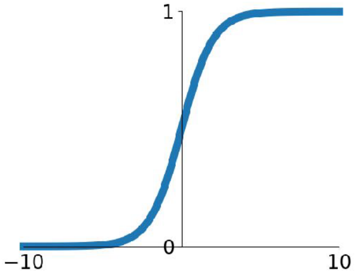
    - Why separate linear functions with non-linear functions?
    - *Very roughly* inspired by real neurons



Activation functions

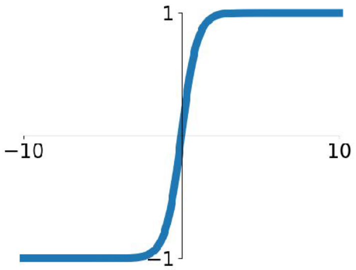
Sigmoid

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



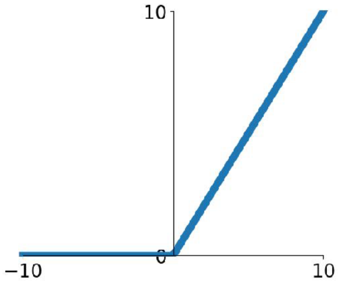
tanh

$\tanh(x)$



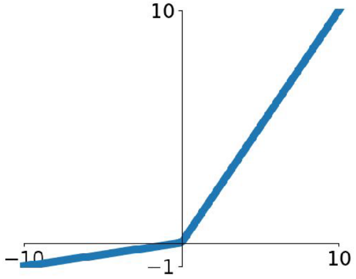
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

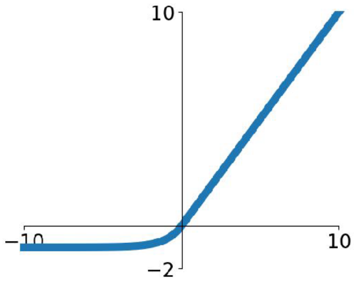


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

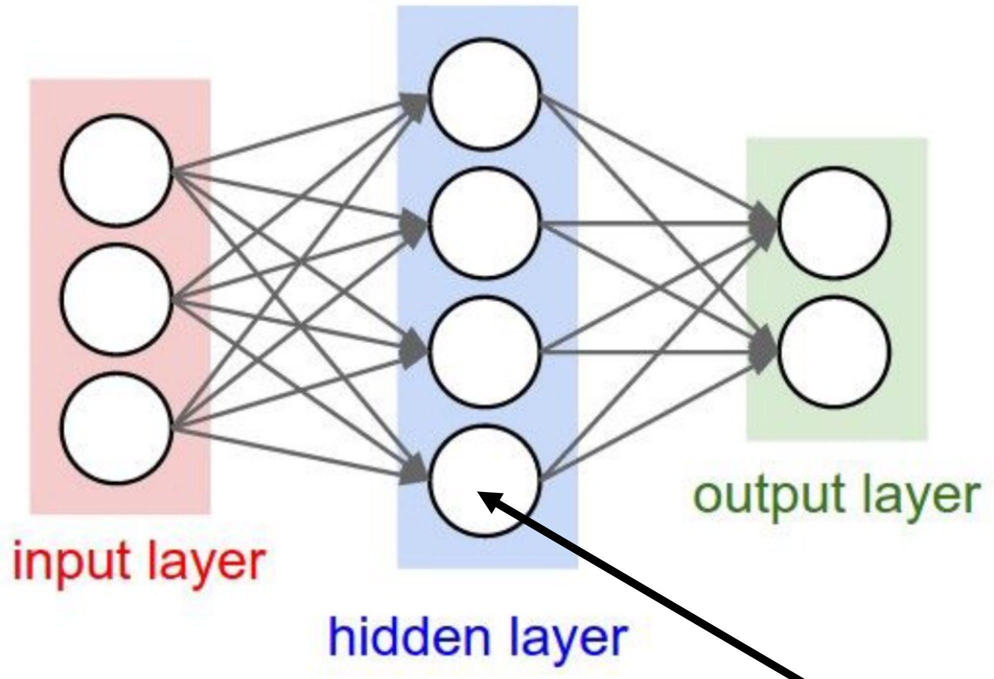
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



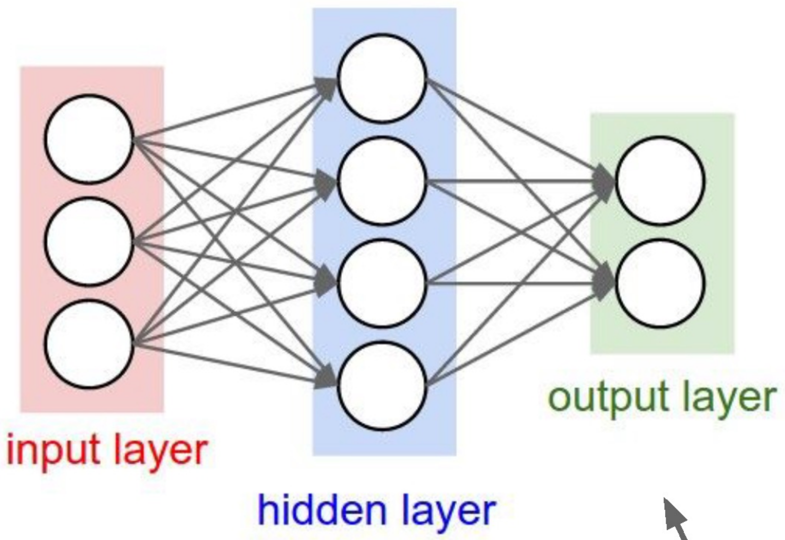
Neural network architecture

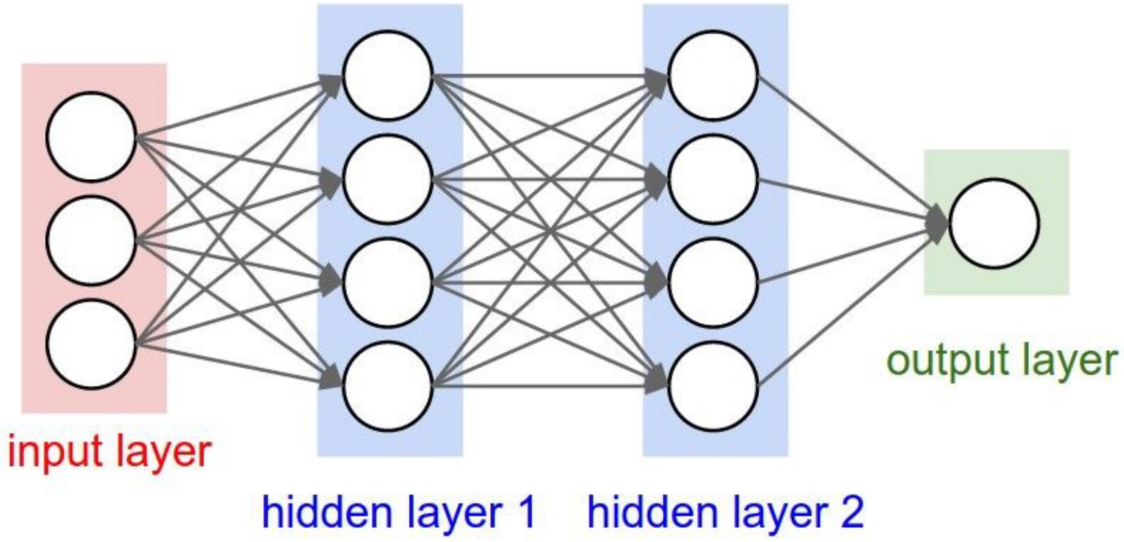
- Computation graph for a 2-layer neural network



Neuron or *unit*

Neural networks: Architectures





“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

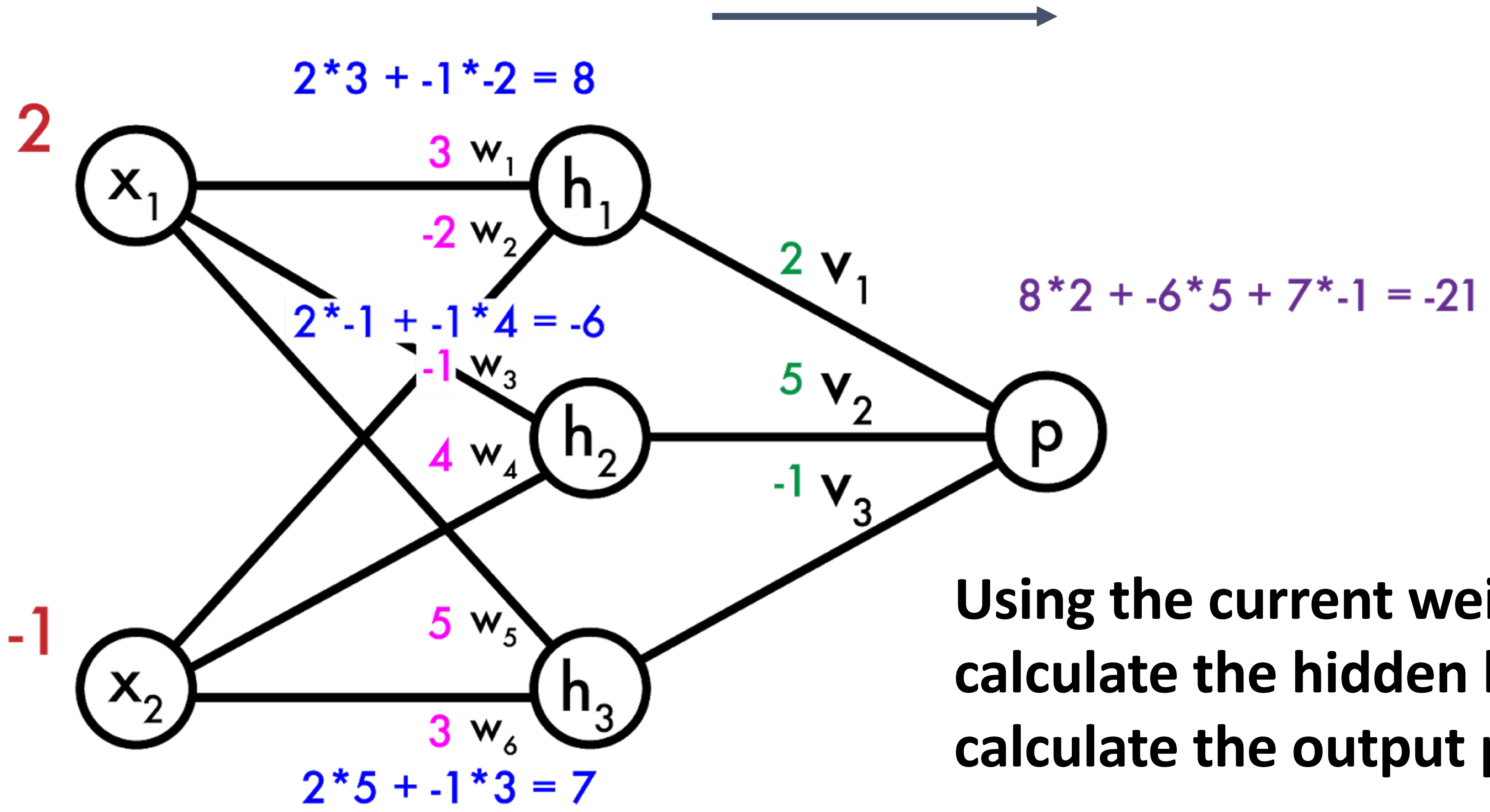
“Fully-connected” layers

Deep networks typically have many layers and potentially millions of parameters

Optimizing parameters with gradient descent

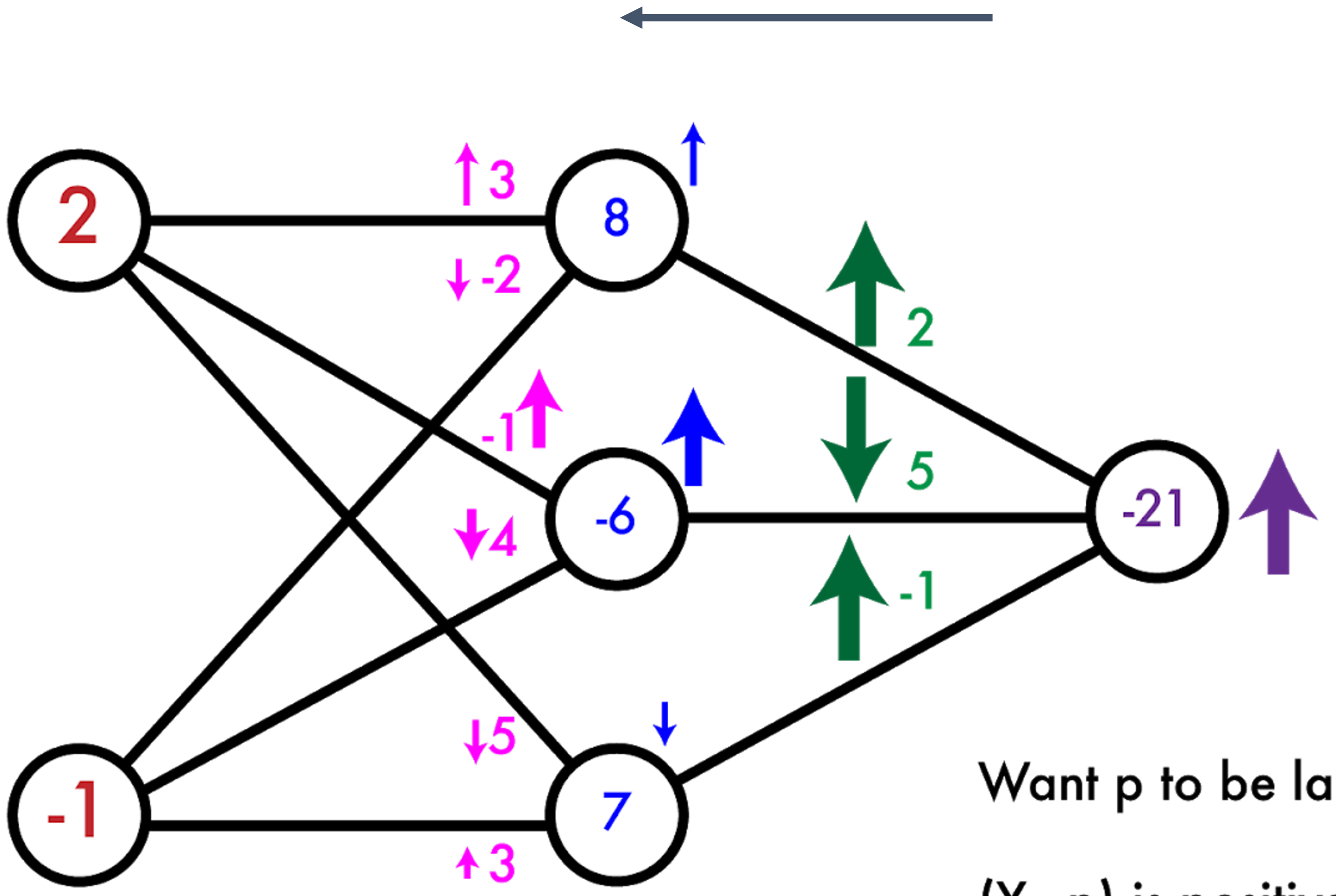
- How do we find the best **W** and **b** parameters?
- In general: *gradient descent*
    1. Start with a guess of a good **W** and **b** (or randomly initialize them)
    2. Compute the loss function for this initial guess and the *gradient* of the loss function
    3. Step some distance in the negative gradient direction (direction of steepest descent)
    4. Repeat steps 2 & 3
- Note: efficiently performing step 2 for deep networks is called *backpropagation*

The Learning Cycle: Forward Propagation



Using the current weights of the model calculate the hidden layer neurons, then calculate the output p.

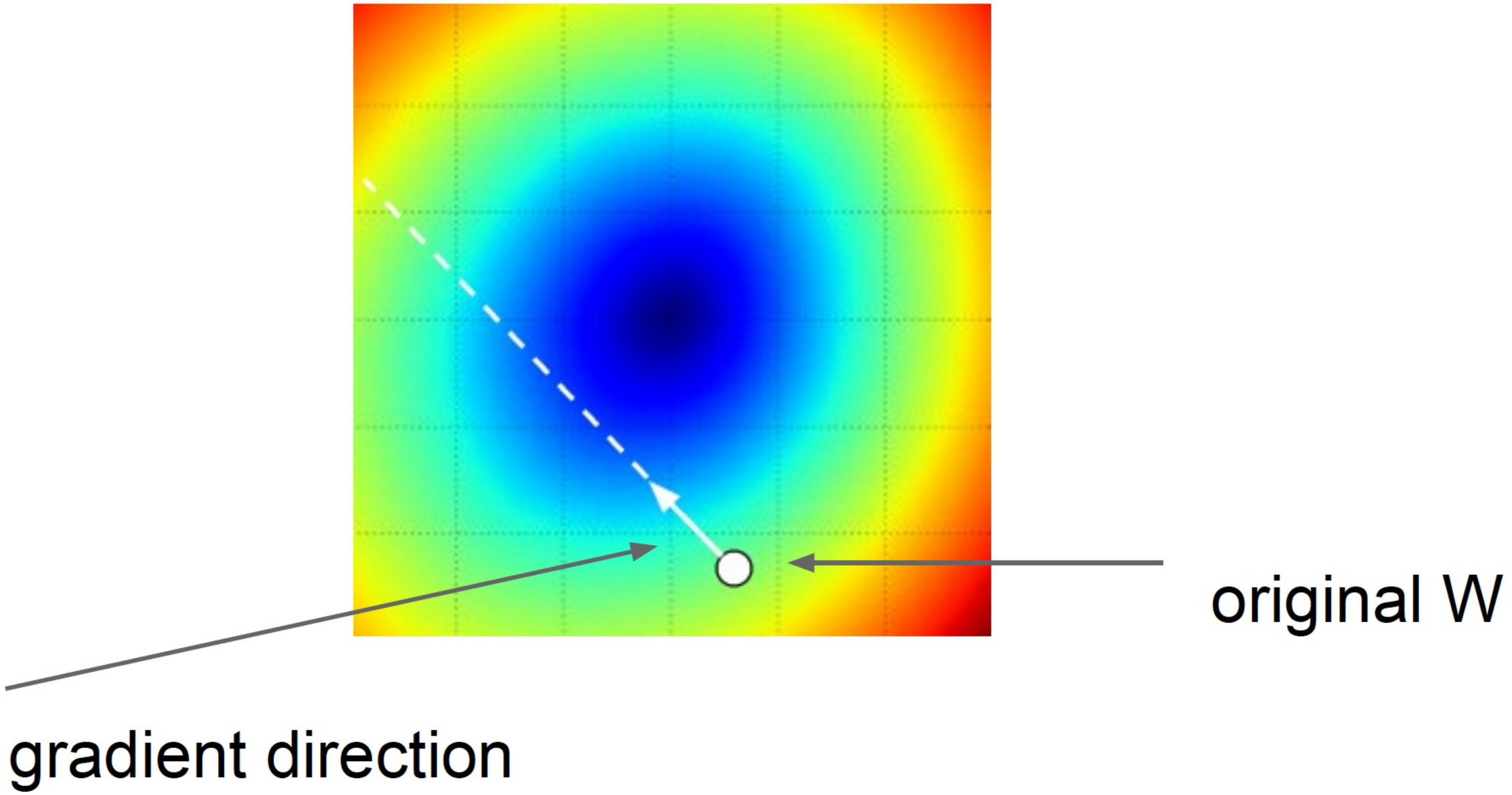
The Learning Cycle: Backward Propagation



Want p to be larger...

(Y - p) is positive

How do we change our weights?



Gradient descent: walk in the direction opposite gradient

- **Q**: How far?
- **A**: Step size: *learning rate*
- Too big: will miss the minimum
- Too small: slow convergence

What Matters in Recognition?

- Data
  - More is always better (as long as it is good data)
  - Annotation is the hard part
- Representation
  - Low level: SIFT, HoG, GIST, edges
  - Mid level: Bag of words, sliding window, deformable model
  - High level: Contextual dependence
  - Deep learned features
- Learning Techniques
  - E.g. choice of classifier or inference method

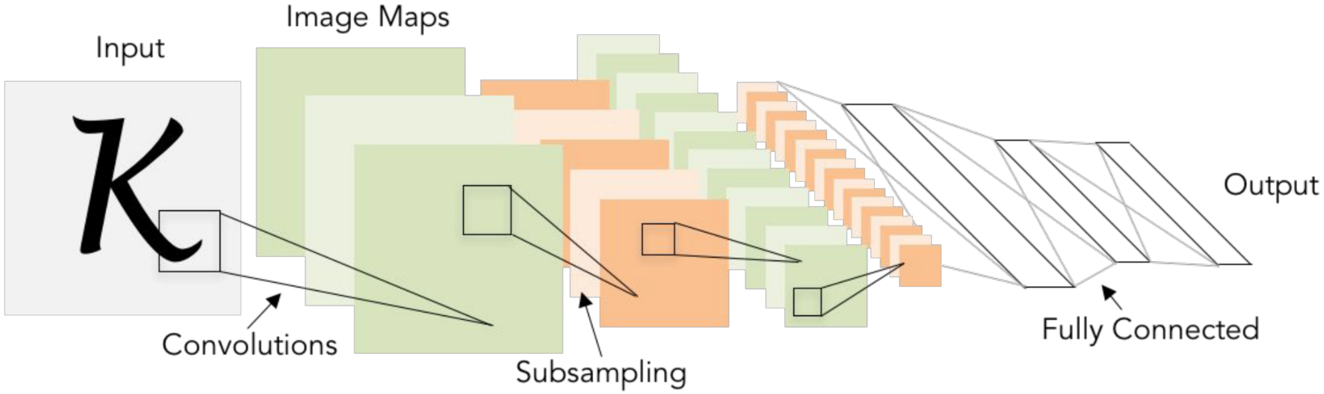
Combine!

Convolutional Neural Networks (CNNs)

Convolutional neural networks

A bit of history:

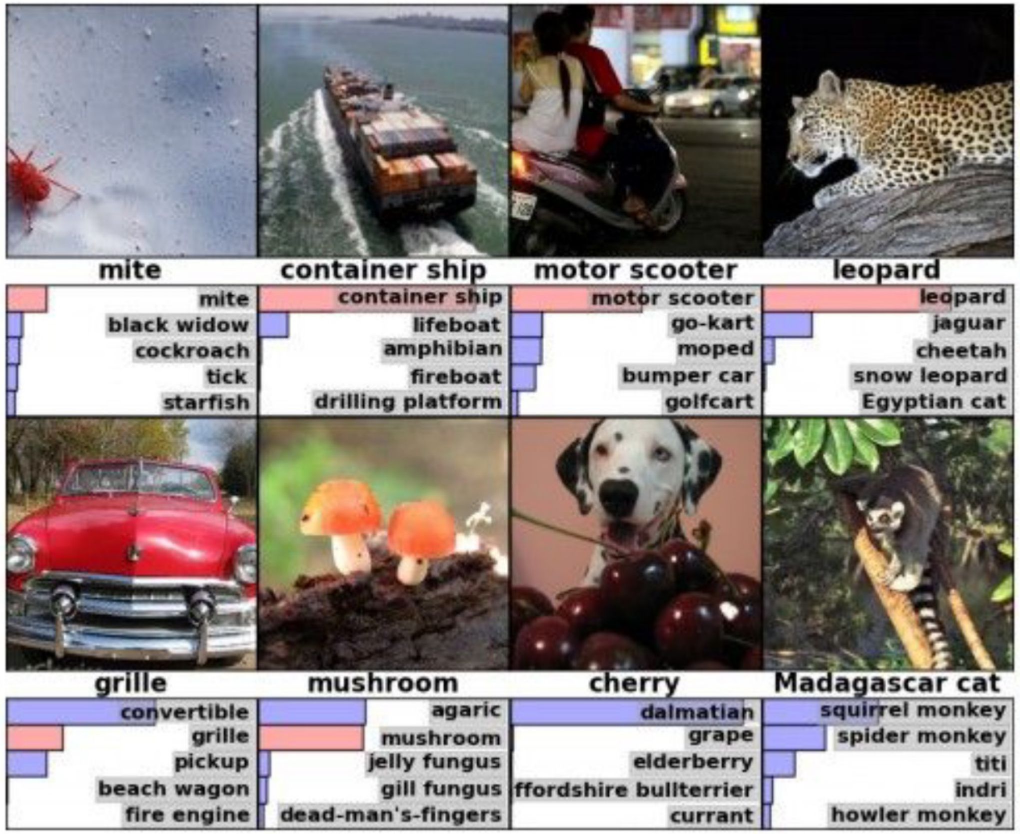
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

Fast-forward to today: ConvNets are everywhere

Classification



Retrieval

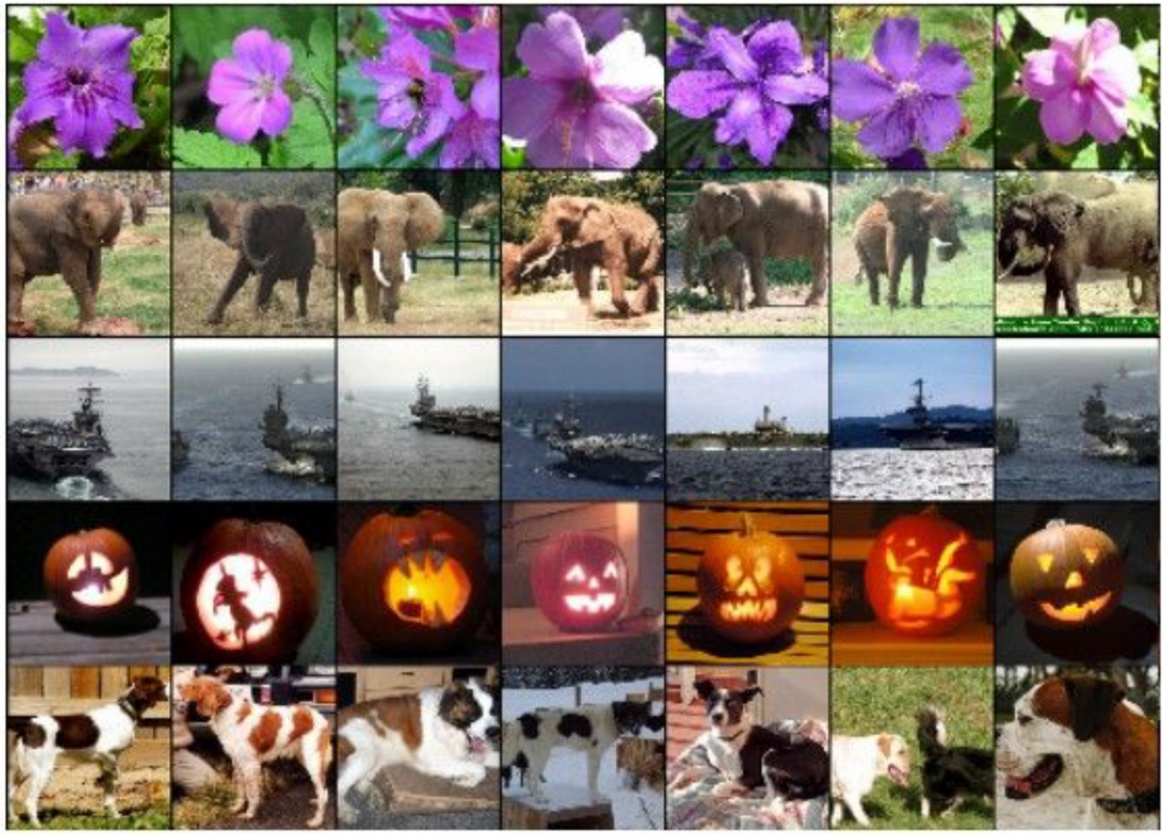


Image features vs ConvNets

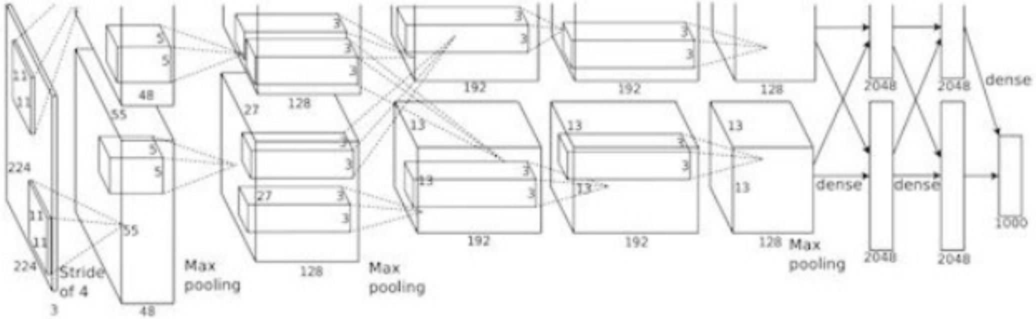
Feature Extraction

f

f = linear classifier

10 numbers giving scores for classes

training



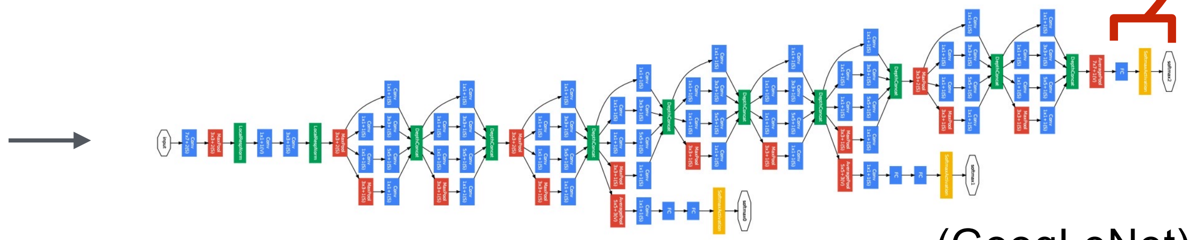
Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012.

10 numbers giving scores for classes

training

Last layer of most CNNs is a linear classifier

This piece is just a linear classifier



(GoogLeNet)

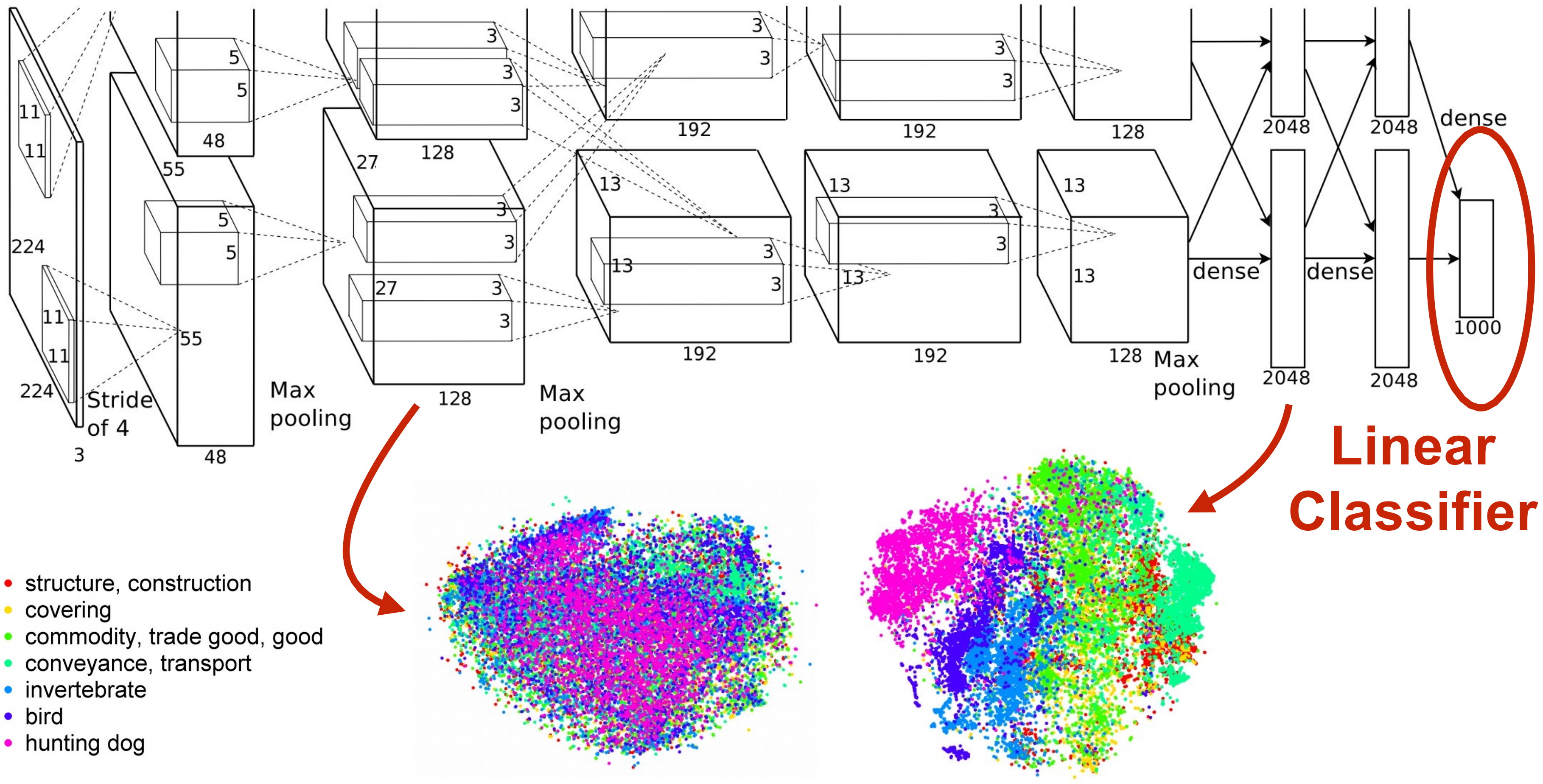
→ Ans

Input Pixels

Perform everything with a big neural network, trained end-to-end

Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable

Visualizing AlexNet in 2D with t-SNE



(c) DeCAF_1 (d) DeCAF_6

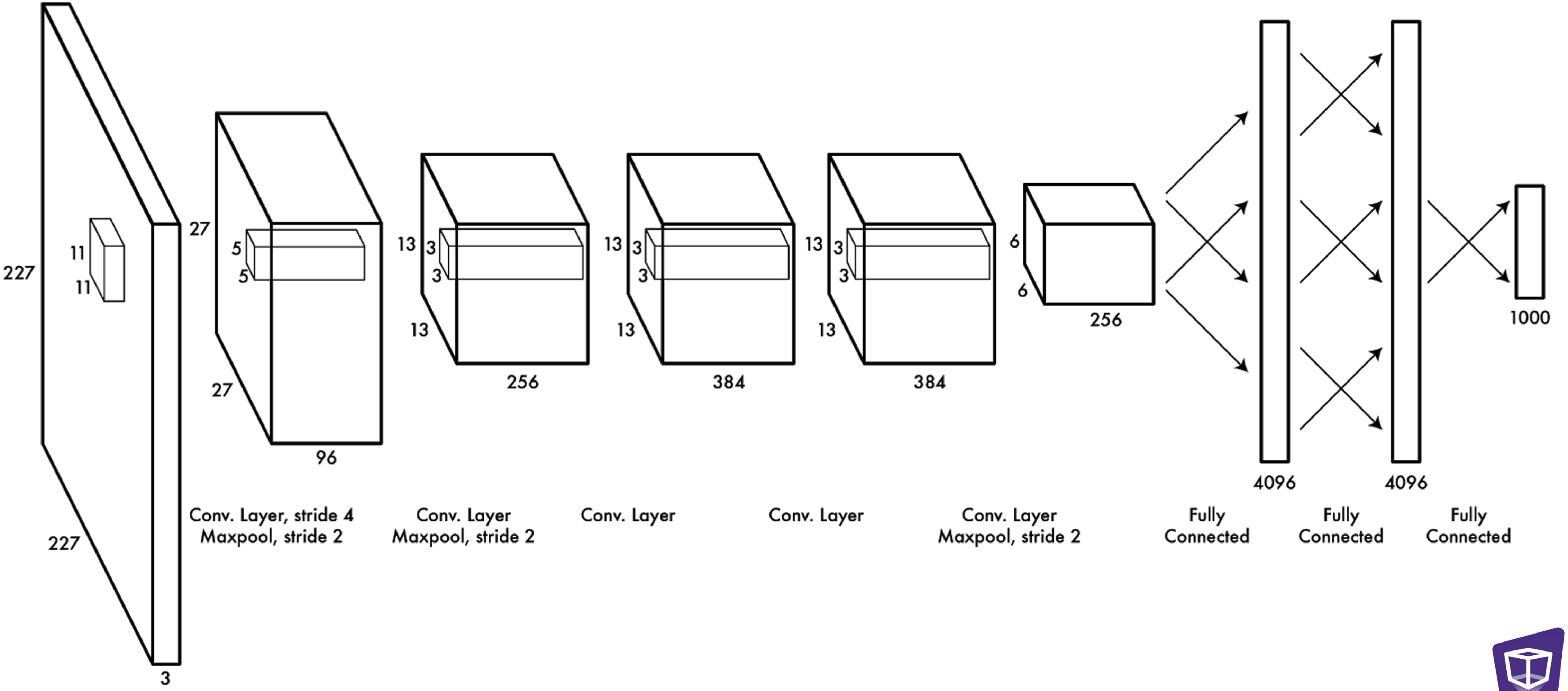
(2D visualization using t-SNE)

[Donahue, "DeCAF: DeCAF: A Deep Convolutional …", arXiv 2013]

Convolutional neural networks

- Layer types:
  - *Convolutional layer*
  - Pooling layer
  - Fully-connected layer

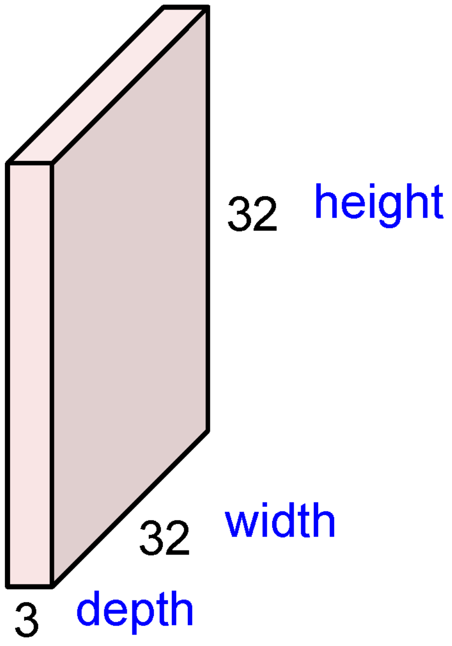
AlexNet: An Early Example



https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

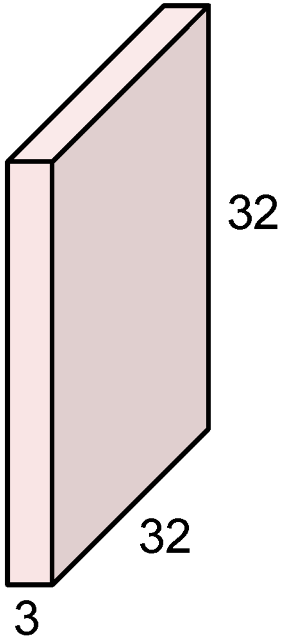
Convolution Layer

32x32x3 image -> preserve spatial structure

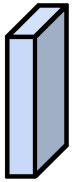


Convolution Layer

32x32x3 image



5x5x3 filter



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

Convolution (Recap)

- Same as cross-correlation, except that the kernel is "flipped" (horizontally and vertically)

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

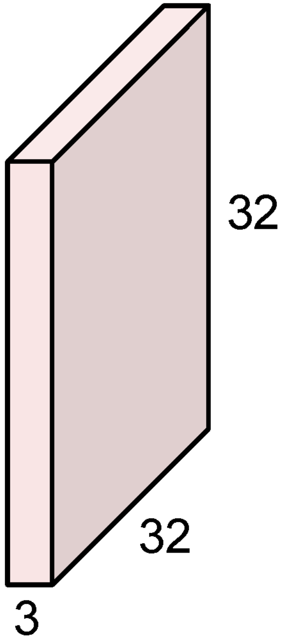
Cross-correlation

- Convolution is **commutative** and **associative**

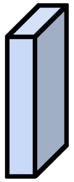
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



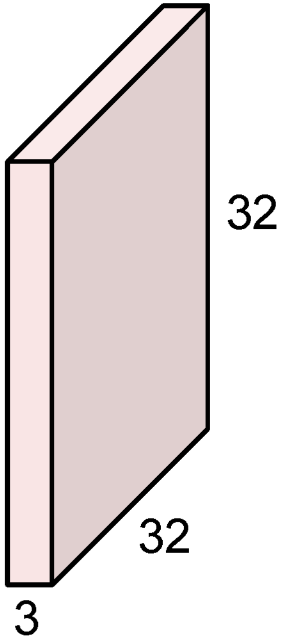
5x5x3 filter



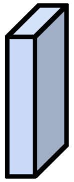
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image



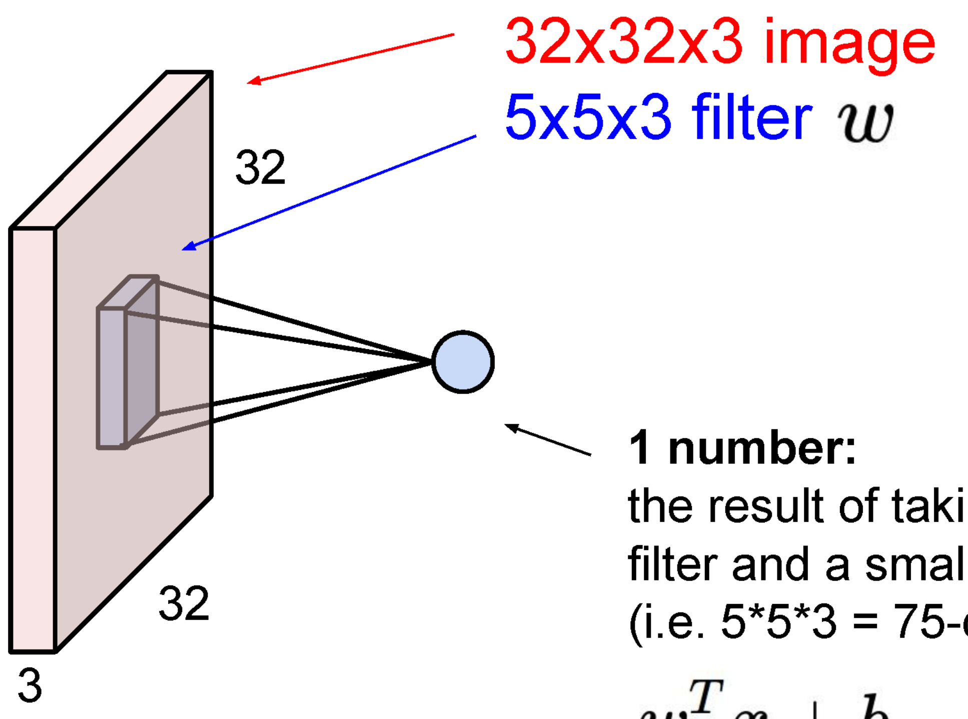
5x5x3 filter



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

Number of weights: 5 x 5 x 3 + 1 = **76**
(vs. 3072 for a fully-connected layer)
(+1 for bias)

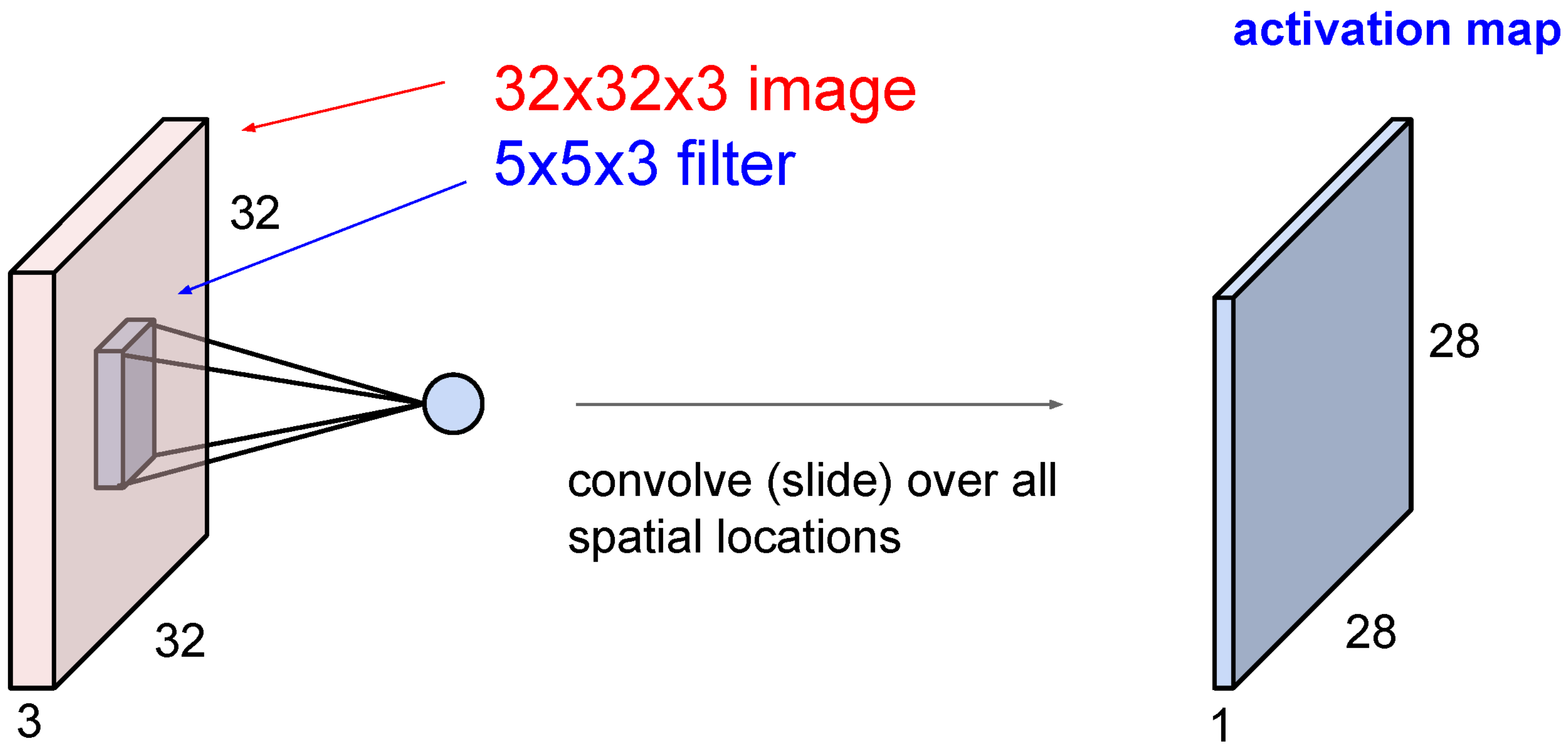
Convolution Layer



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

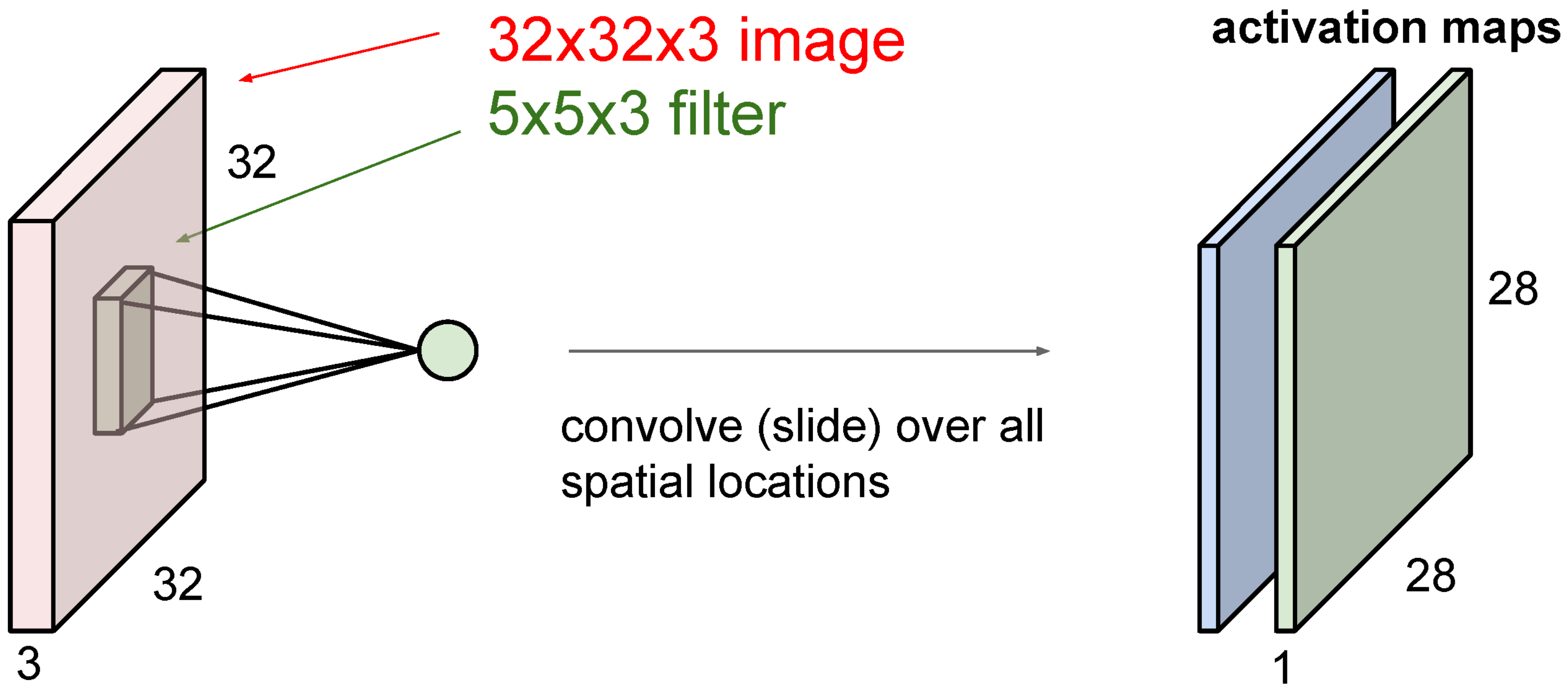
$$w^T x + b$$

Convolution Layer

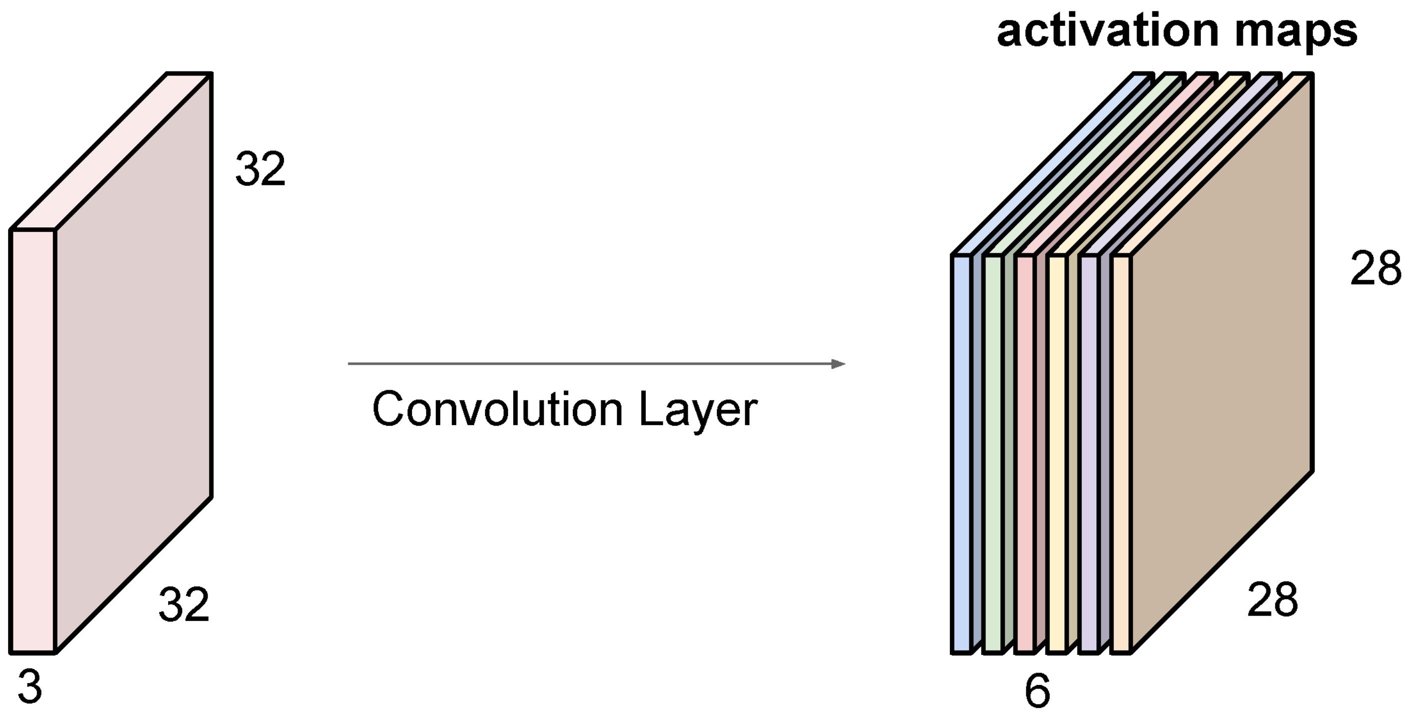


Convolution Layer

consider a second, green filter



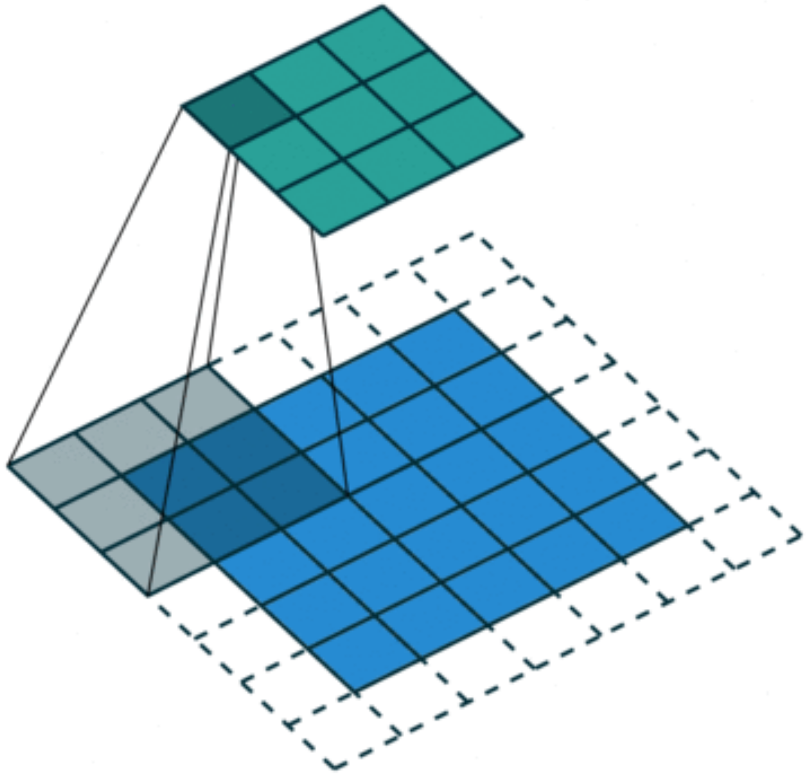
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

(total number of parameters: 6 x (75 + 1) = **456**)

Padding & Stride in CNN



padding=1, stride=2

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

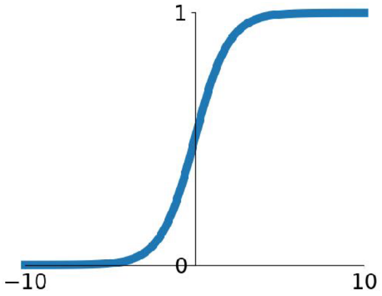
$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Activation functions

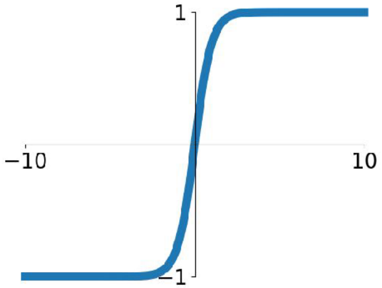
Sigmoid

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



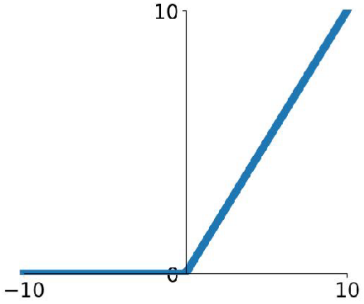
tanh

$$\tanh(x)$$



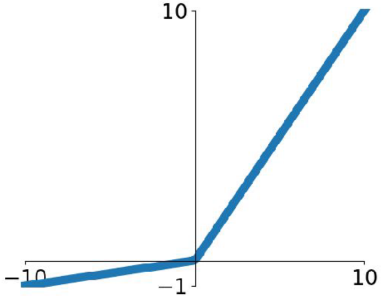
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

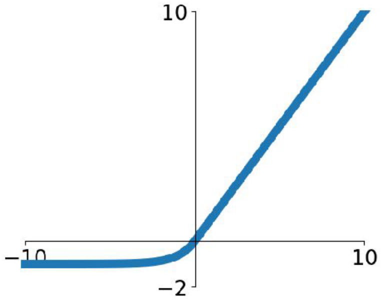


Maxout

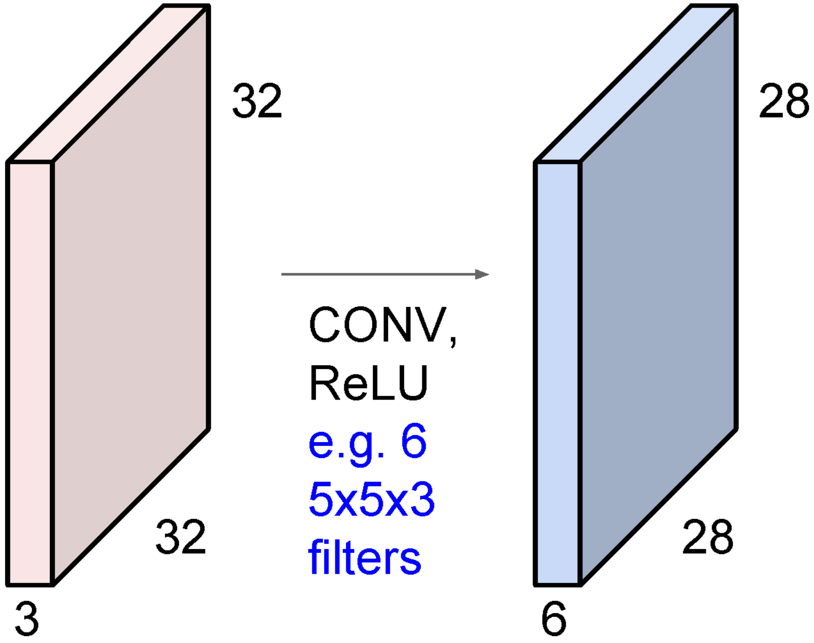
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

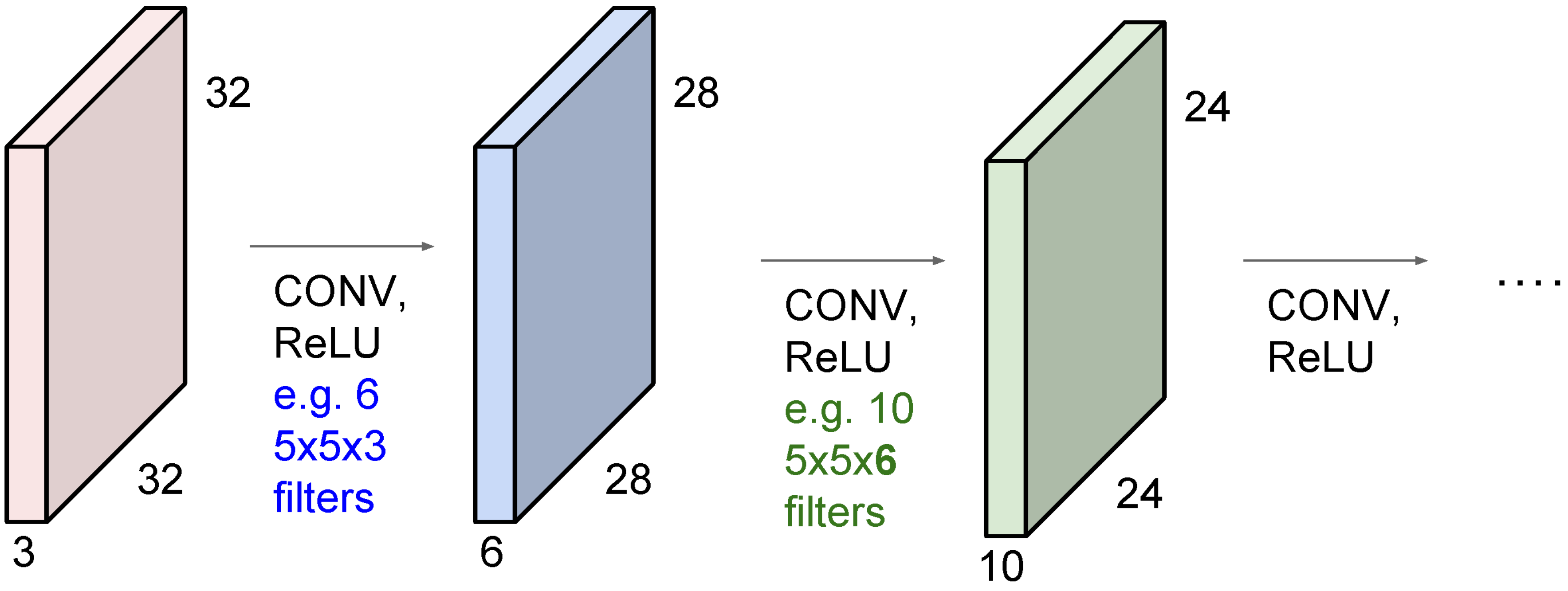
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



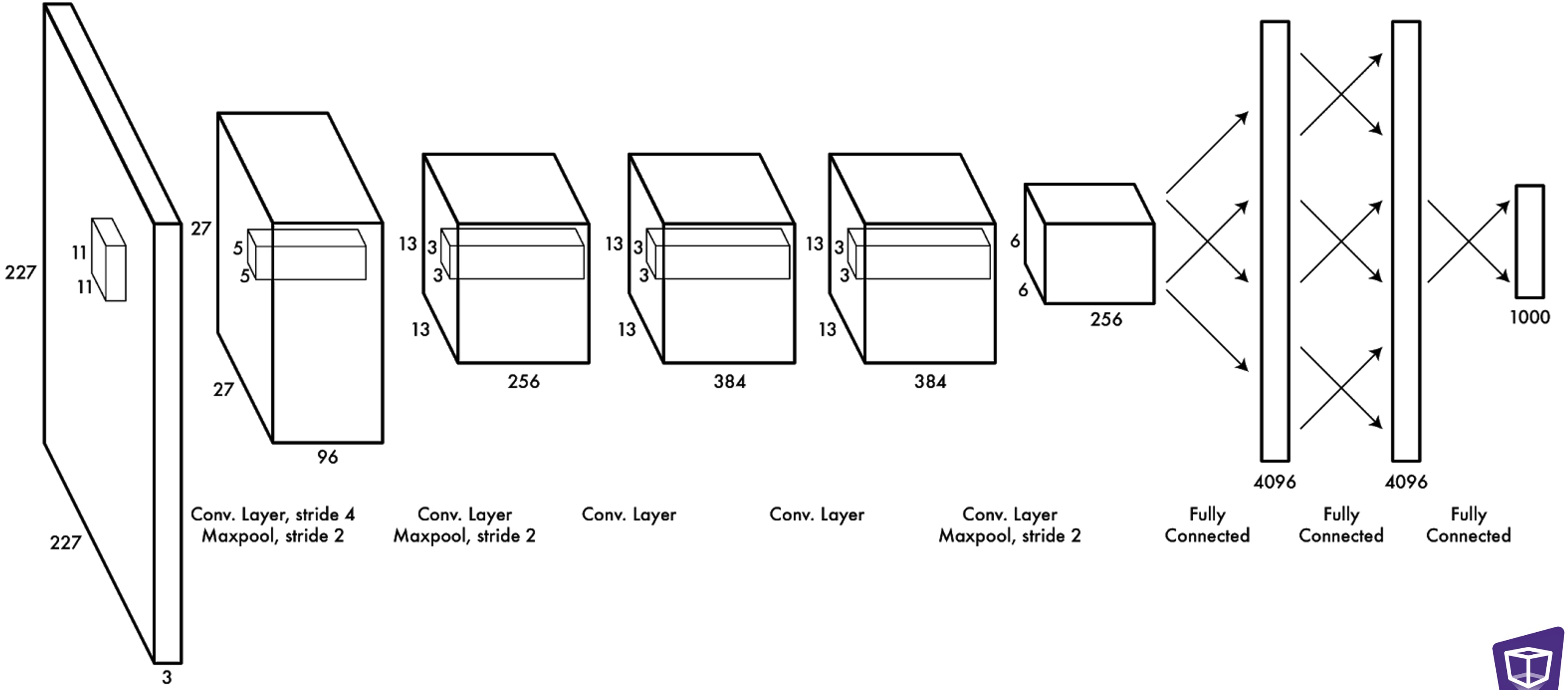
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



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



AlexNet: An Early Example

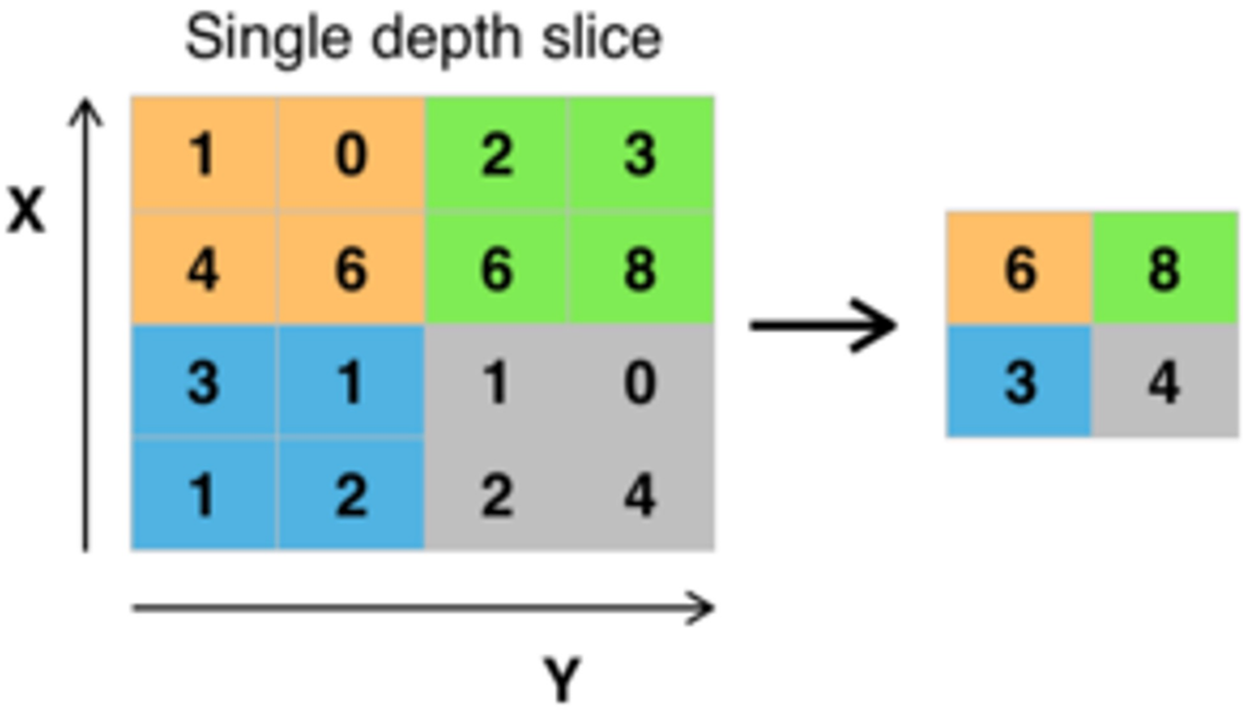


https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

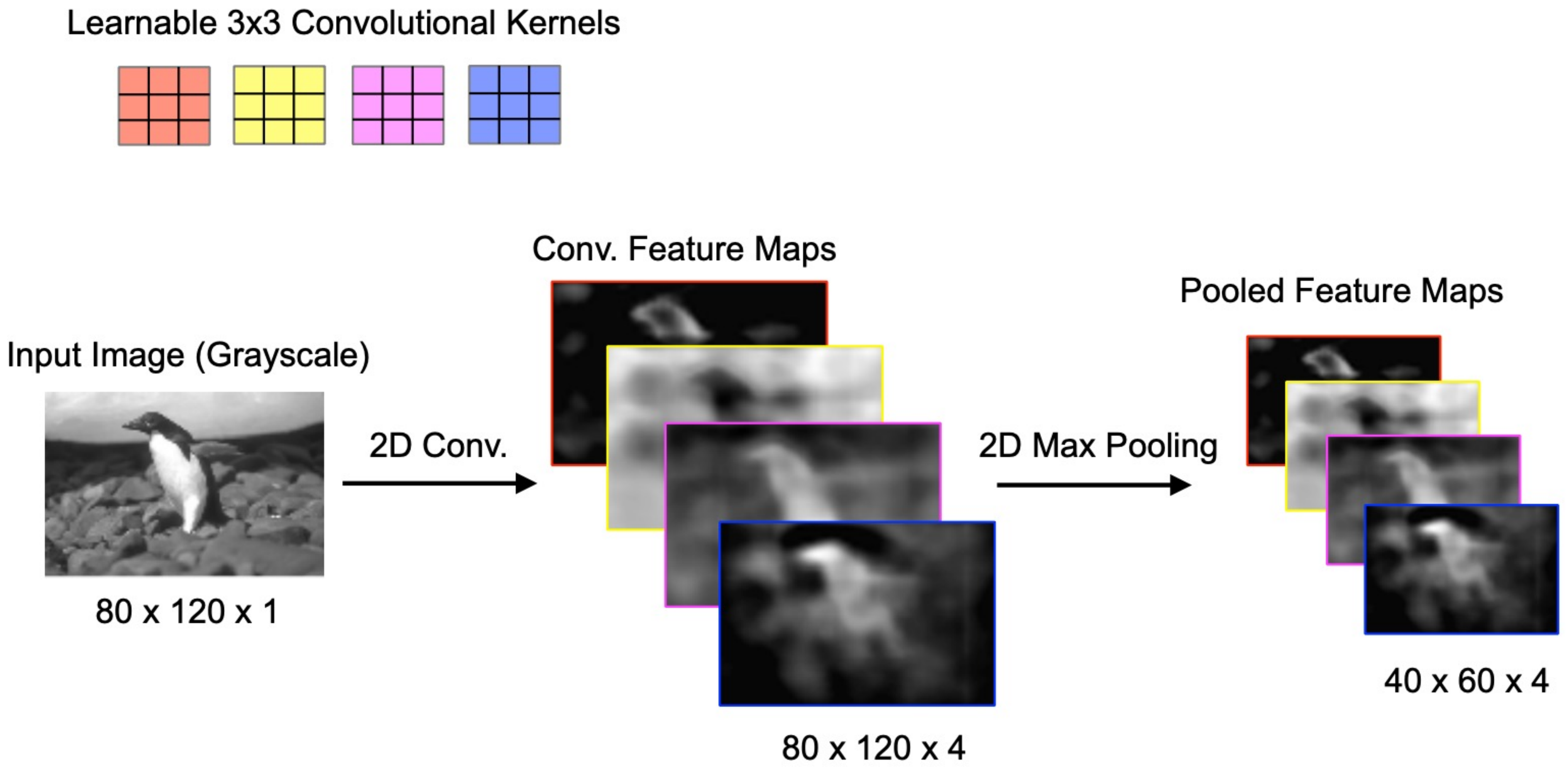
How Else to Shrink the Model Size?

Pooling Layer:

- Max Pooling
- Other pooling options like average pooling are also used

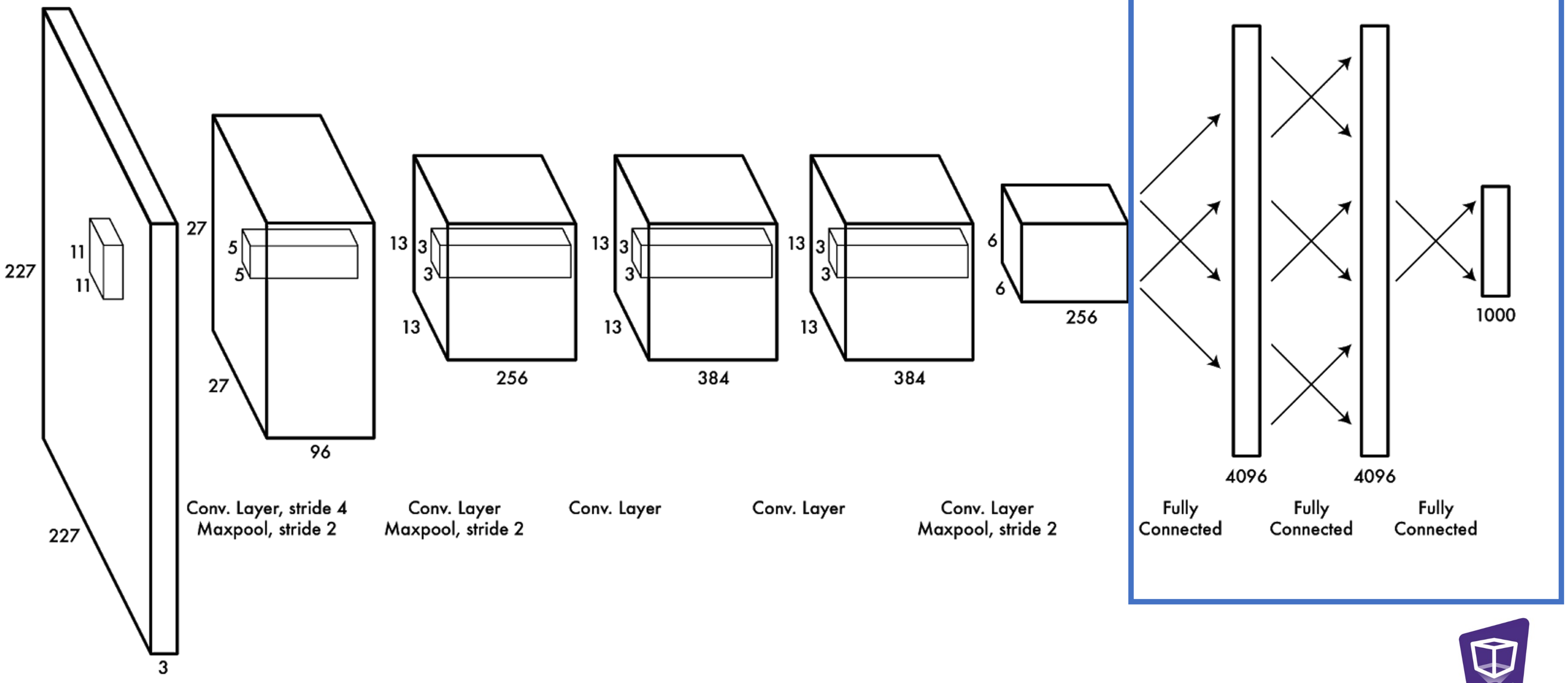


Convolutional Networks



Slide Credits: Gedas Bertasius

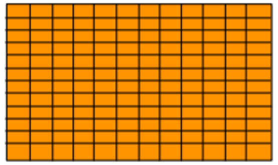
AlexNet: An Early Example



https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Convolutional Networks

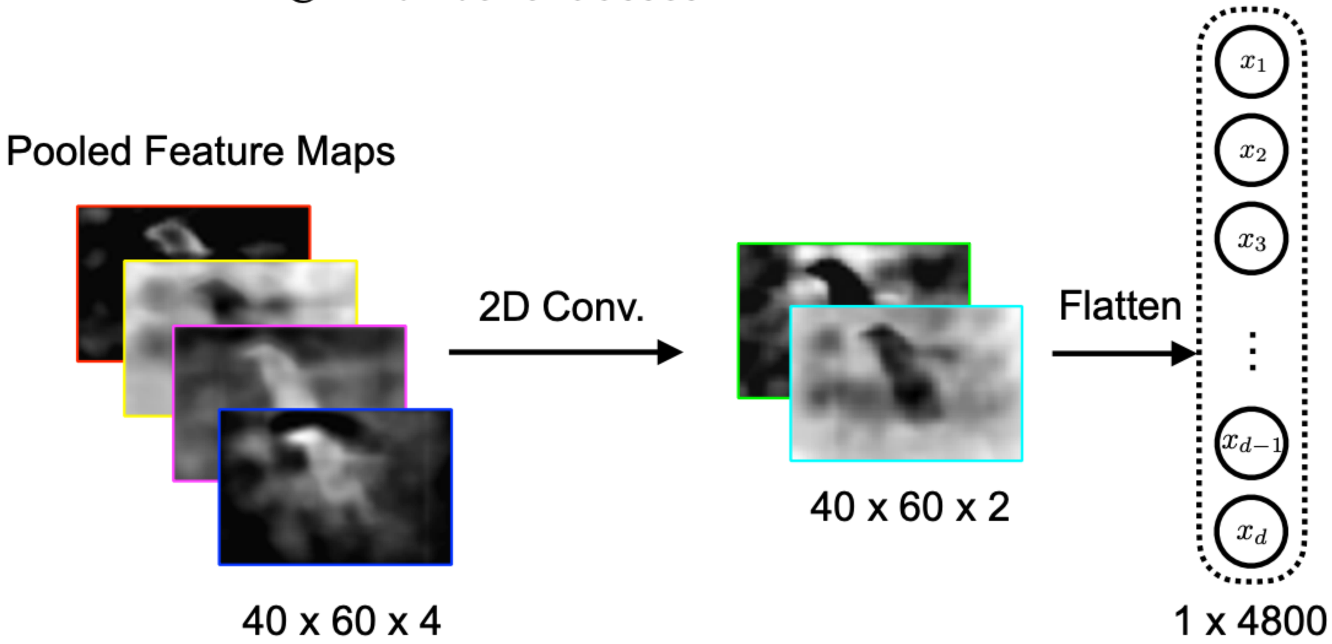
Learnable FC Layer Weight Matrix



$$W \in \mathbb{R}^{d \times C}$$

d - feature dimensionality (4800 in this example)

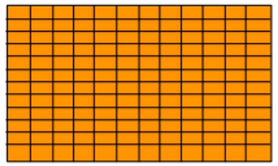
C - number of classes



Slide Credits: Gedas Bertasius

Convolutional Networks

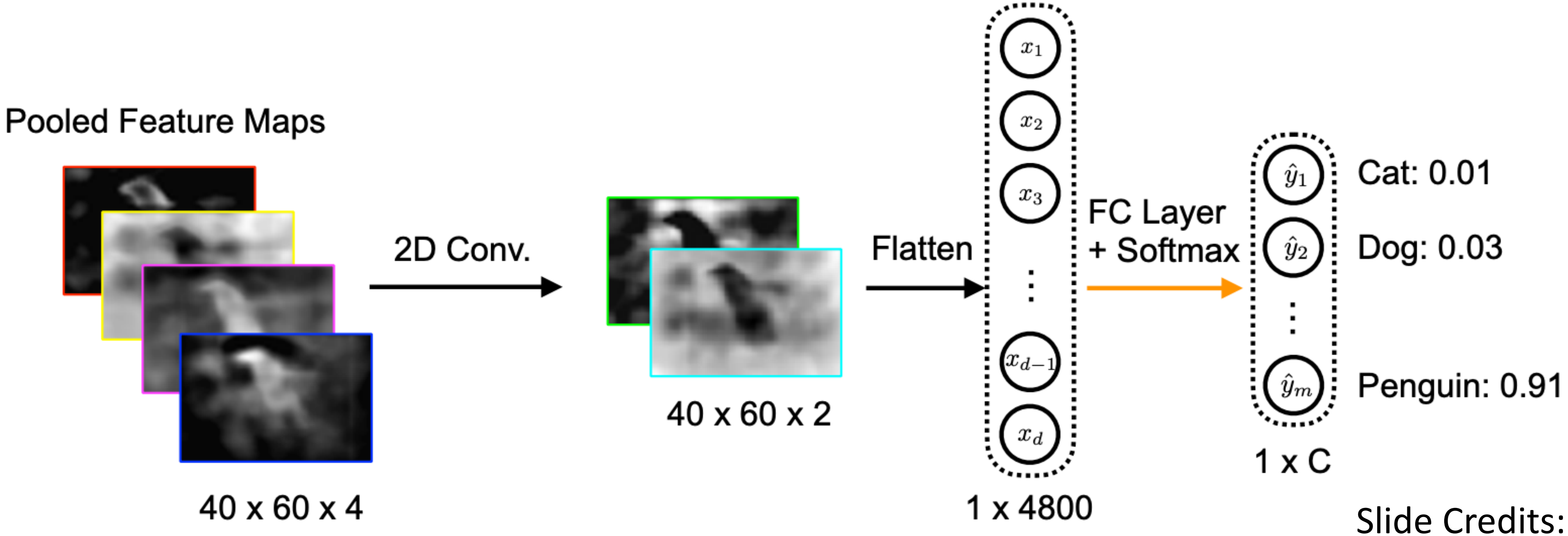
Learnable FC Layer Weight Matrix



$$W \in \mathbb{R}^{d \times C}$$

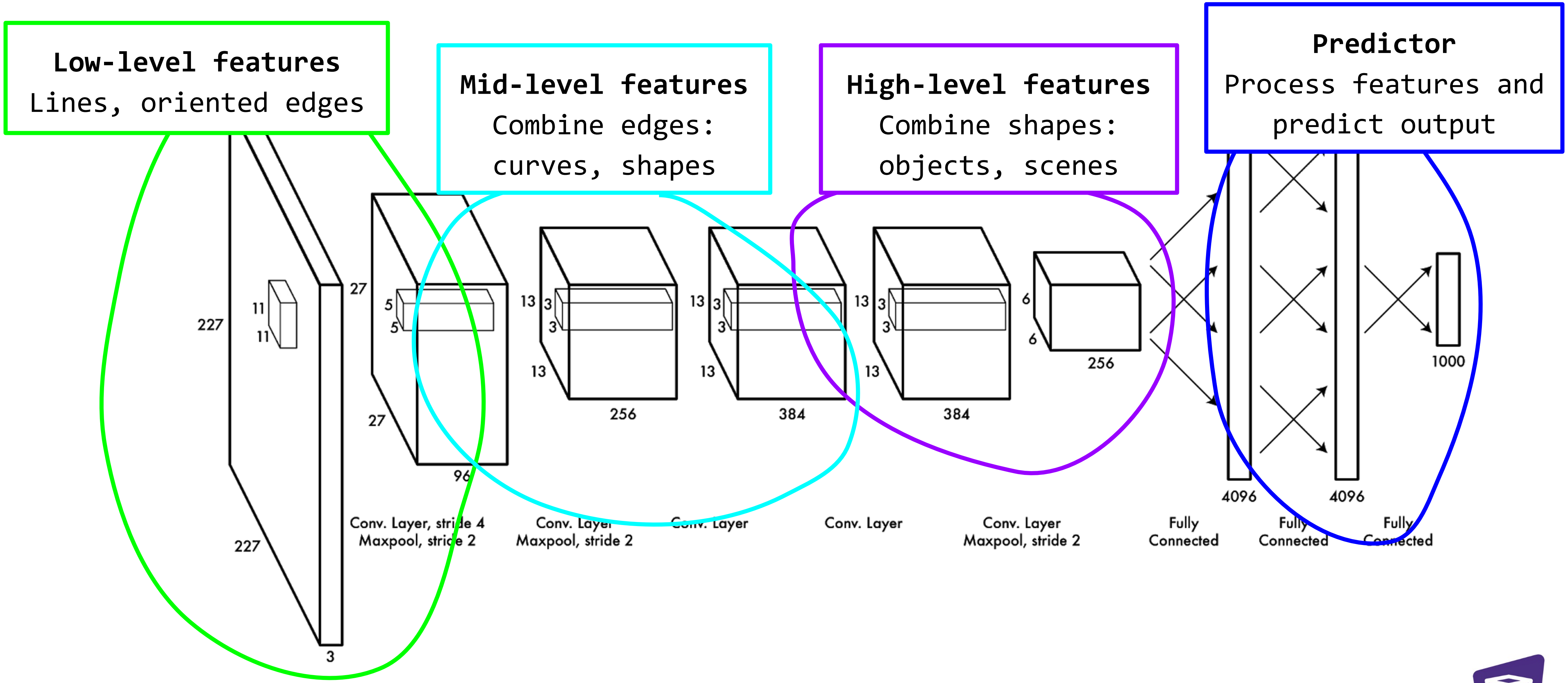
d - feature dimensionality (4800 in this example)

C - number of classes



Slide Credits: Gedas Bertasius

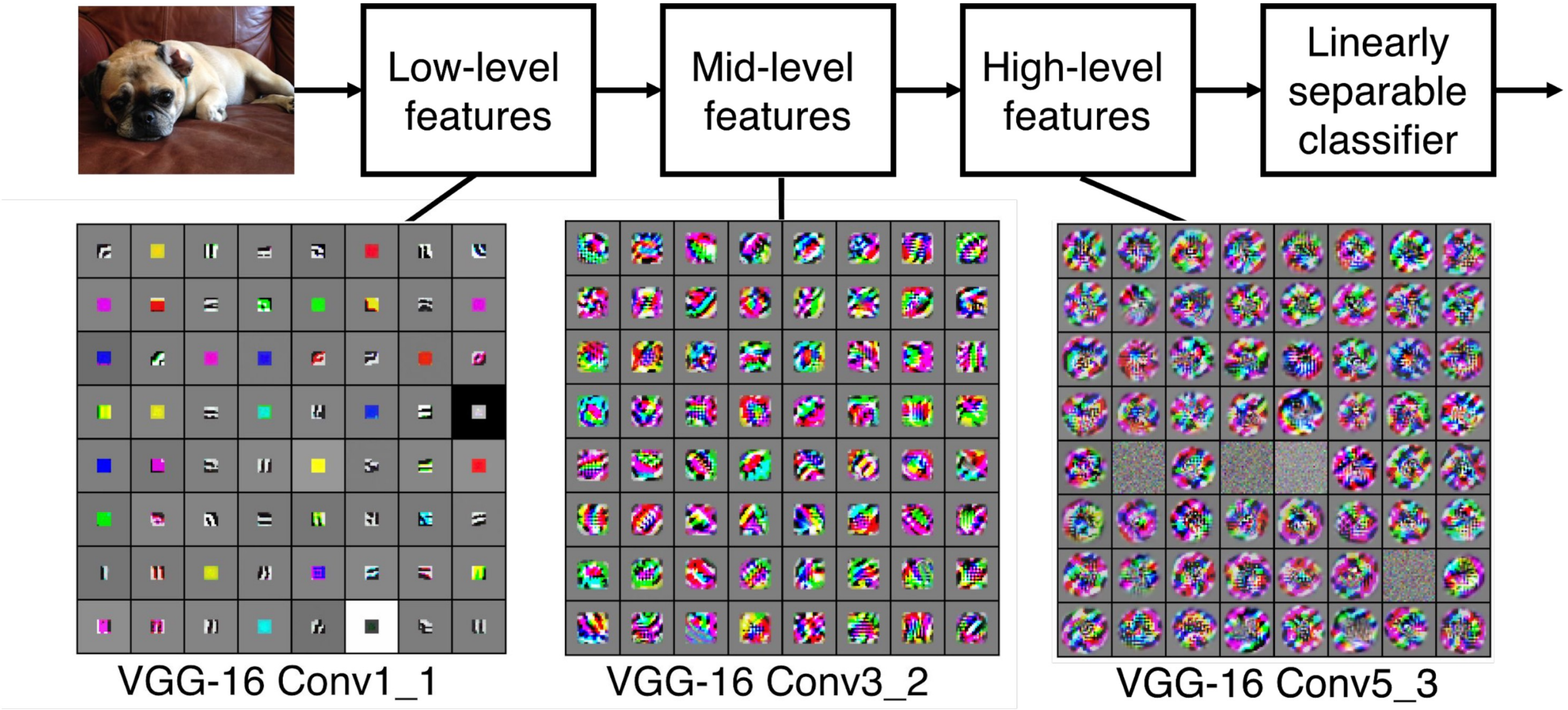
Where Models Learn Features of an Image



Preview

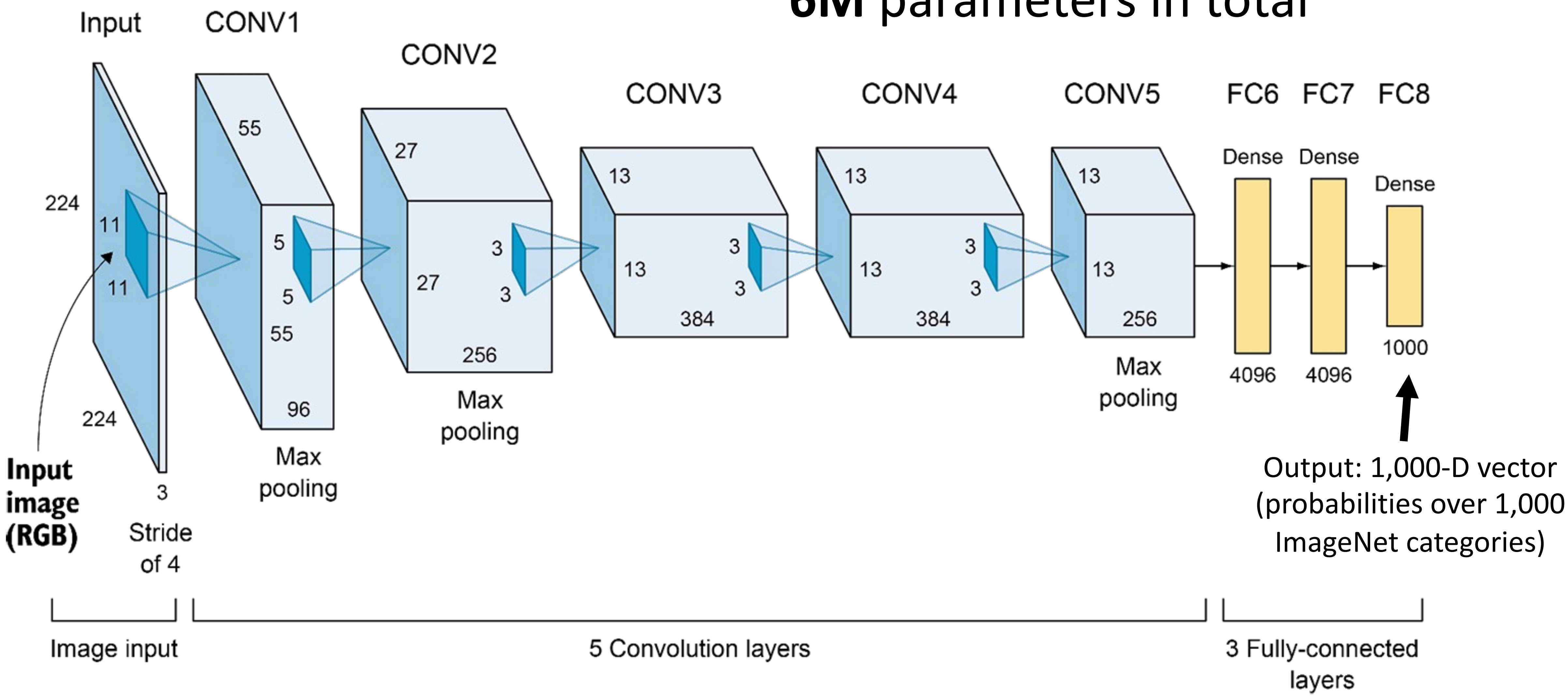
[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

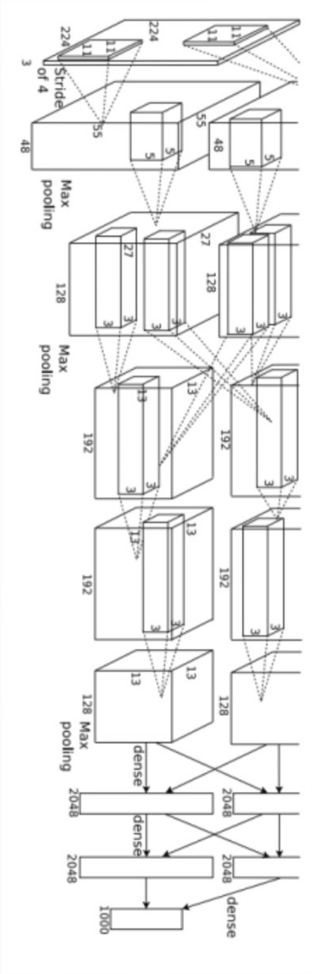
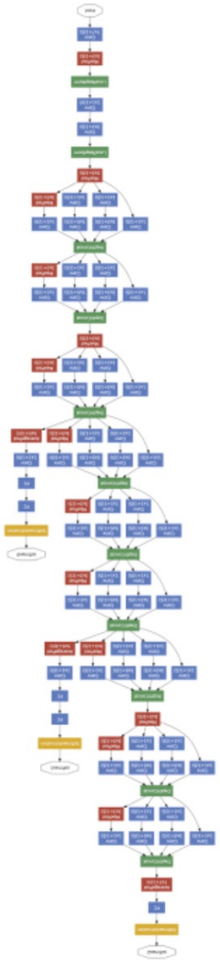
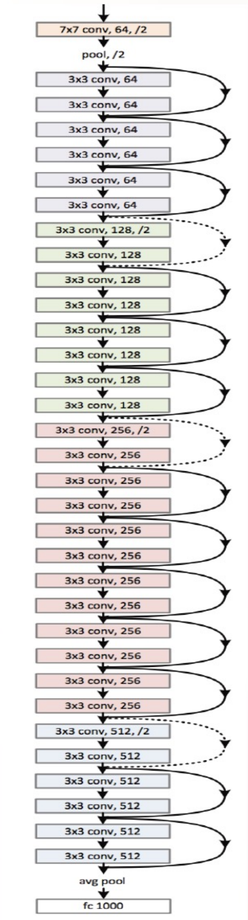


AlexNet (2012)

6M parameters in total



Elgendy, *Deep Learning for Vision Systems,* https://livebook.manning.com/book/grokking-deep-learning-for-computer-vision/chapter-5/v-3/

“AlexNet”	“GoogLeNet”	“VGG Net”	“ResNet”
			
[Krizhevsky et al. NIPS 2012]	[Szegedy et al. CVPR 2015]	[Simonyan & Zisserman, ICLR 2015]	[He et al. CVPR 2016]

Big picture

- A convolutional neural network can be thought of as a function from images to class scores
  - With millions of adjustable weights…
  - … leading to a very non-linear mapping from images to features / class scores.
  - We will set these weights based on classification accuracy on training data…
  - … and hopefully our network will generalize to new images at test time

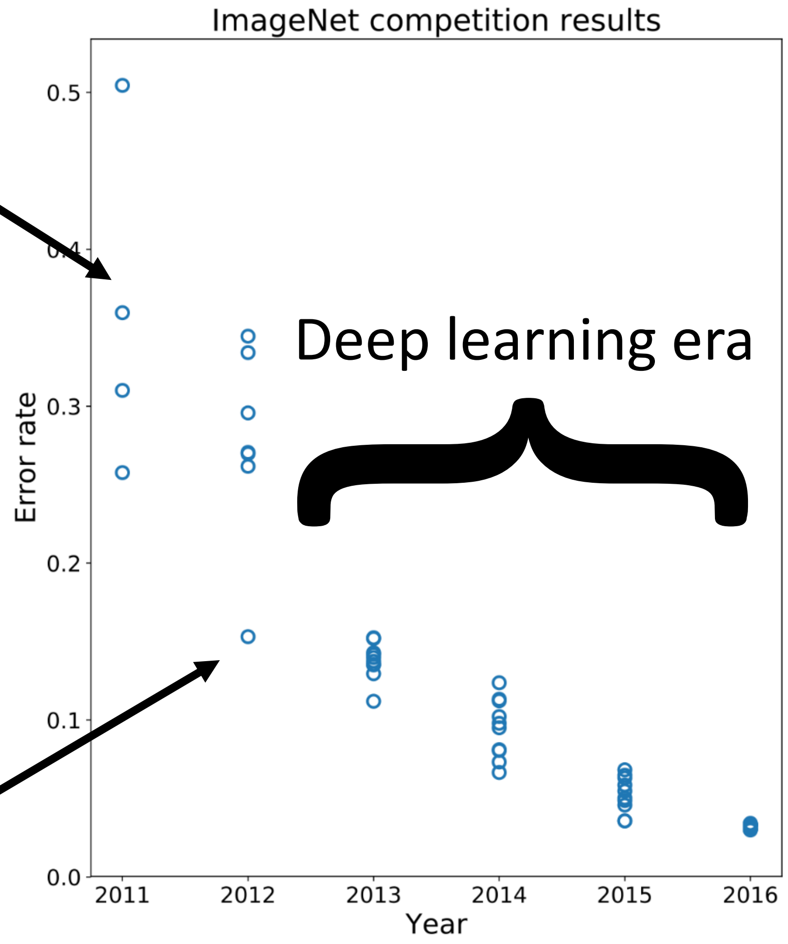
Data is key—enter ImageNet

- ImageNet (and the ImageNet Large-Scale Visual Recognition Challege, aka **ILSVRC**) has been key to training deep learning methods
  - J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. *CVPR,* 2009.
- **ILSVRC**: 1,000 object categories, each with ~700-1300 training images. Test set has 100 images per categories (100,000 total).
- Standard ILSVRC error metric: top-5 error
  - if the correct answer for a given test image is in the top 5 categories, your answer is judged to be correct

Performance improvements on ILSVRC

- ImageNet Large-Scale Visual Recognition Challenge
- Held from 2011-2017
- 1000 categories, 1000 training images per category
- Test performance on held-out test set of images

Pre-deep learning era



AlexNet

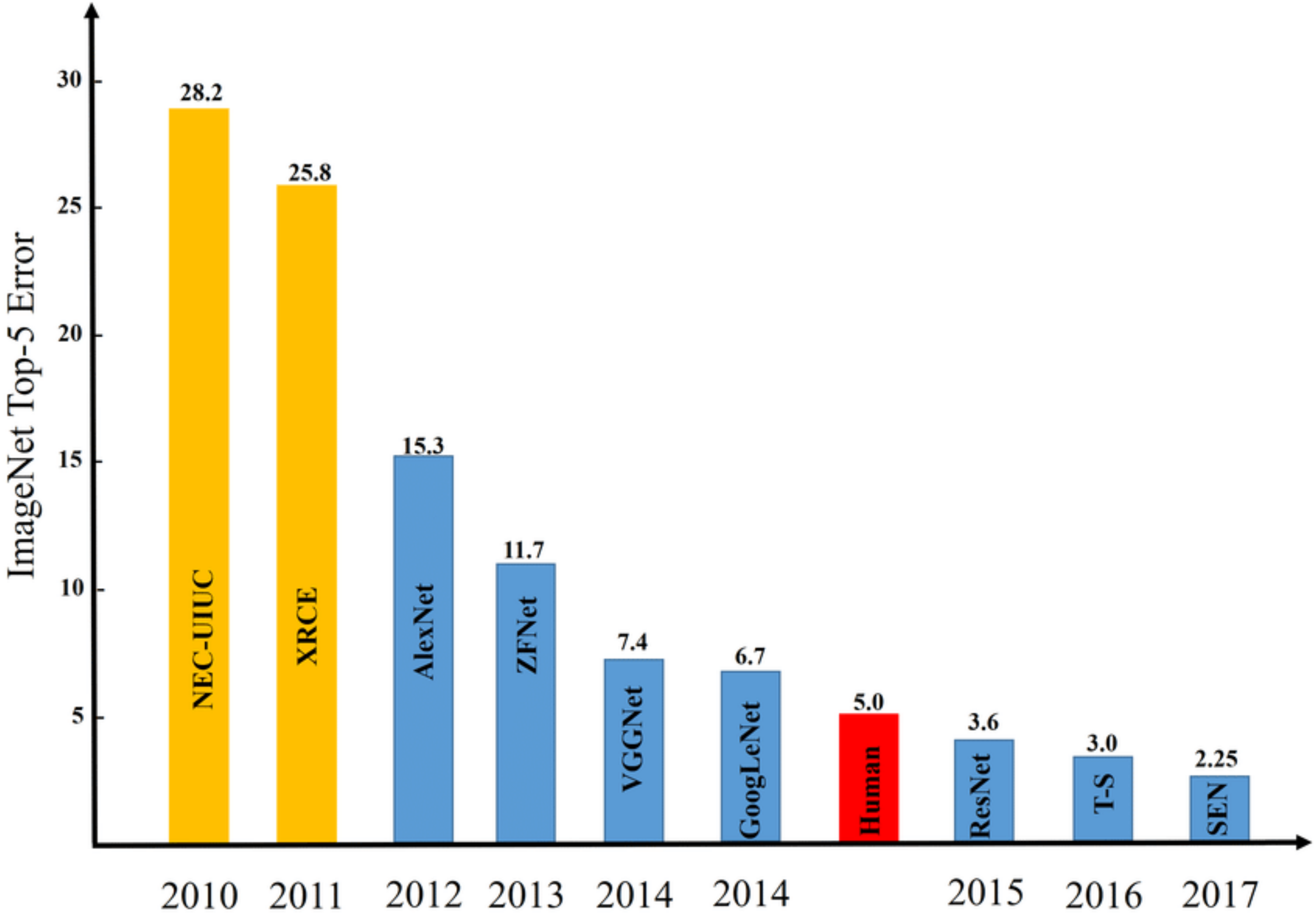
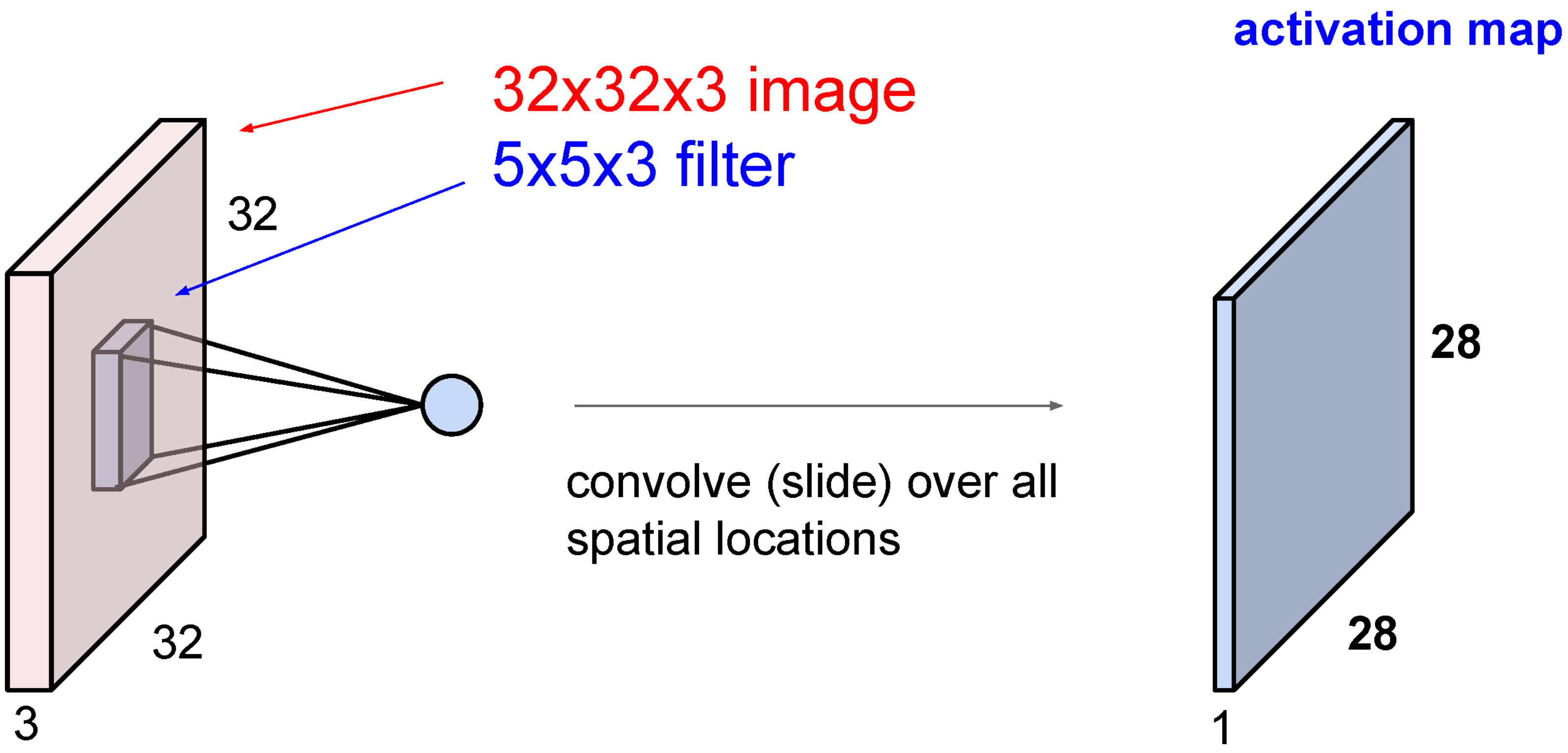


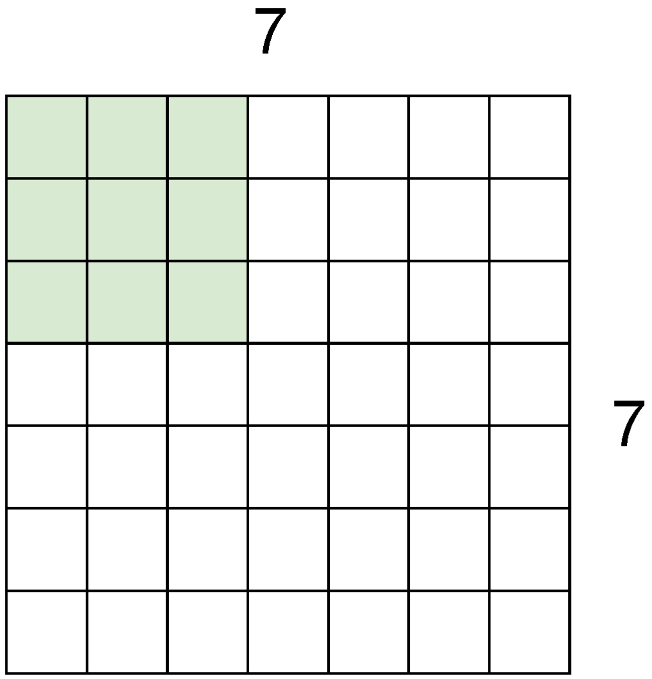
Image credit: Zaid Alyafeai, Lahouari Ghouti

Closer look at Convolution
(Extra Slides)

A closer look at spatial dimensions:

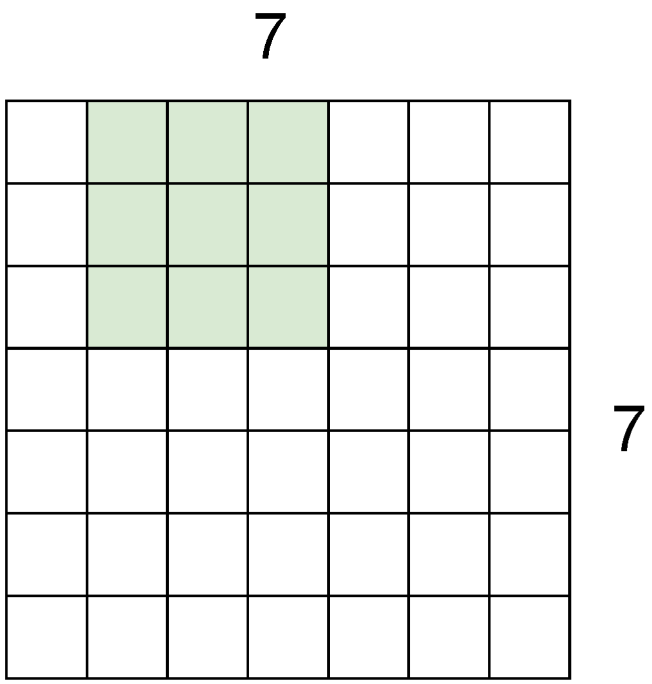


A closer look at spatial dimensions:



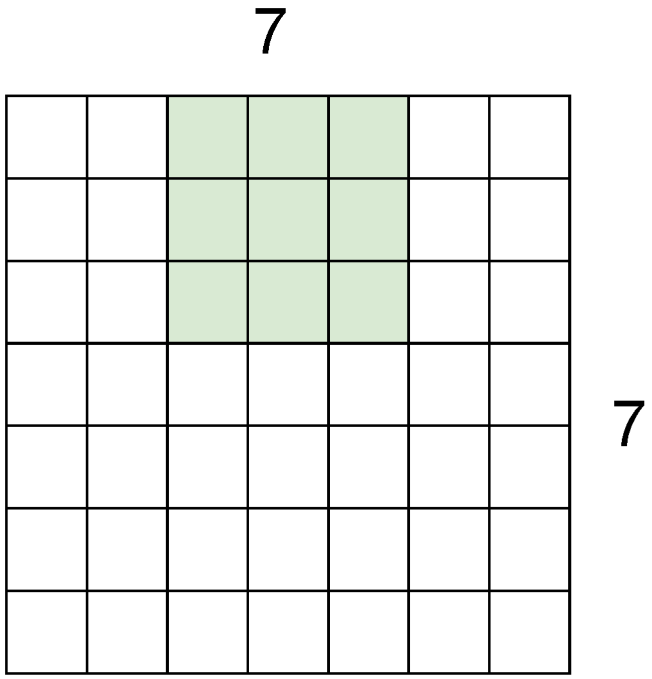
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



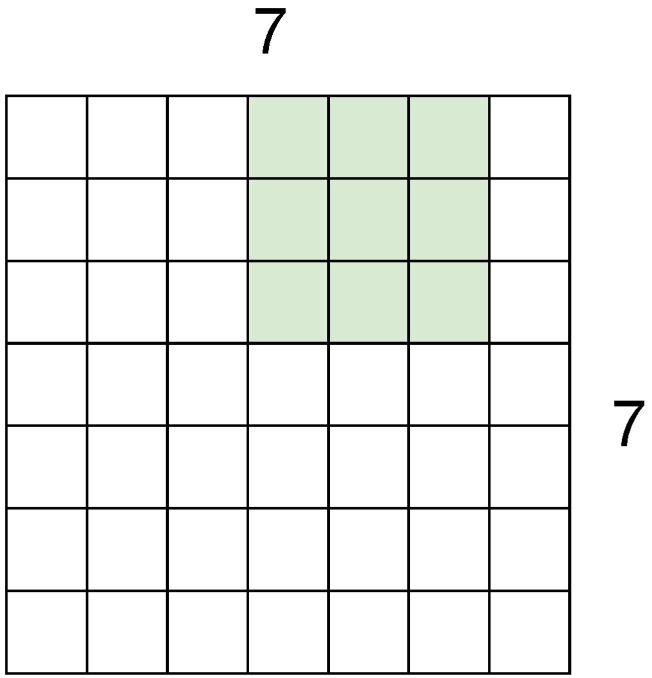
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



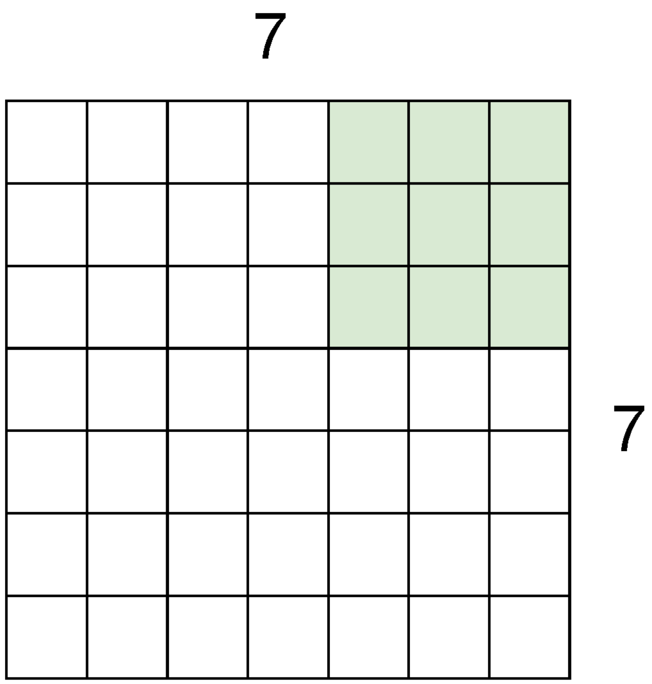
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

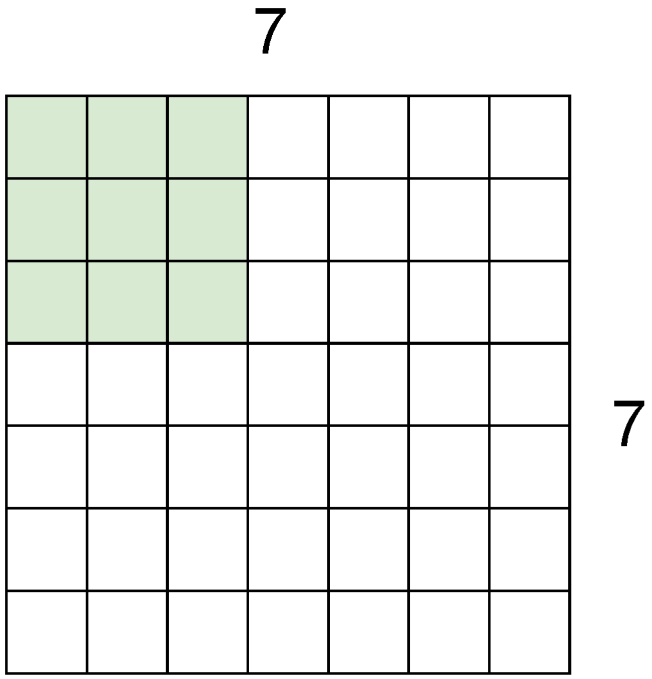
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

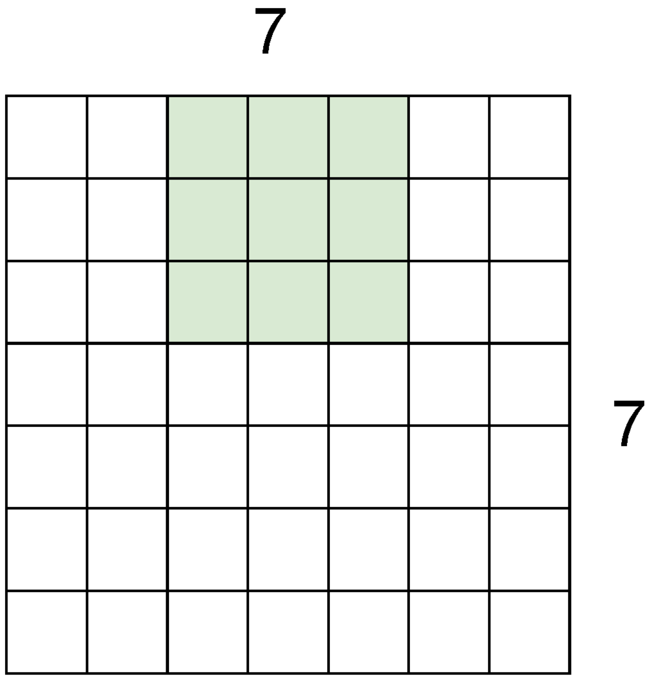
=> 5x5 output

A closer look at spatial dimensions:



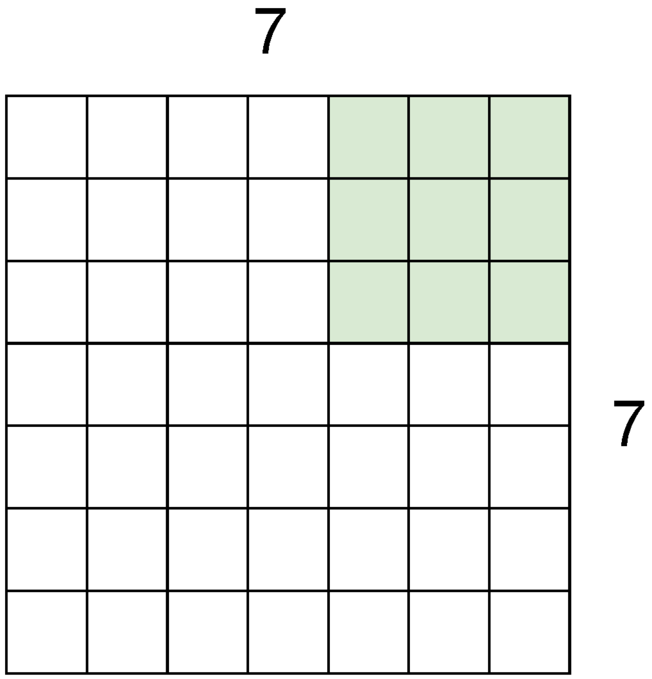
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



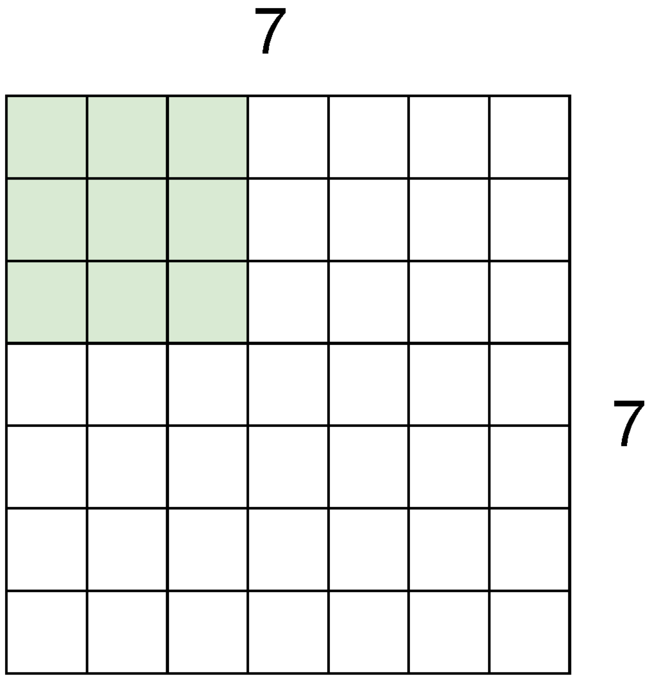
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



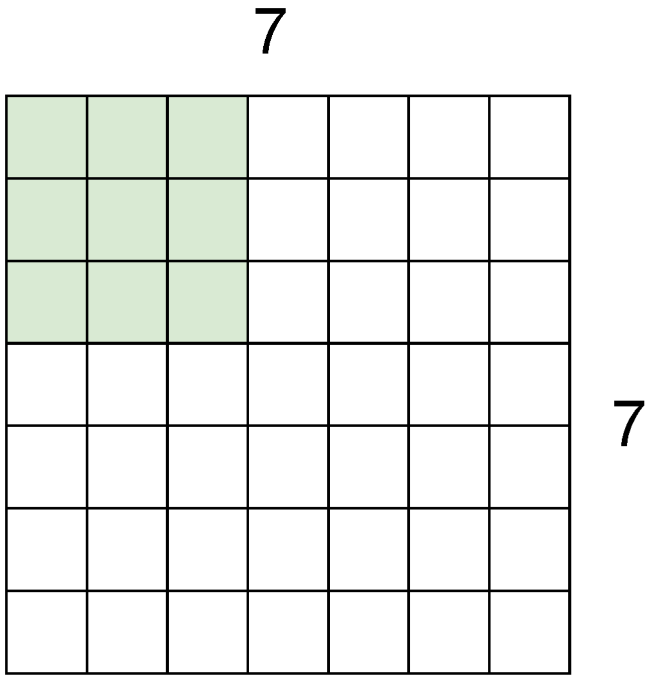
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



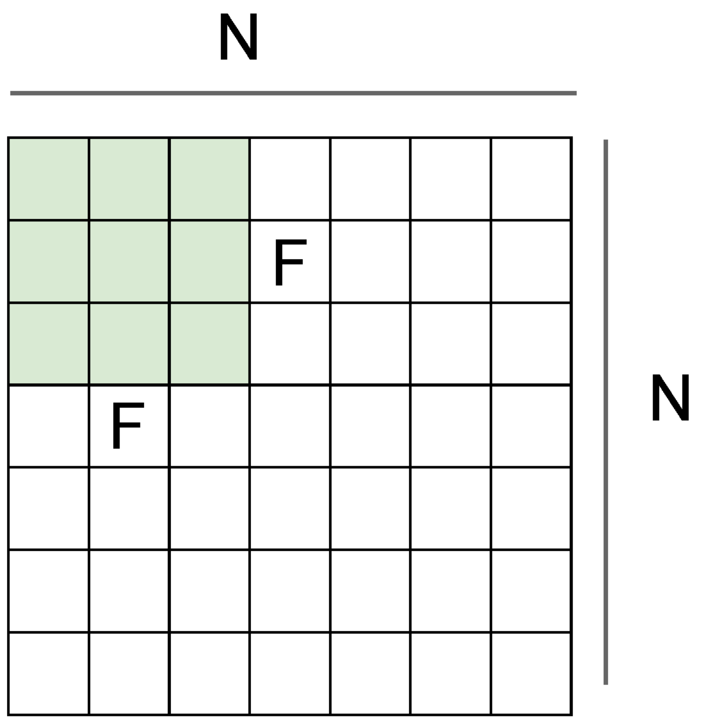
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

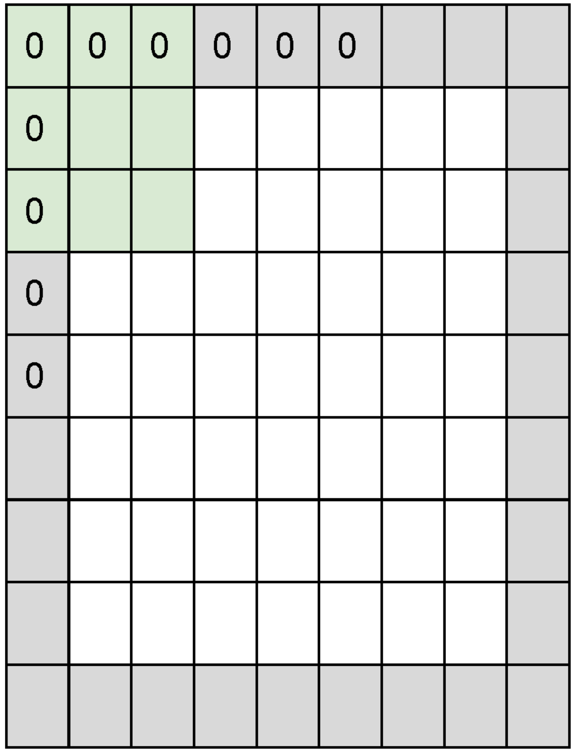
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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 = 2.33 :\

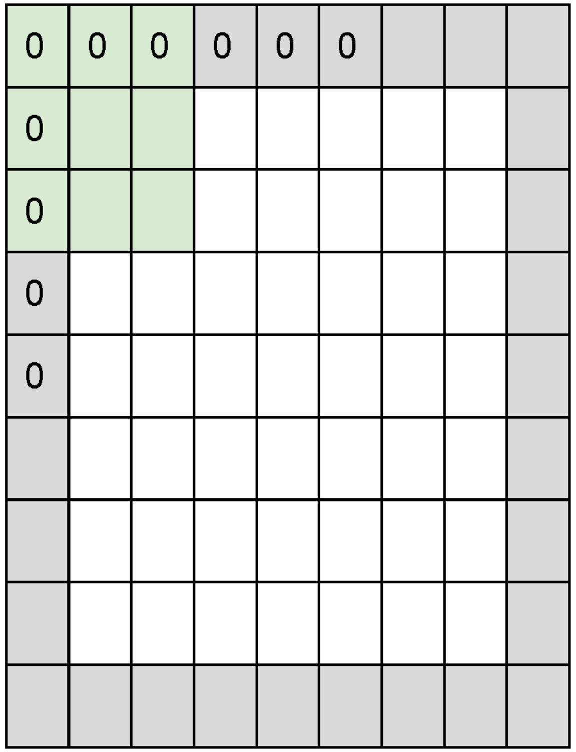
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

In practice: Common to zero pad the border

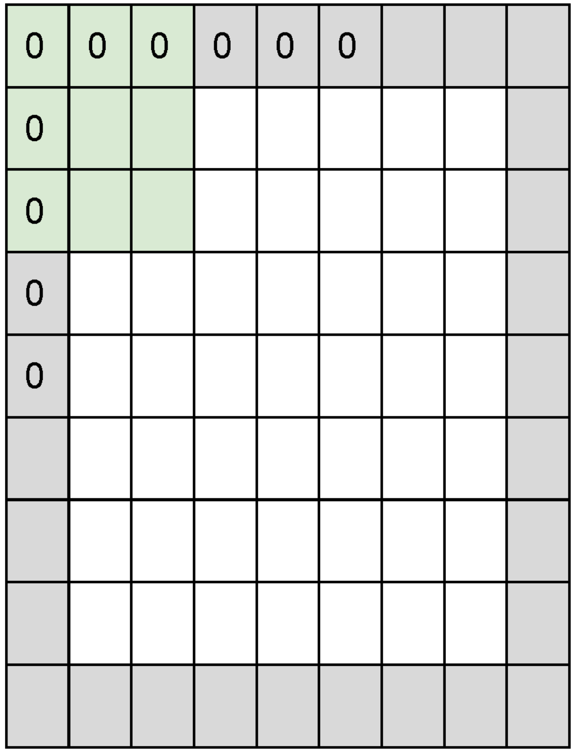


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

Output filter size: (N + 2*pad – F)/S + 1

In practice: Common to zero pad the border

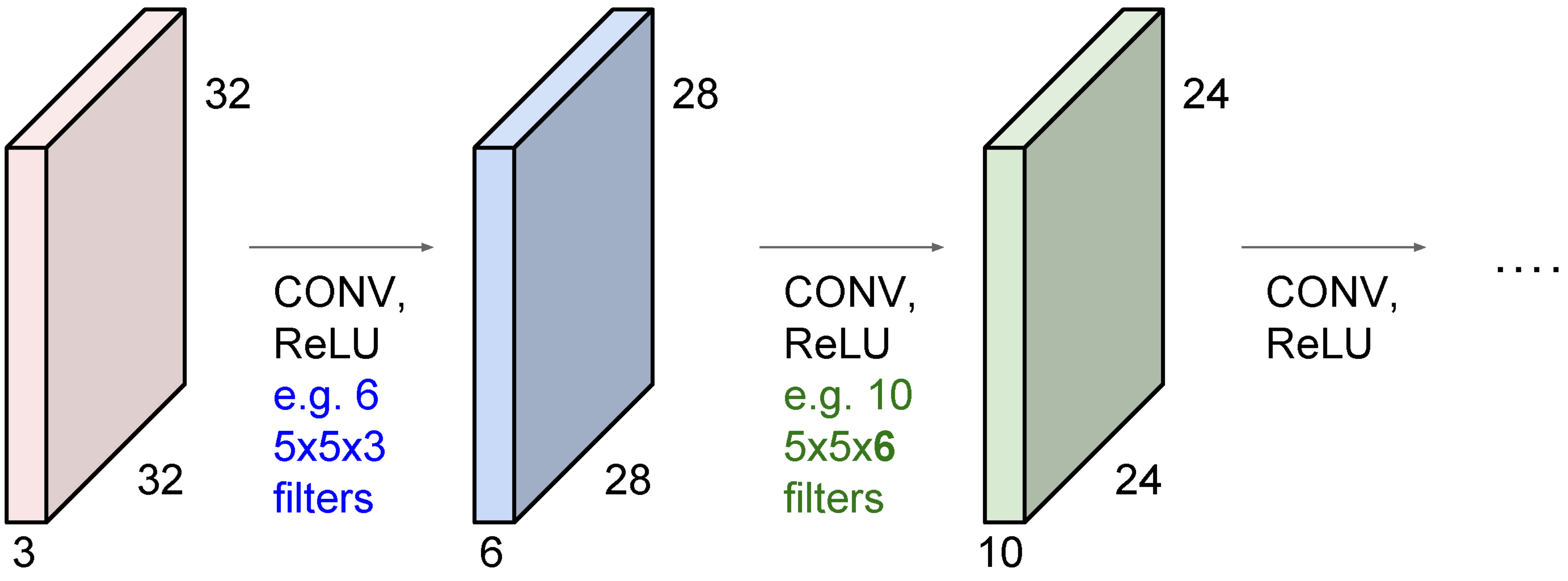


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Remember back to…

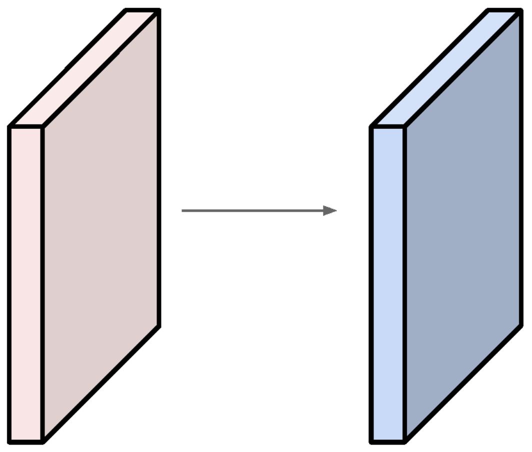
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

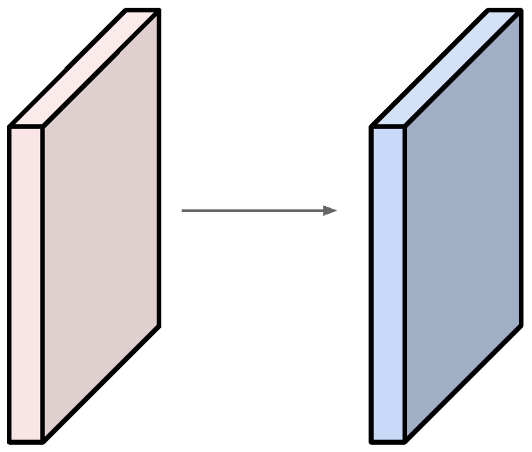
Output volume size: ?



Examples time:

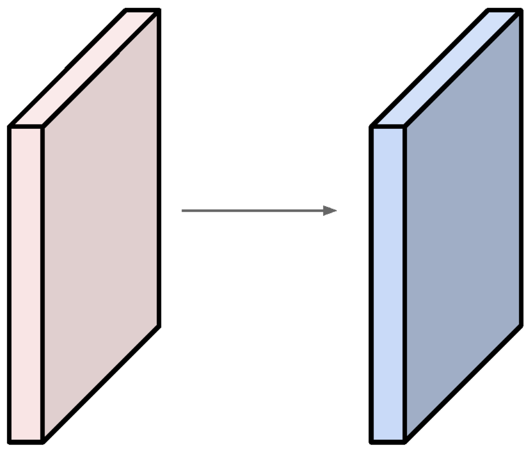
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10



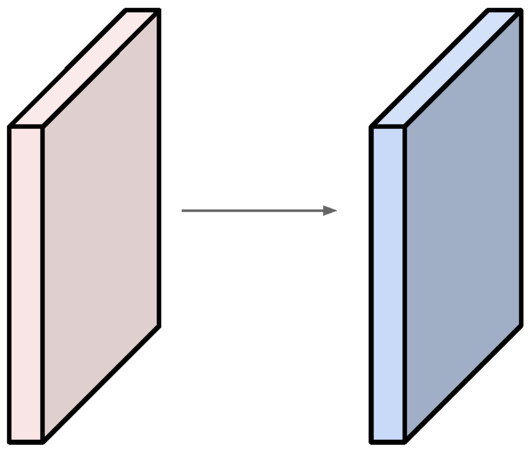
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

Examples time:



Input volume: **32x32x3**

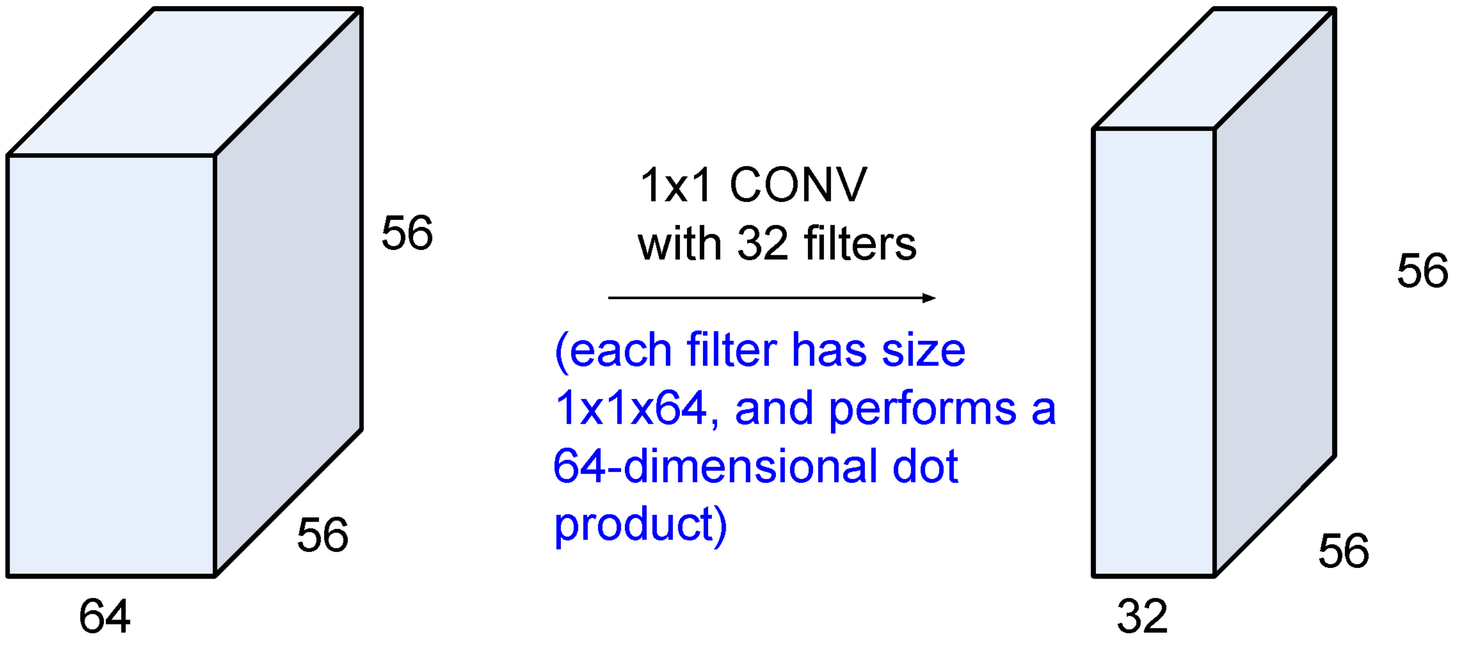
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

(btw, 1x1 convolution layers make perfect sense)



Convolutional layer—properties

- Small number of parameters to learn compared to a fully connected layer
- Preserves spatial structure—output of a convolutional layer is shaped like an image
- **Translation equivariant**: passing a translated image through a convolutional layer is (almost) equivalent to translating the convolution output (but be careful of image boundaries)

Self-study

[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

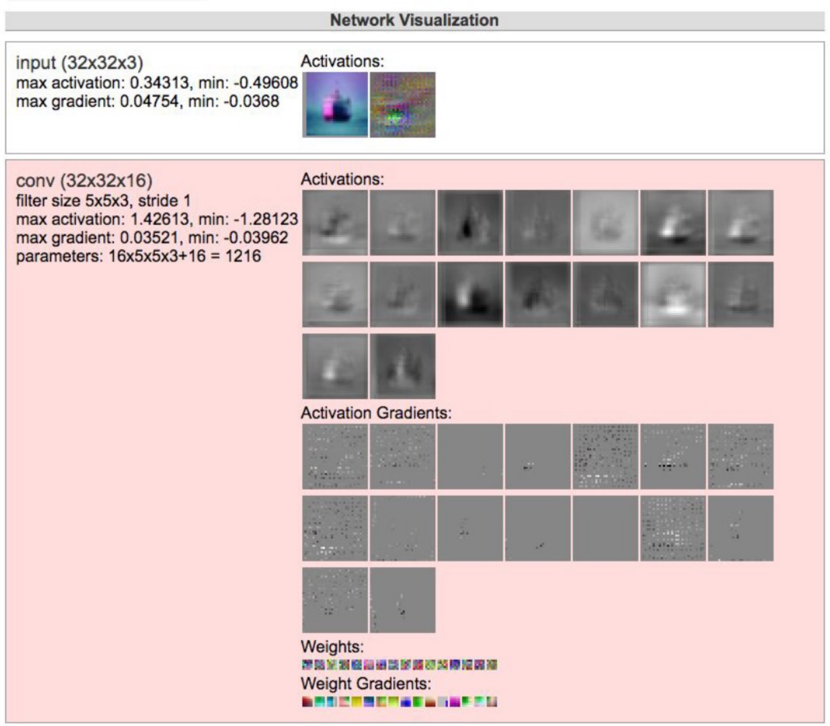
Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vercially.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.



https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Train a neural network for classification on CIFAR10 dataset in google colab

[Google Colab Page]

Where to Look for More Information

- Explore existing computer vision and machine learning frameworks
  - https://pytorch.org/
  - https://www.tensorflow.org/
  - https://keras.io/
  - https://opencv.org/
- Watch more in-depth lecture series
  - The Ancient Secrets of Computer Vision - Joseph Redmon
  - Deep Learning Specialization - Andrew Ng
- Checkout other online courses and guides
  - https://ai.google/education/
  - https://www.udacity.com/course/deep-learning-pytorch--ud188

Slide Credits

- CS5670, Introduction to Computer Vision, **Cornell Tech, by Noah Snavely.**
- **CS 543 Computer Vision, by Stevlana Lazebnik, UIUC.**
- **EECS 442 Computer Vision, by Justin Johnson & David Fouhey, U Michigan.**