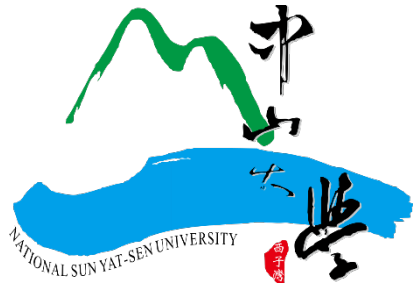
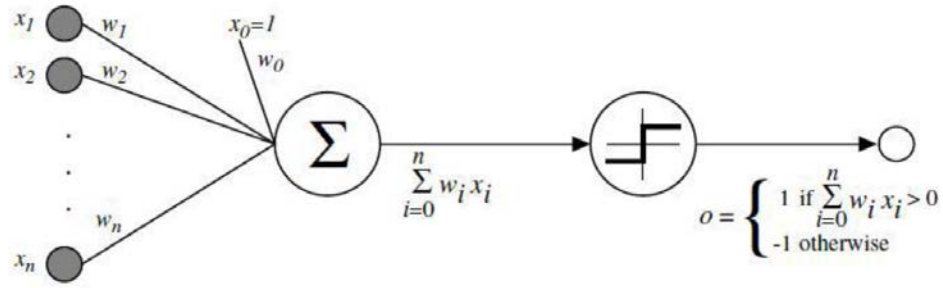
Introduction to Convolutional Neural Network (CNN)



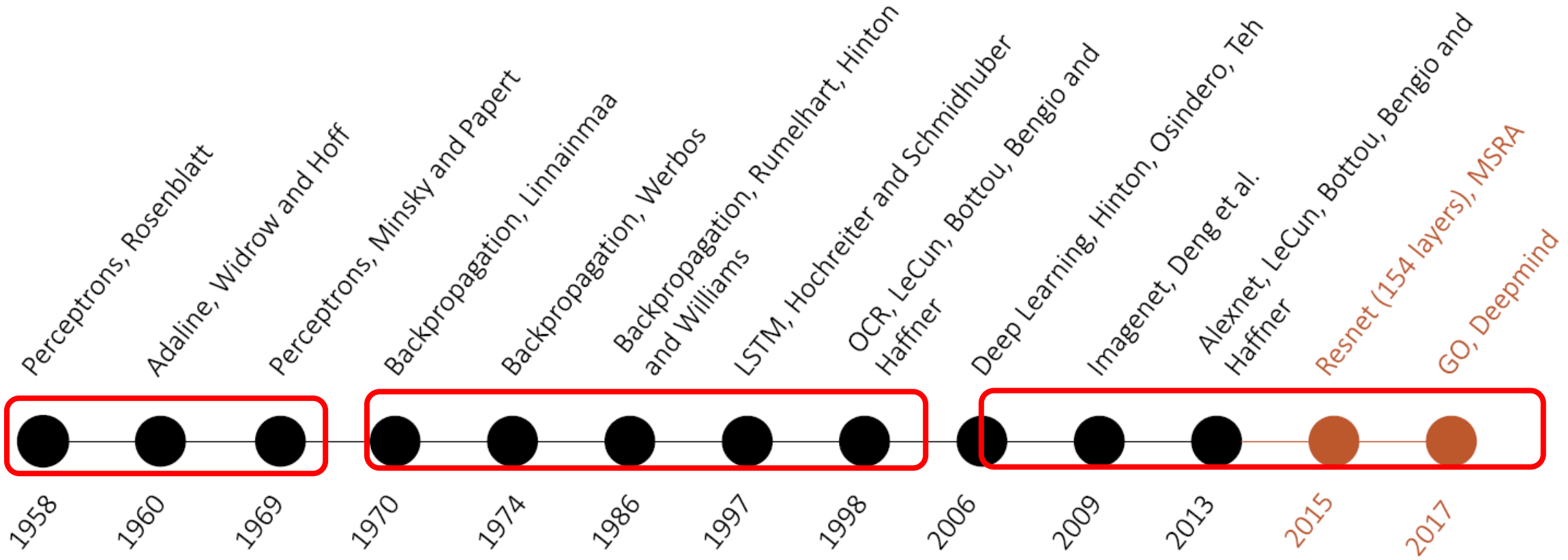
Outlines

- AI History
- Artificial Neural Network (ANN)
- Convolutional Neural Network (CNN)

Evolution of AI



- first appeared in 1960s: perception for binary decision
- 1970~2000: backpropagation, recurrent nets with few layers
- 2005~: deep learning with many layers



Three Waves of AI

- 1st wave: Handcrafted Knowledge (1980's)
  - create sets of **rules** to represent knowledge in well-defined domain (expert system)
  - no learning capability
- 2nd wave: Statistical Learnings (2010's)
  - create **statistical models** for specific problem domains and train them on big data
  - nuanced classification and prediction capability
  - limited explainable capability through visualization
  - no contextual capability
- 3rd wave: Contextual Adaptation (2020's)
  - construct **contextual explanatory** models
  - Artificial General Intelligence (AGI)
  - DARPA AI-Next projects in late 2018

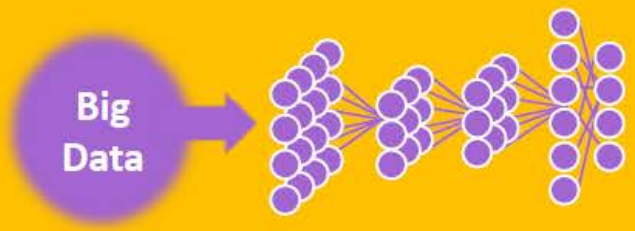
Three Waves of AI

Handcrafted Knowledge	Statistical Learning	Contextual Adaptation
First Wave	Second Wave	Third Wave
Expert Systems	Deep Learning, CNN, RNN, GAN, Explainable AI	Symbiosis AI, AGI*

* Artificial General Intelligence

Human create
sets of rules
to represent knowledge
in well-defined domains

Human create
statistical models
for specific problem domains
and train them on big data



Systems construct
contextual explanatory models
for classes of
real world phenomena

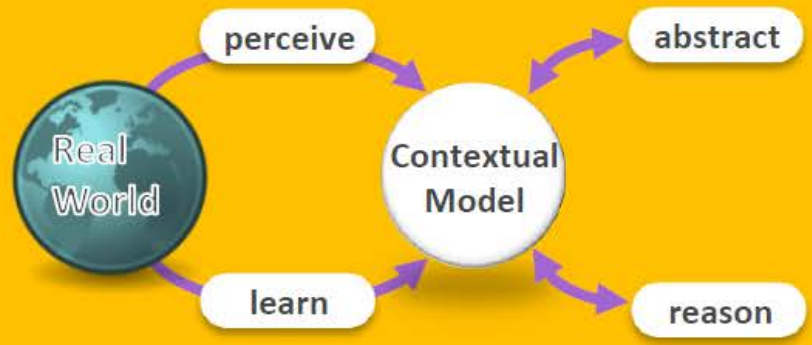
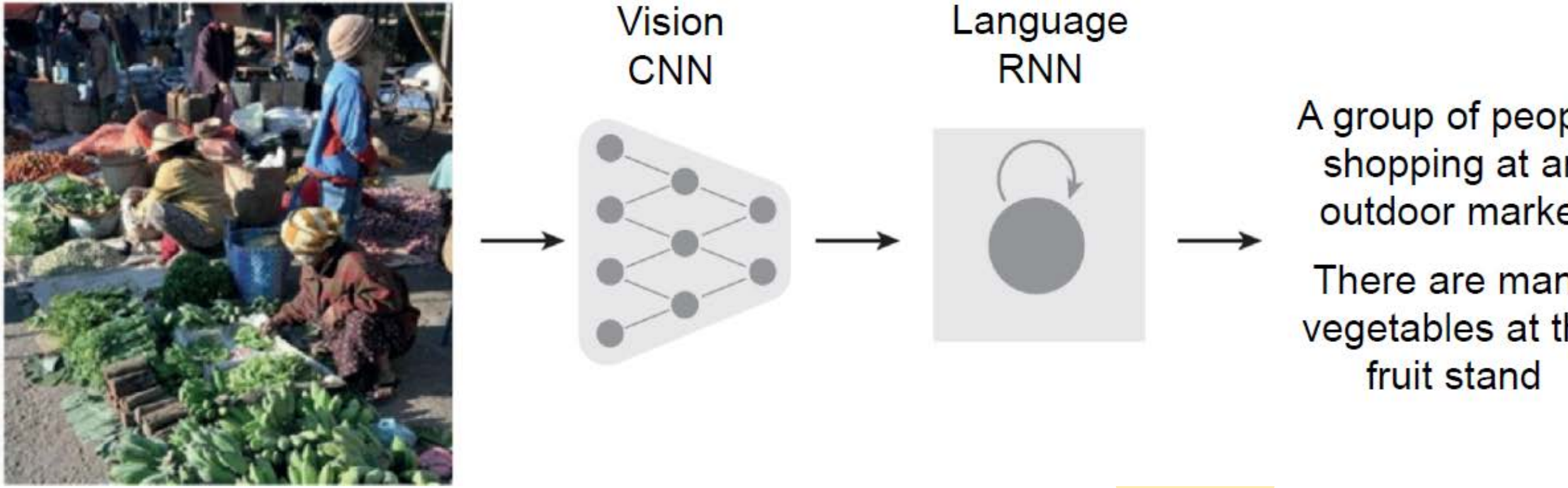


Image Captioning Examples

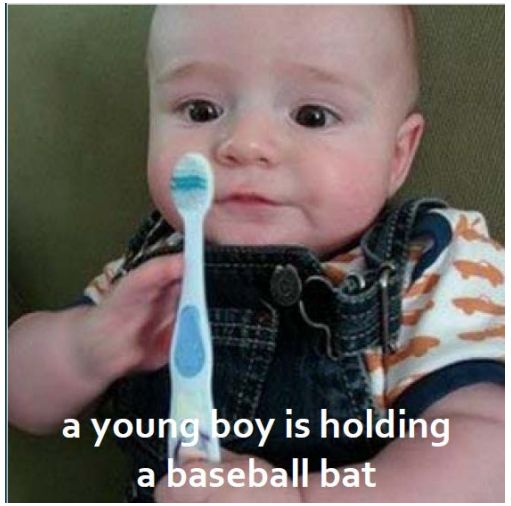
- a deep convolution neural net (CNN) produces a set of “words”
- a language-generating recurrent neural net (RNN) “translate” the words into captions



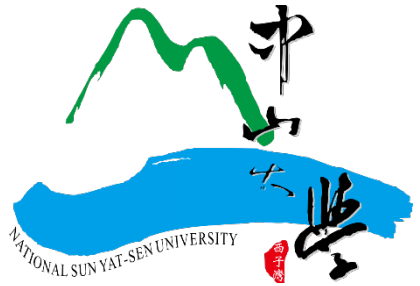
Challenges with 2nd Wave AI

- Generative Adversarial Network (GAN)

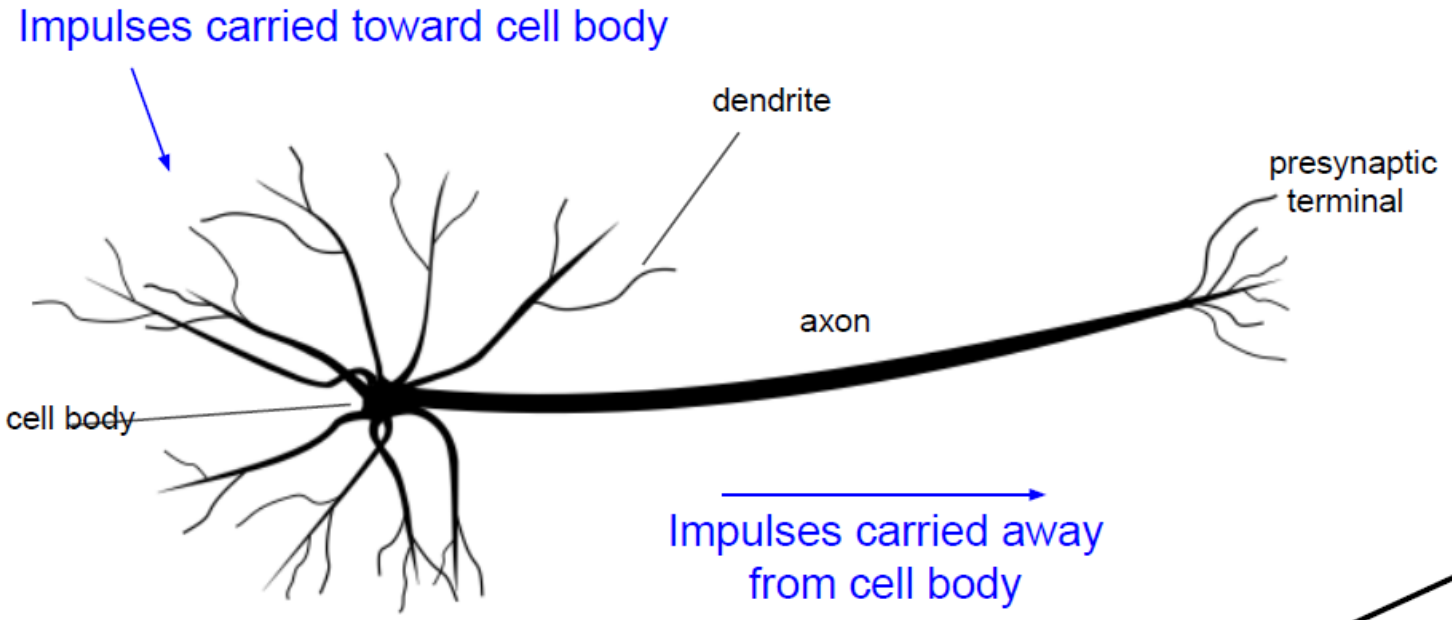
- classification and image captioning

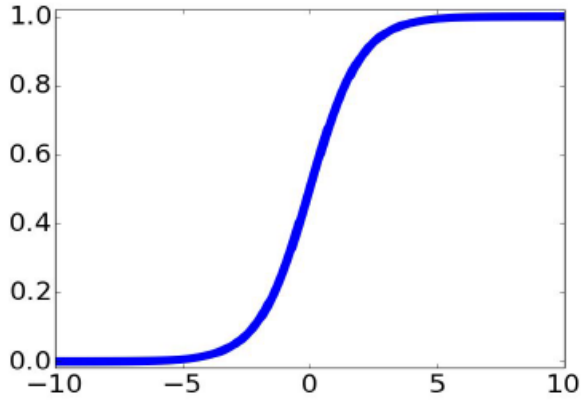


ANN



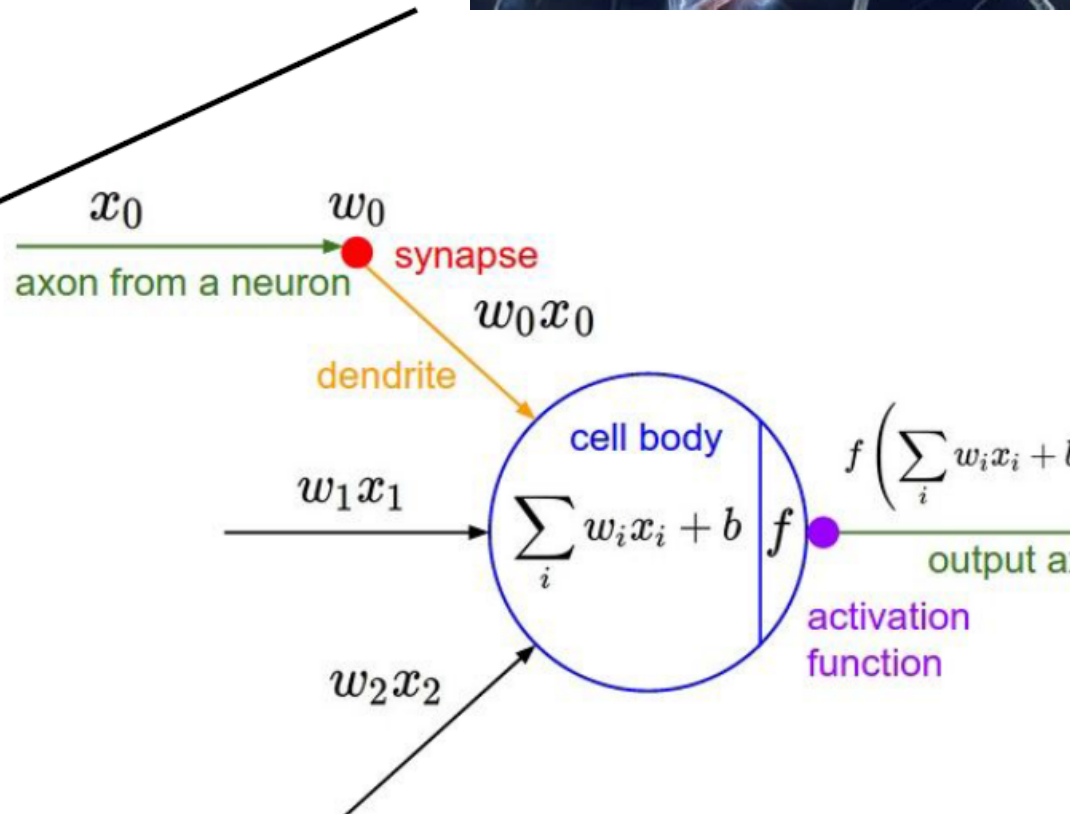
Neuron Biology





sigmoid activation function

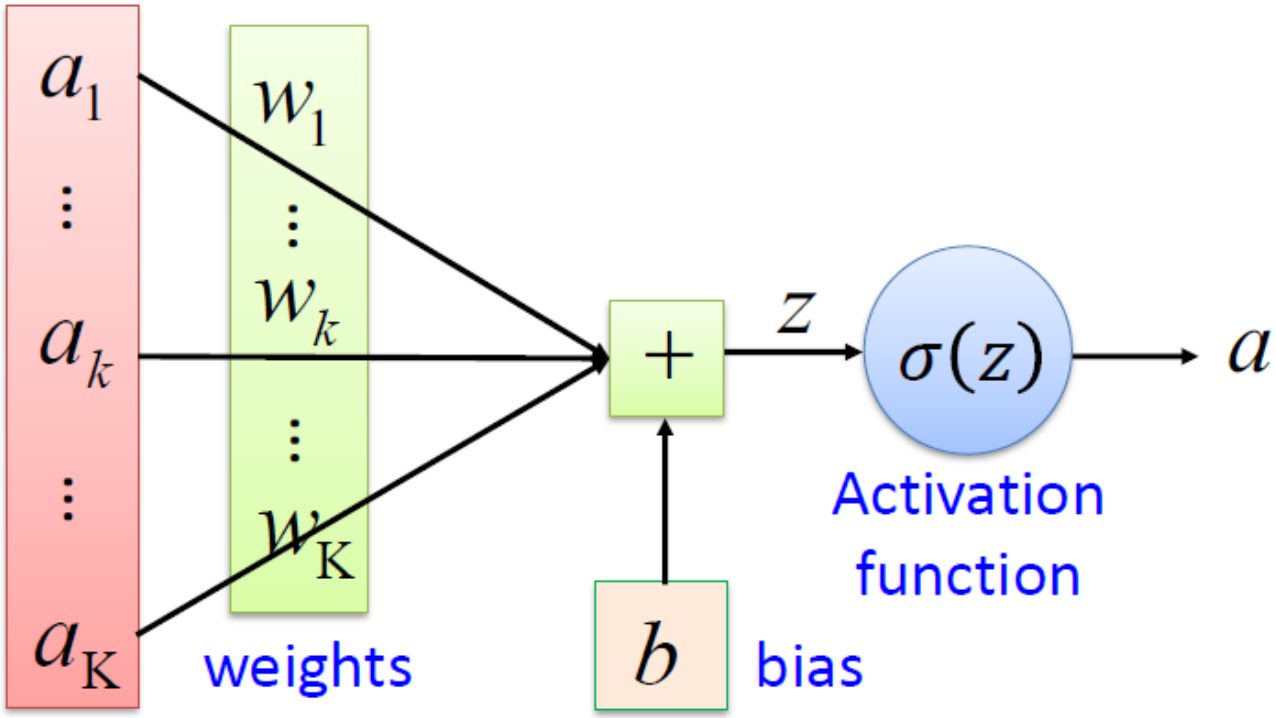
$$\frac{1}{1+e^{-x}}$$



Model for Artificial Neuron

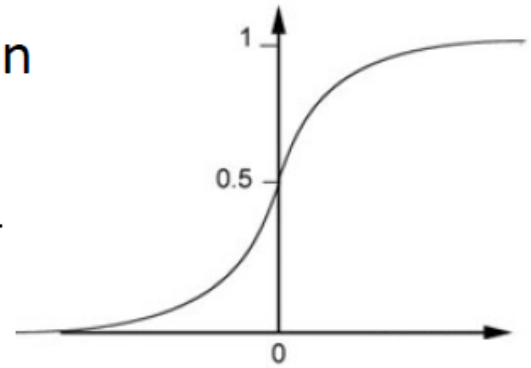
- weighted sum of inputs followed by non-linear activation function

$$z = a_1 w_1 + \cdots + a_k w_k + \cdots + a_K w_K + b$$



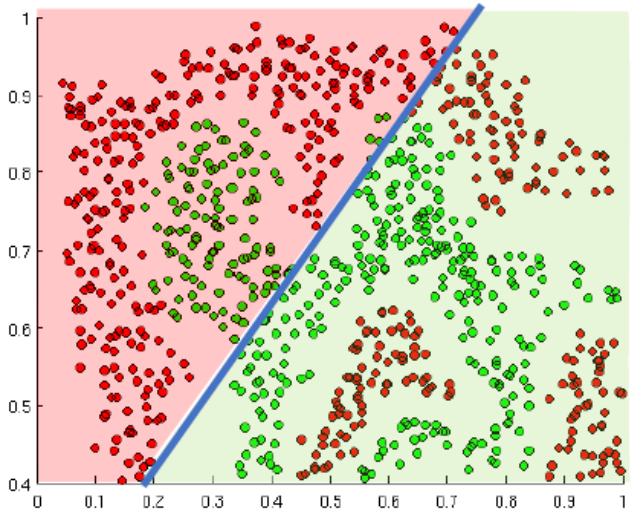
Sigmoid Function

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



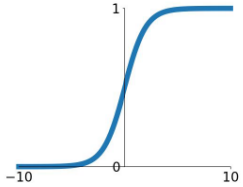
Non-linear Activation Functions

- introduce non-linearity into networks
  - for classification purpose
    - non-linearity approximates complex functions
- ANN usually uses tanh or sigmoid
- CNN usually uses ReLU (Rectified Linear Unit)
- GAN encoder usually uses leaky ReLU



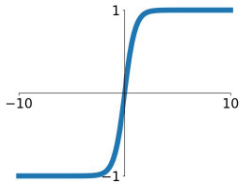
Sigmoid

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



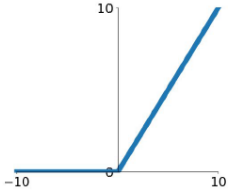
tanh

$$\tanh(x)$$



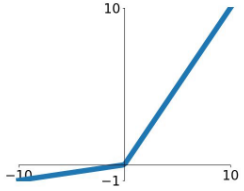
ReLU

$$\max(0, x)$$



Leaky ReLU

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

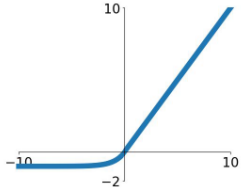


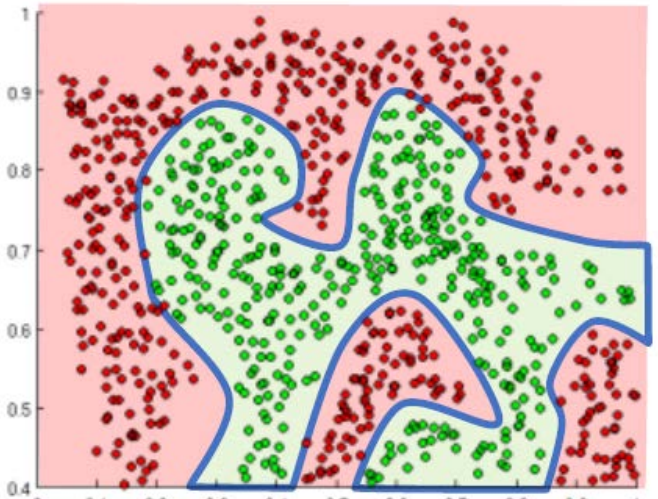
Maxout

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

ELU

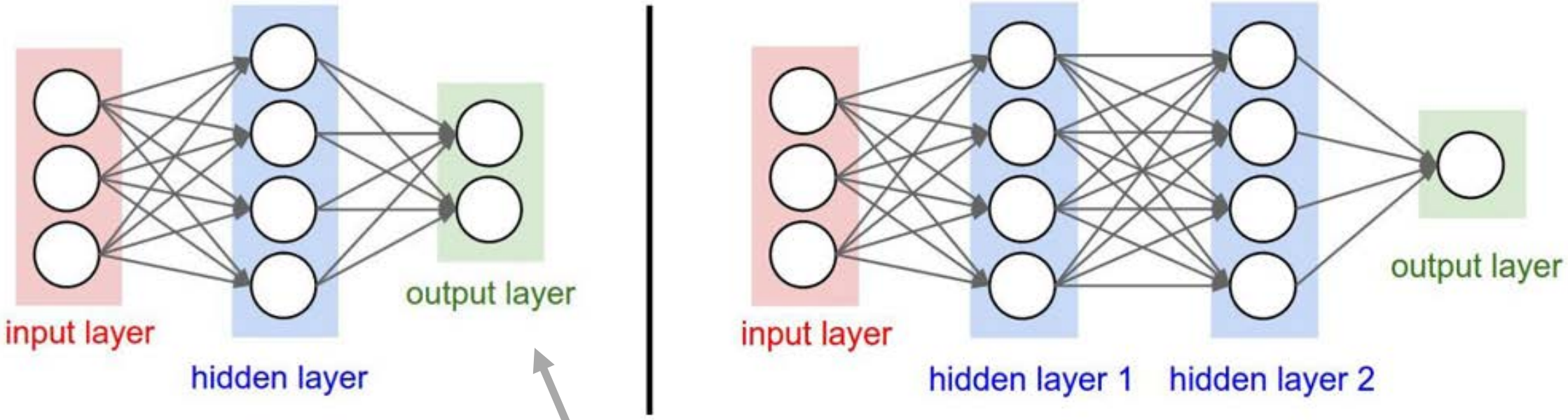
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$





Artificial Neural Network (ANN)

- also called Multi-Layer Perception (MLP) , or
- Fully Connected (FC) layers in CNN models



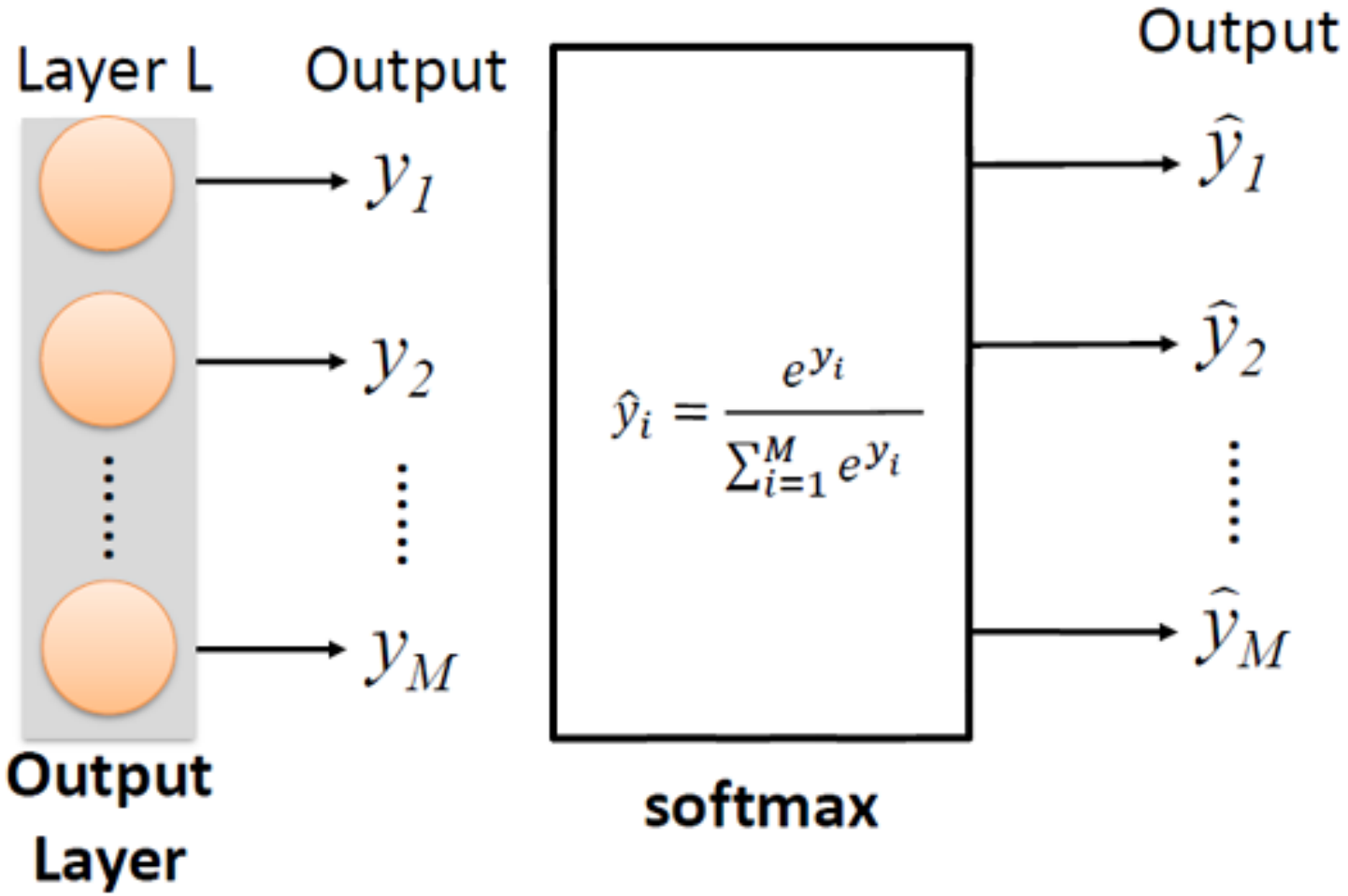
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

Softmax

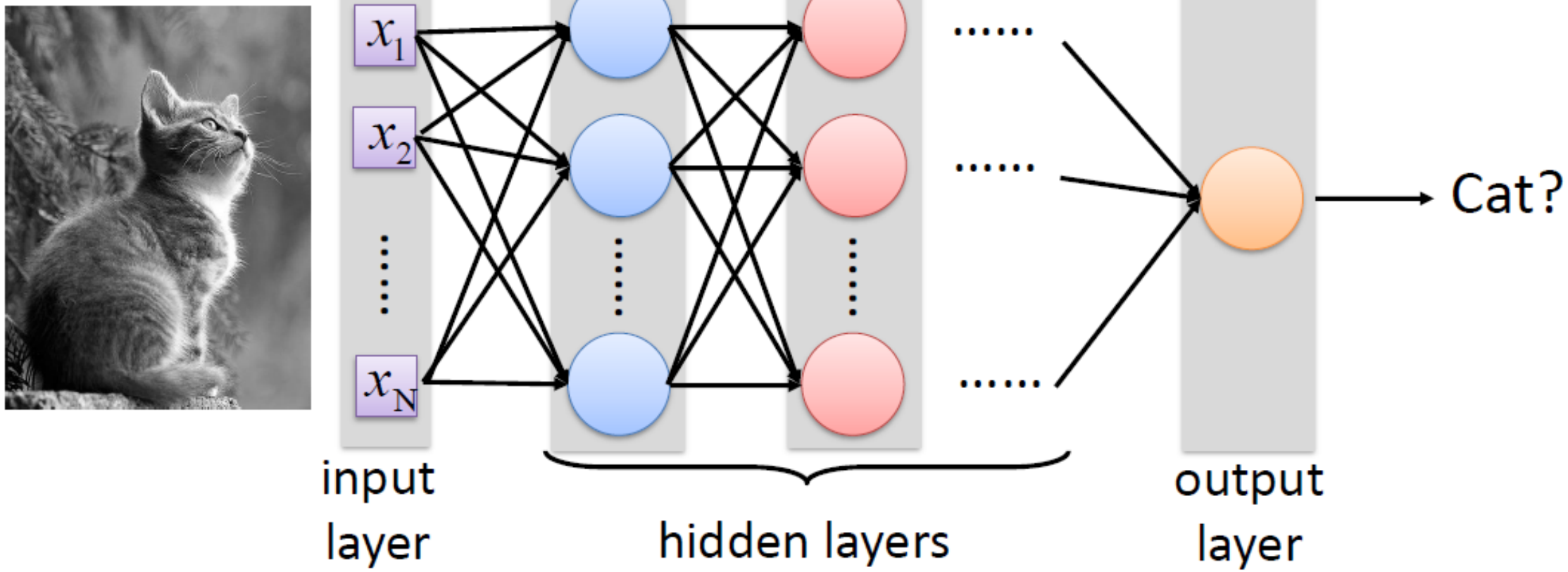
- append at the output layer to generate probability in multi-class classification



- $\hat{y}_i$ represents probability of the i-th class

Image Classification with ANN

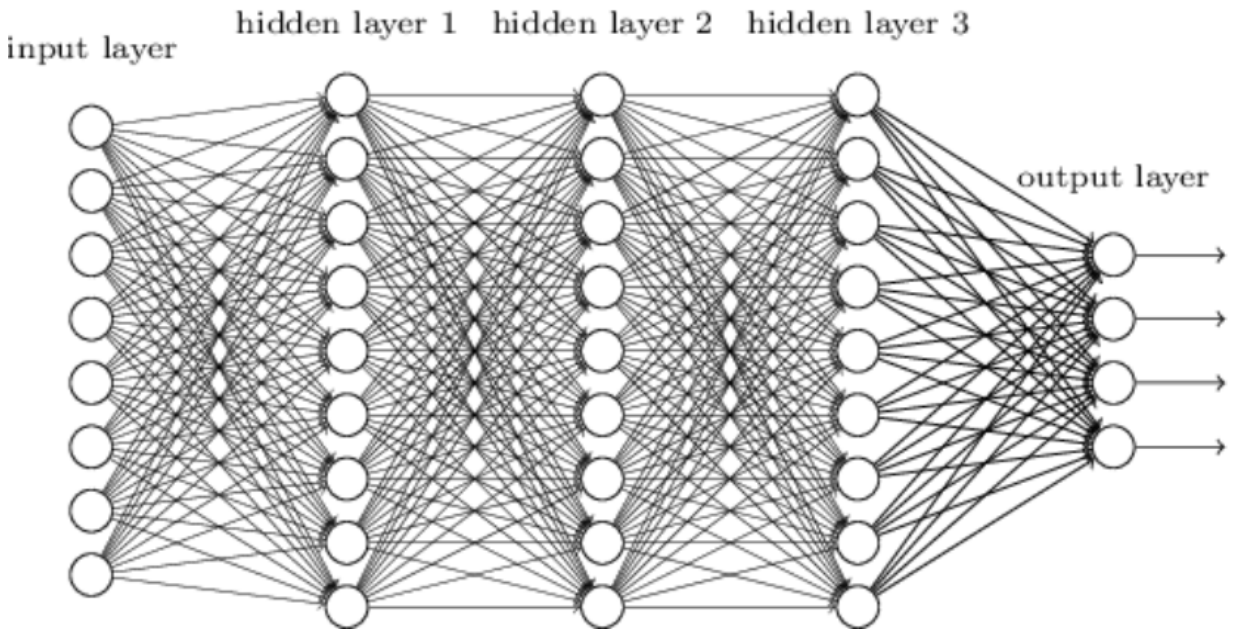
- deep means many hidden layers in ANN



Fully-Connected ANN

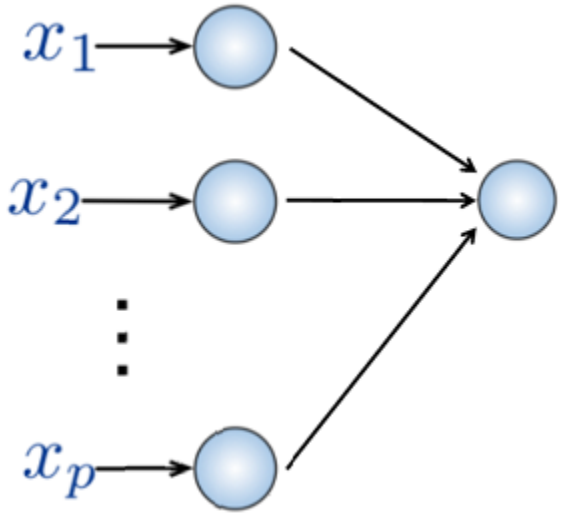
- Full connections between input and output neurons
- no spatial information among neighboring neurons
- many parameters

How can we use **spatial structure** in the input to inform the architecture of the network?



Input :

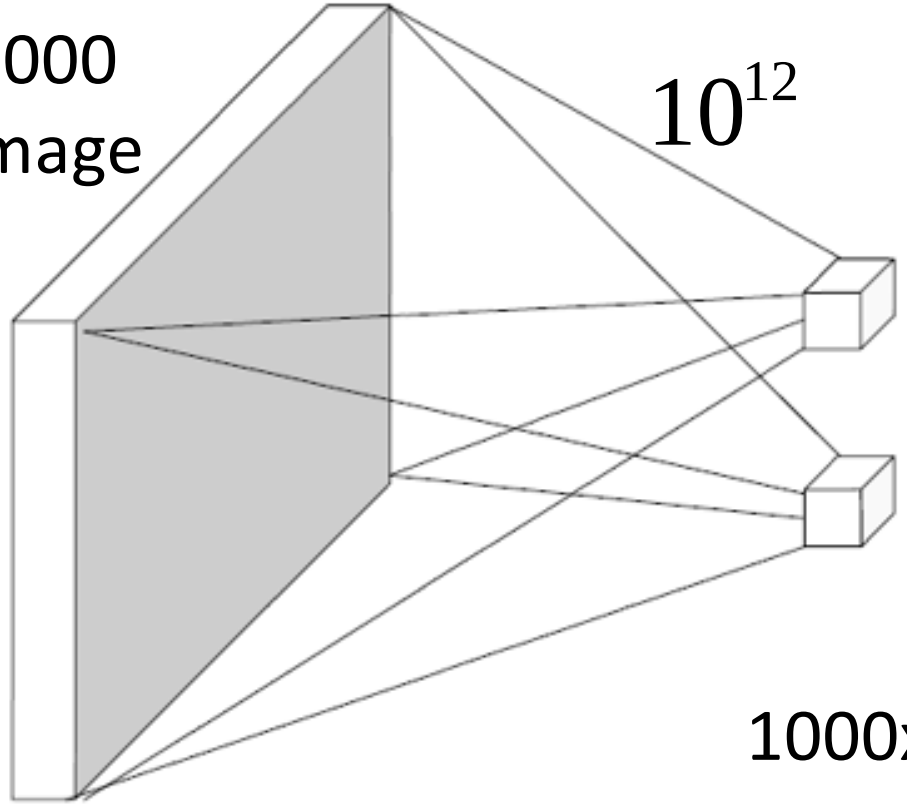
- 2D image
- Vector of pixel values



Problems with ANN

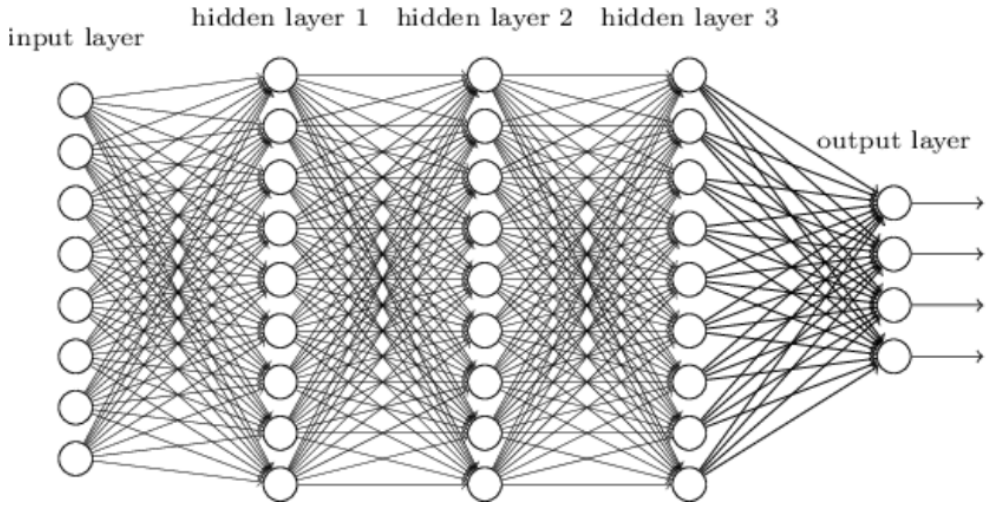
- Too many links when applying fully connected layers to images

1000x1000
input image



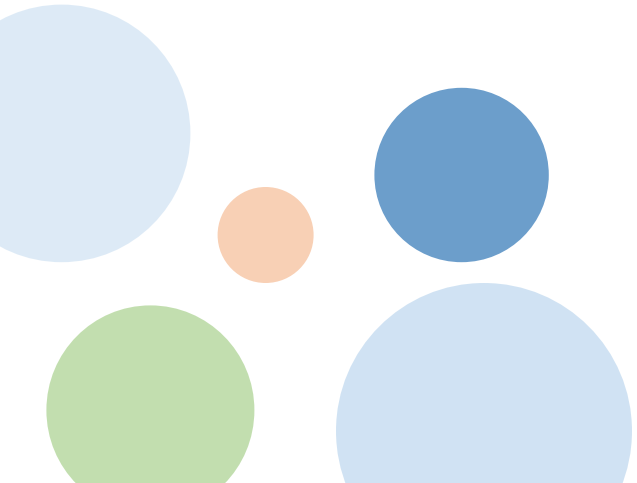
Fully connected layer

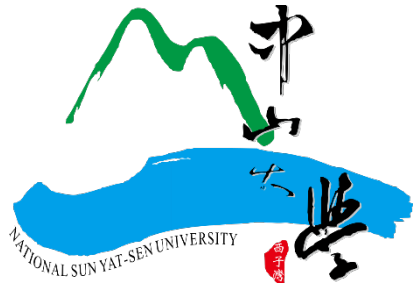
1000x1000
output image



For a 1000x 1000 image, there are 1M links for a hidden node.
If there are 1M hidden nodes, there are 10^{12} parameters

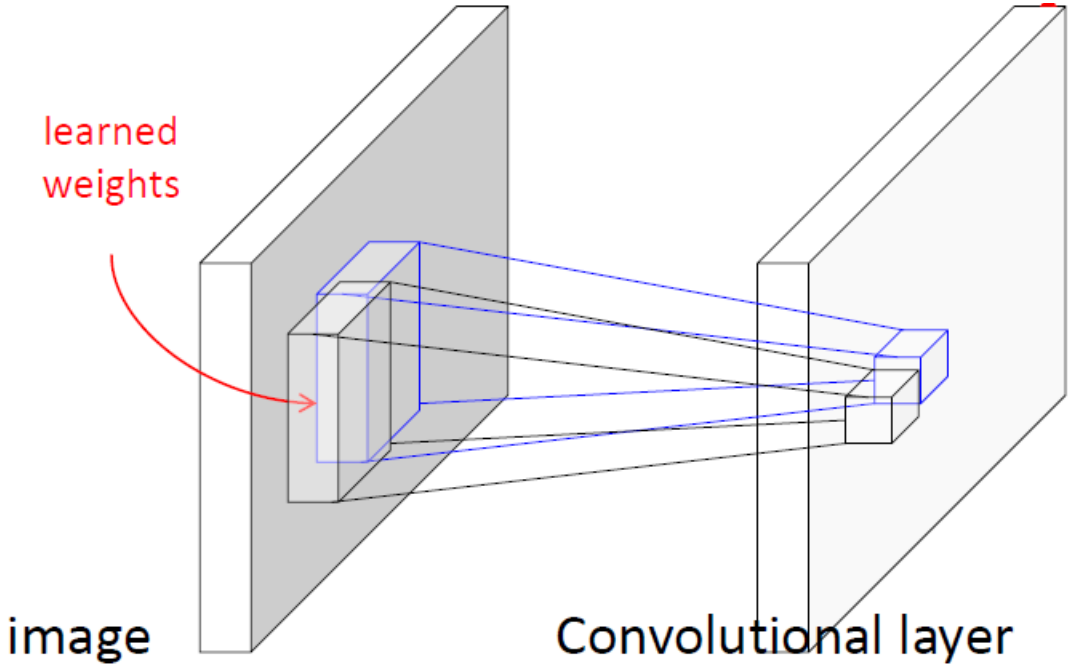
CNN

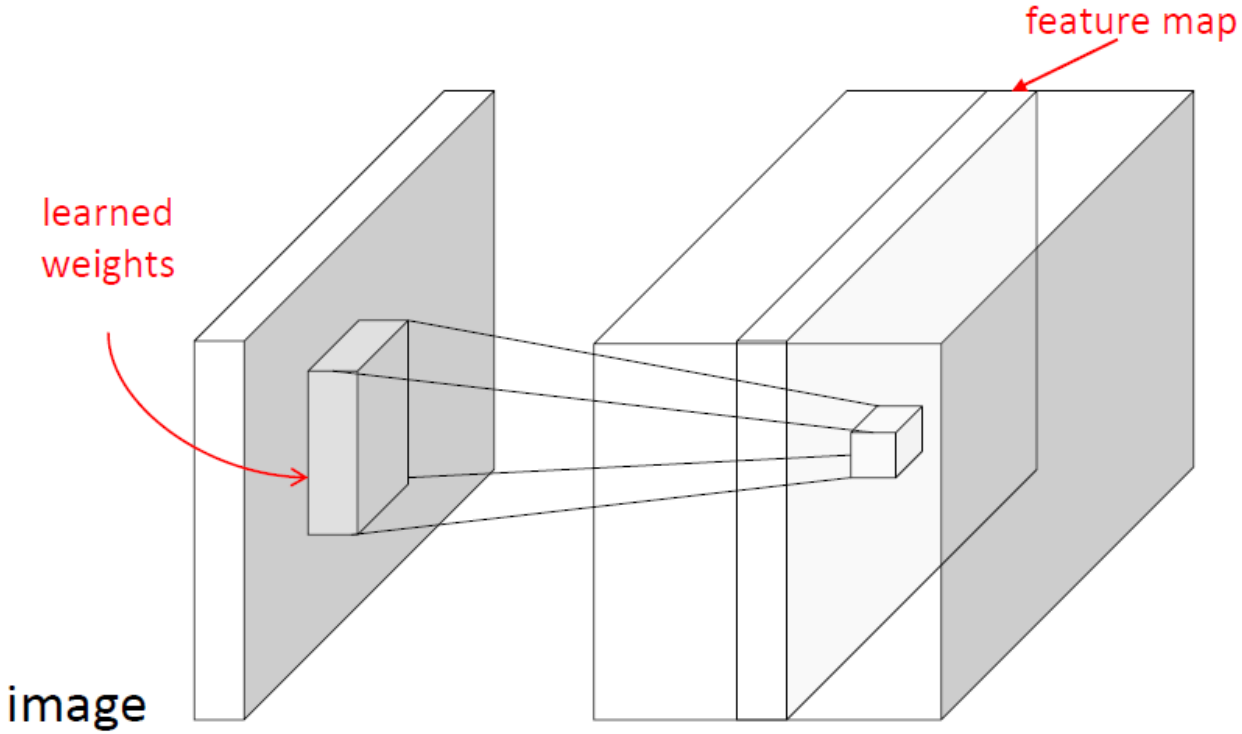




Convolutional Neural Networks (CNN)

- local connectivity due to spatial dependencies in images
- shared filter across the same image
- M input feature maps (input channels) -> N output feature maps (output channels)

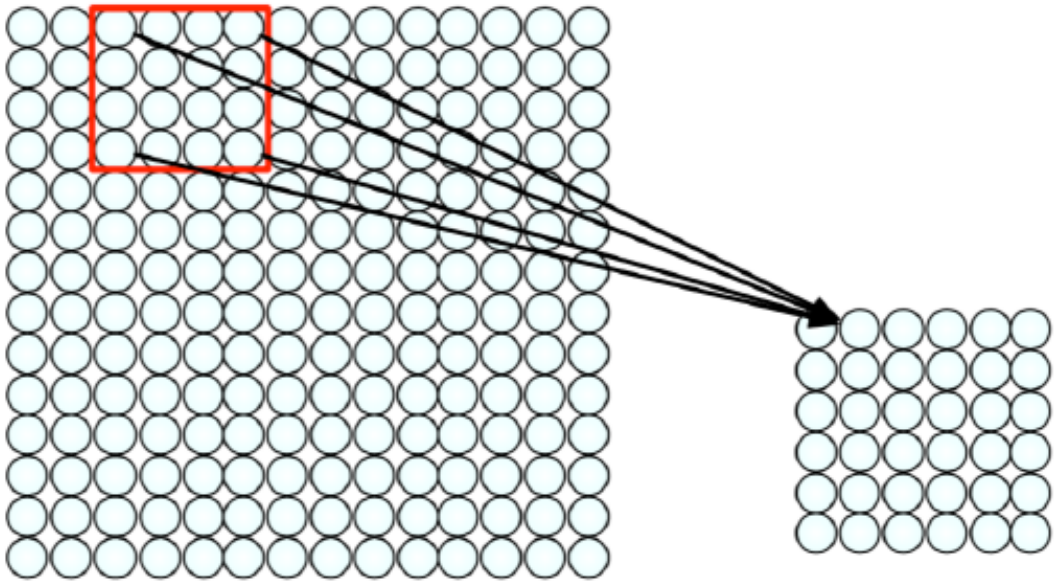




Convolution with Spatial Structure

Replace fully-connected ANN by convolution to reduce # of parameters and operations => Convolutional Neural Network (CNN)

- example:
  - Filter of size 4x4: 16 different weights
  - Apply this same filter to 4x4 patches in input
  - Shift by 2 pixels for next patch (stride=2)
  - This "patchy" operation is called 2D **convolution**



I) Apply a set of weights — a filter— to extract **local features** for an input feature map

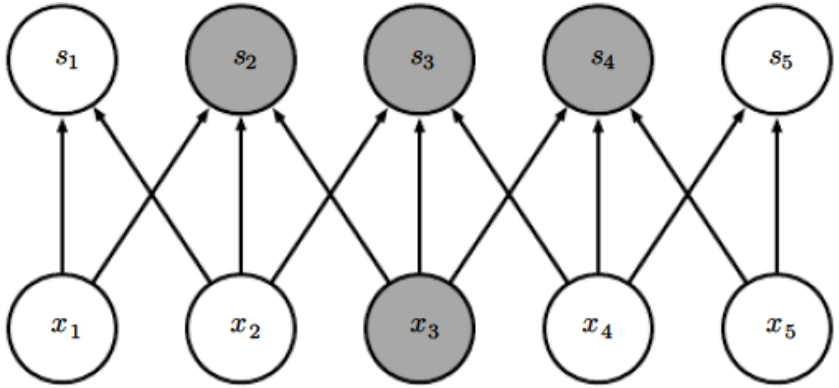
2) Use **multiple filters** to extract different features (generate **different output feature maps**)

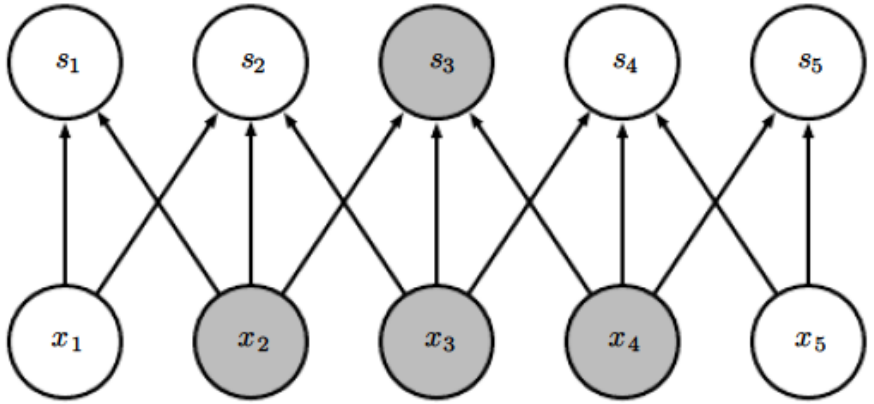
3) **Spatially share** parameters of each filter across the same input feature map

Connection: Local vs. Full

- output units affected by x3 input units affecting s3

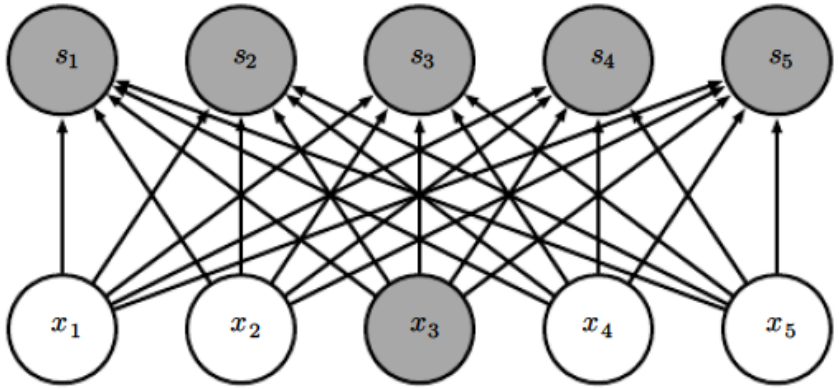
Local Convolution

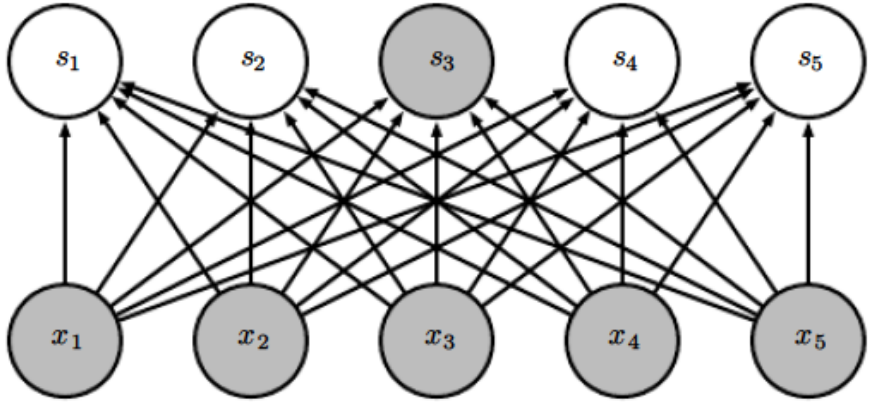




receptive field (3 local features)

Fully Connected

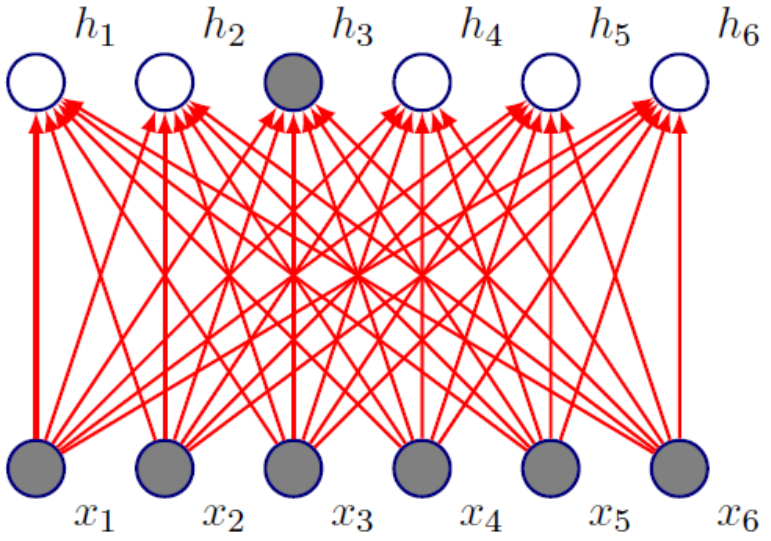


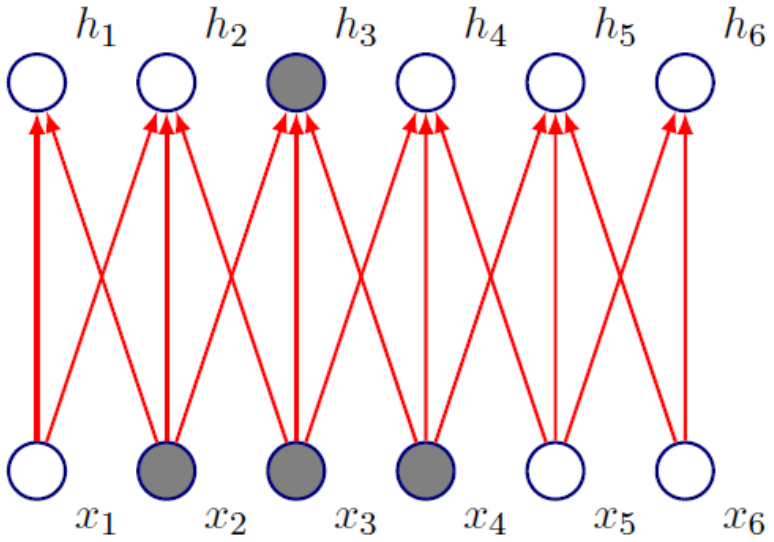


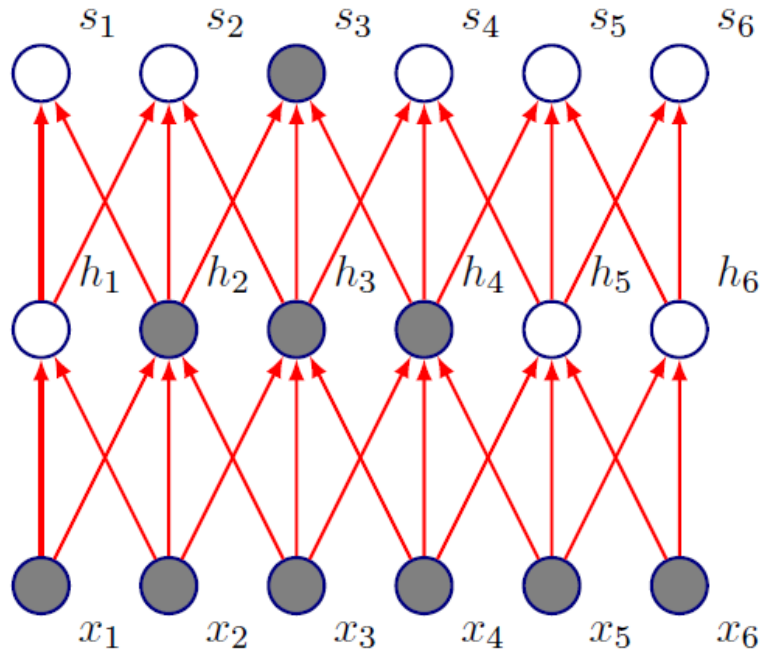
receptive field (all features)

Motivation for Convolution

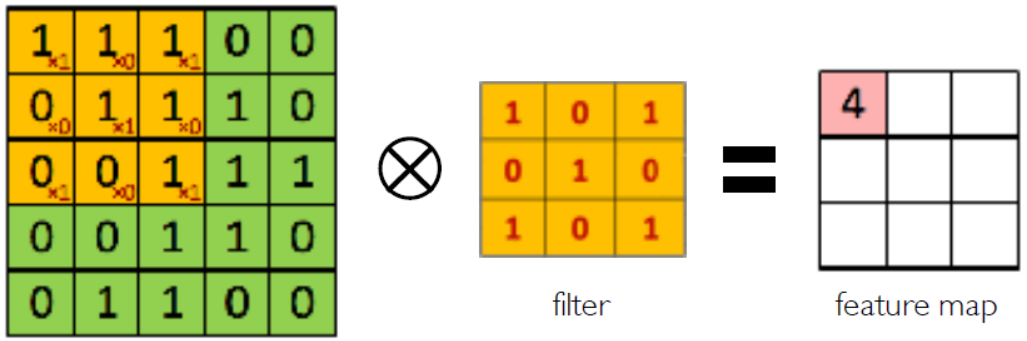
- sparse interaction
- parameter sharing
- enlarge receptive field via more layers

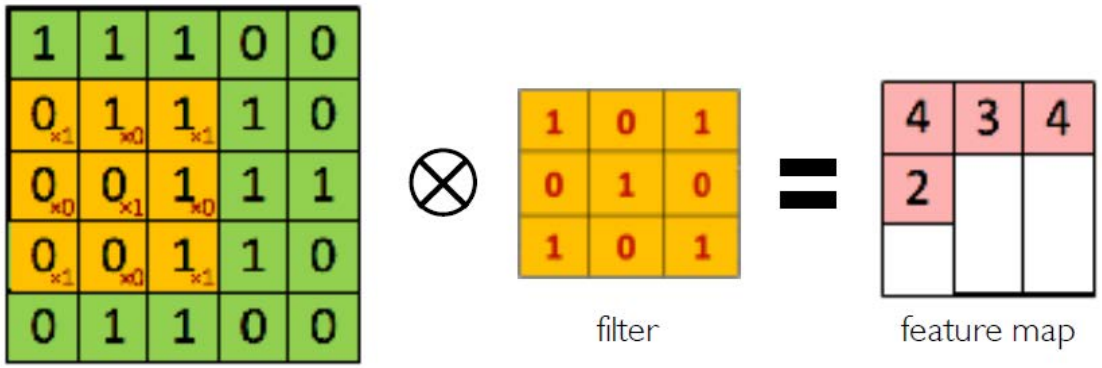


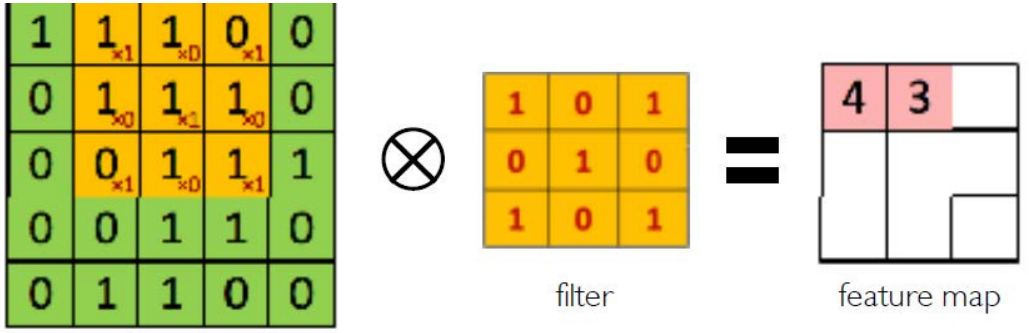


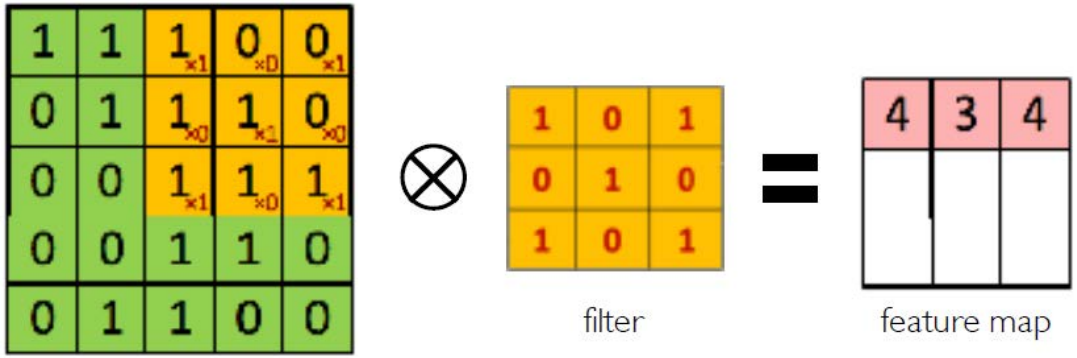


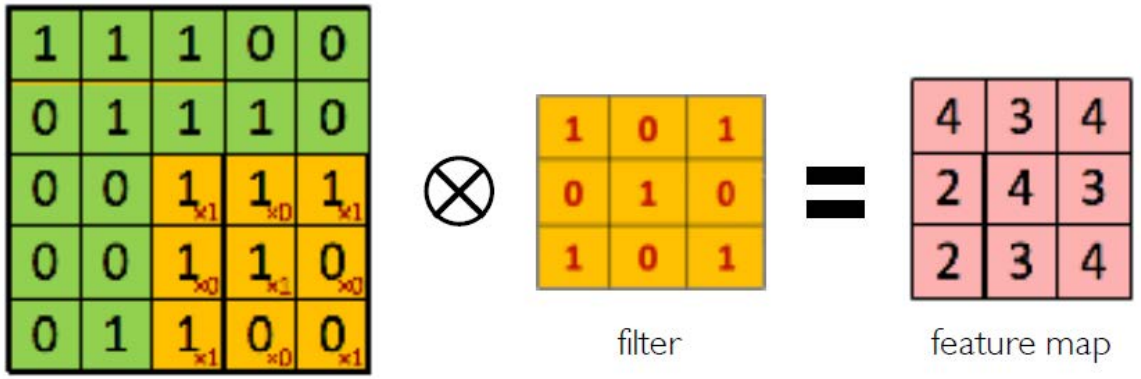
Convolution example (stride=1)



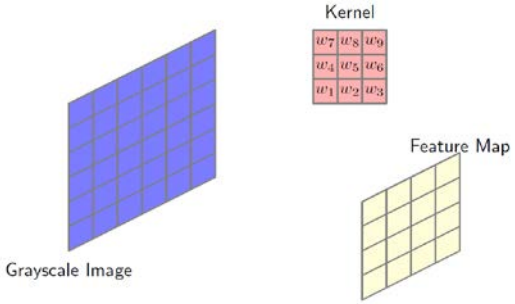




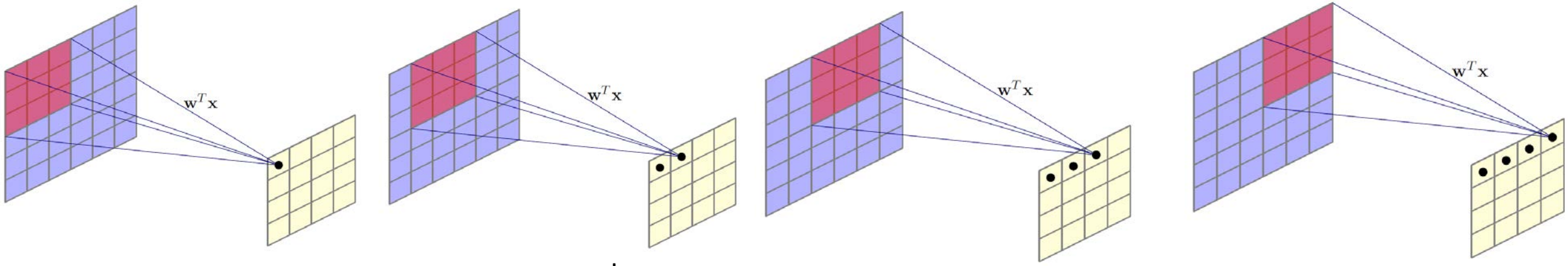




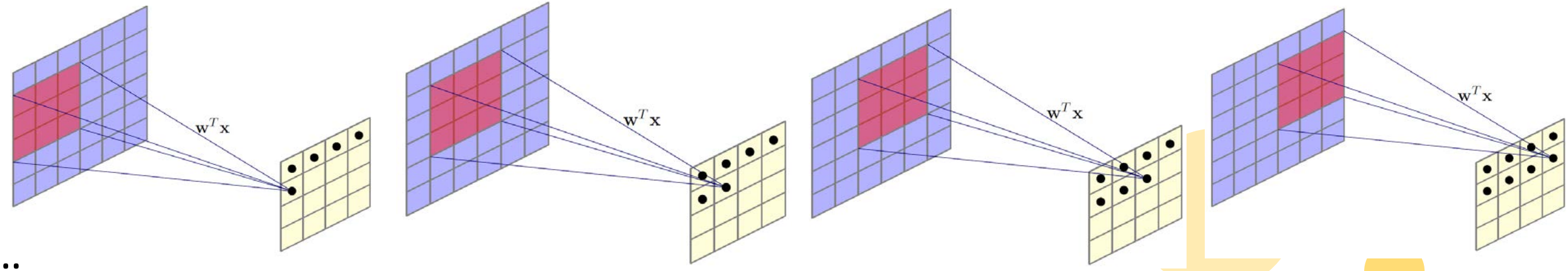
Convolution (in steps)



- convolution to generate the 1st row of an output feature map



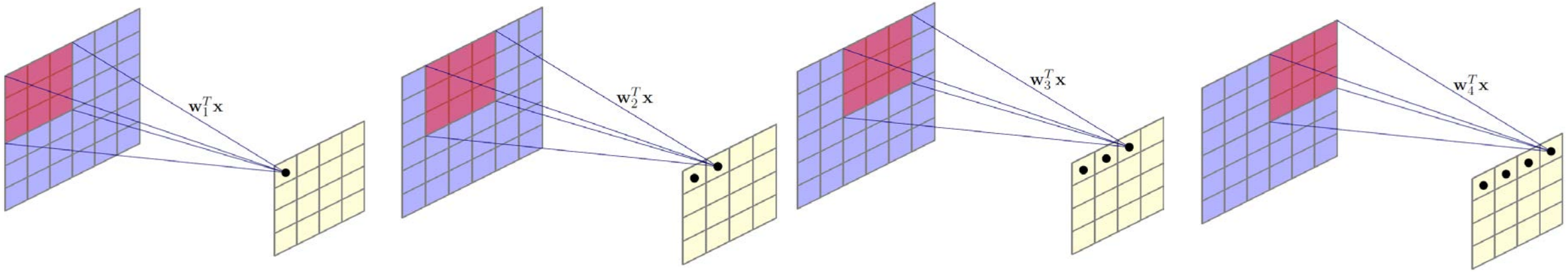
- convolution to generate the 2nd row of an output feature map



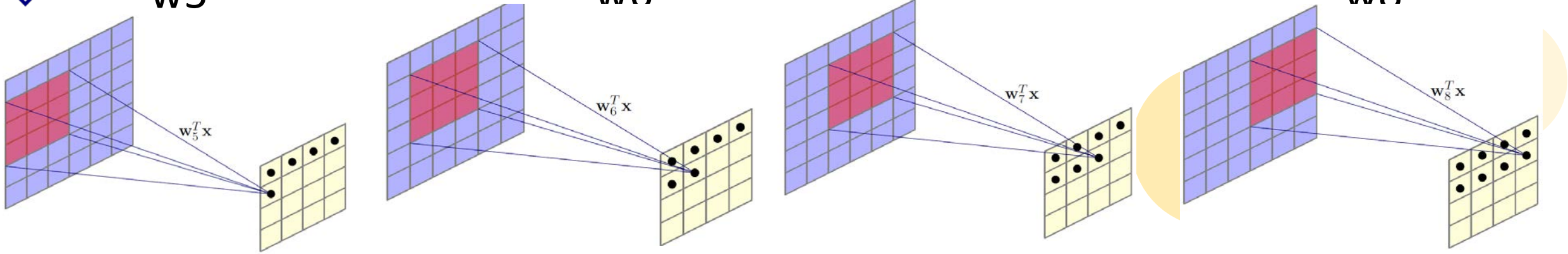
...

Locally Connected vs. Convolution

- convolution: **shared** filter weights for different output pixels
- locally connected: **different** filter weights for different output pixels
- w1 w2 w3 w4

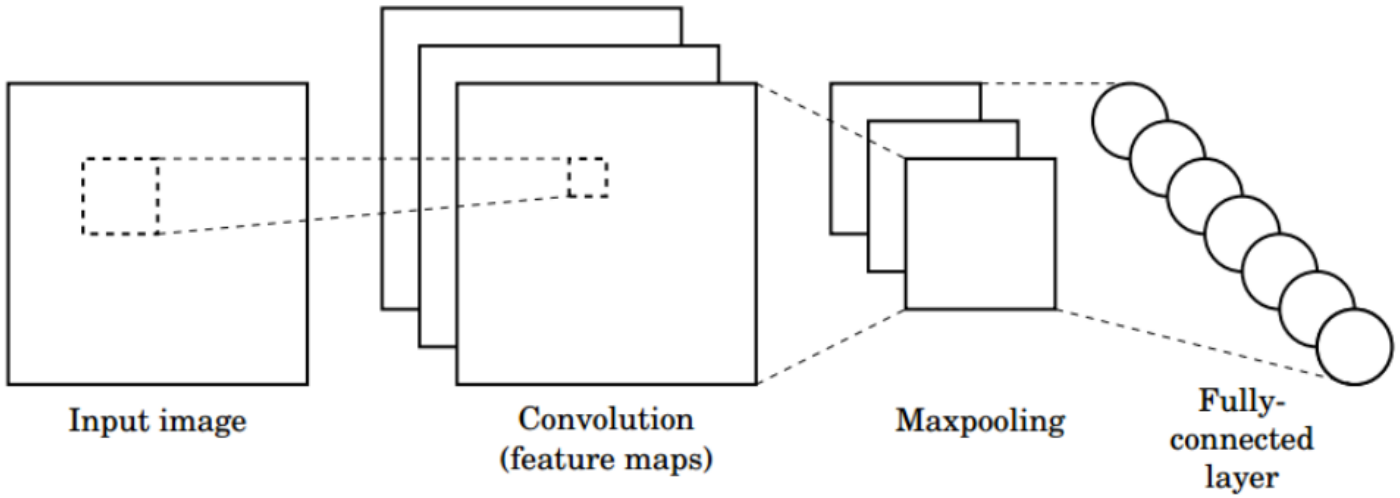


- w5 w6 w7 w8



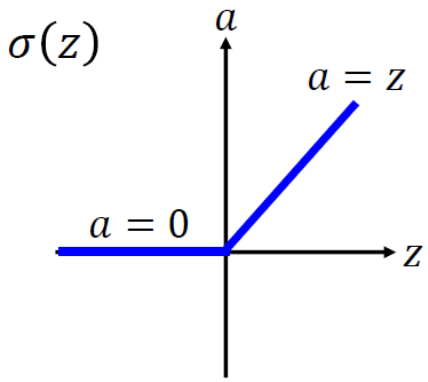
CNN for Classification

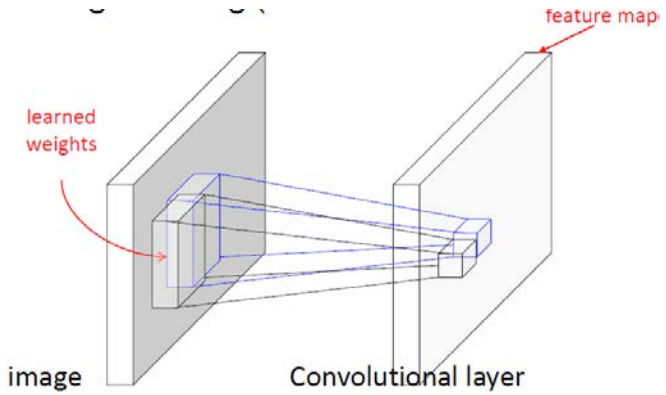
- CNN = Convolutional (Conv) layers + Fully Connected (FC) layers
- convolution
  - exploits **spatial structure** with **shared parameters** (filter weights)
  - used for feature extraction
- Pooling
  - exploit **local invariance**
  - used to reduce image size, and thus computation complexity
- non-linear activation function (i.e. ReLU) after each layer

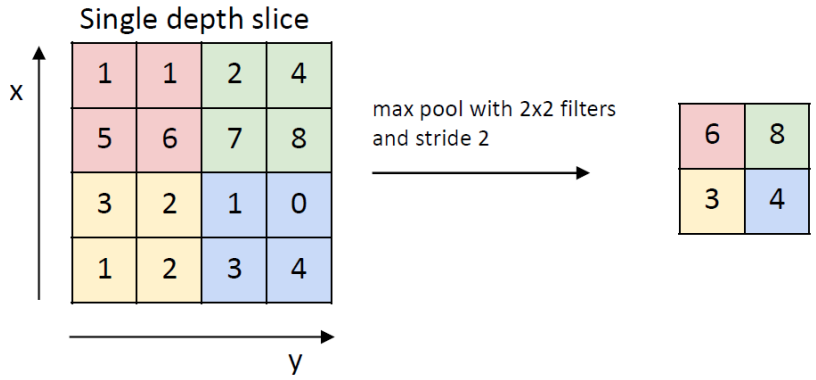


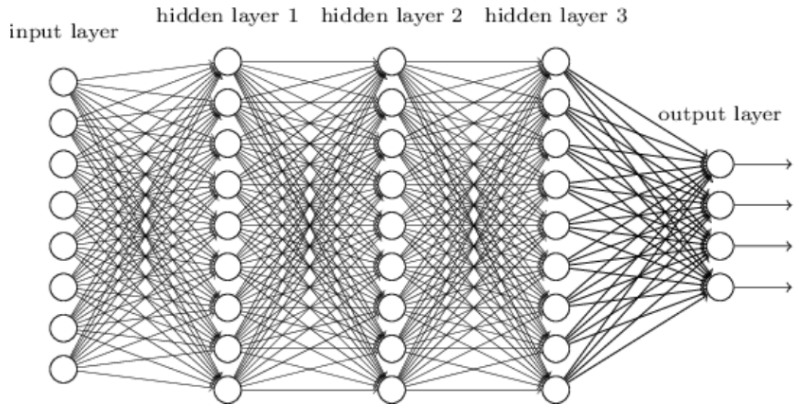
Building Blocks of CNN

- Convolution (Conv) layers
  - 3D Convolutions
- Activation
  - ReLU
- Pooling layers
  - size reduction of feature maps
- Fully Connected (FC) layers
  - classification



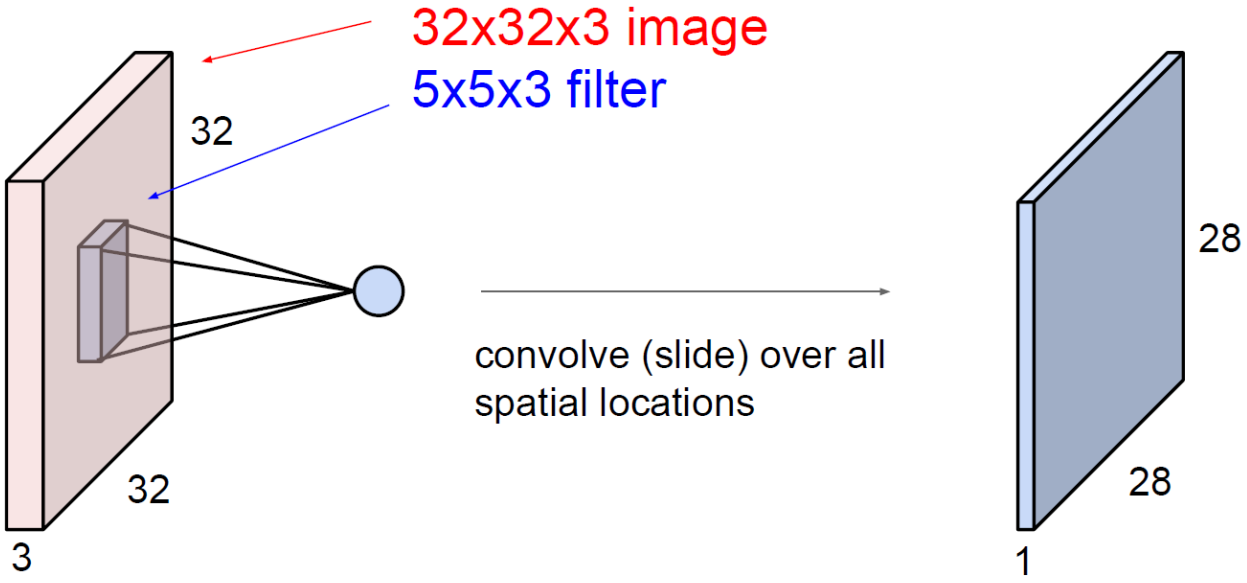






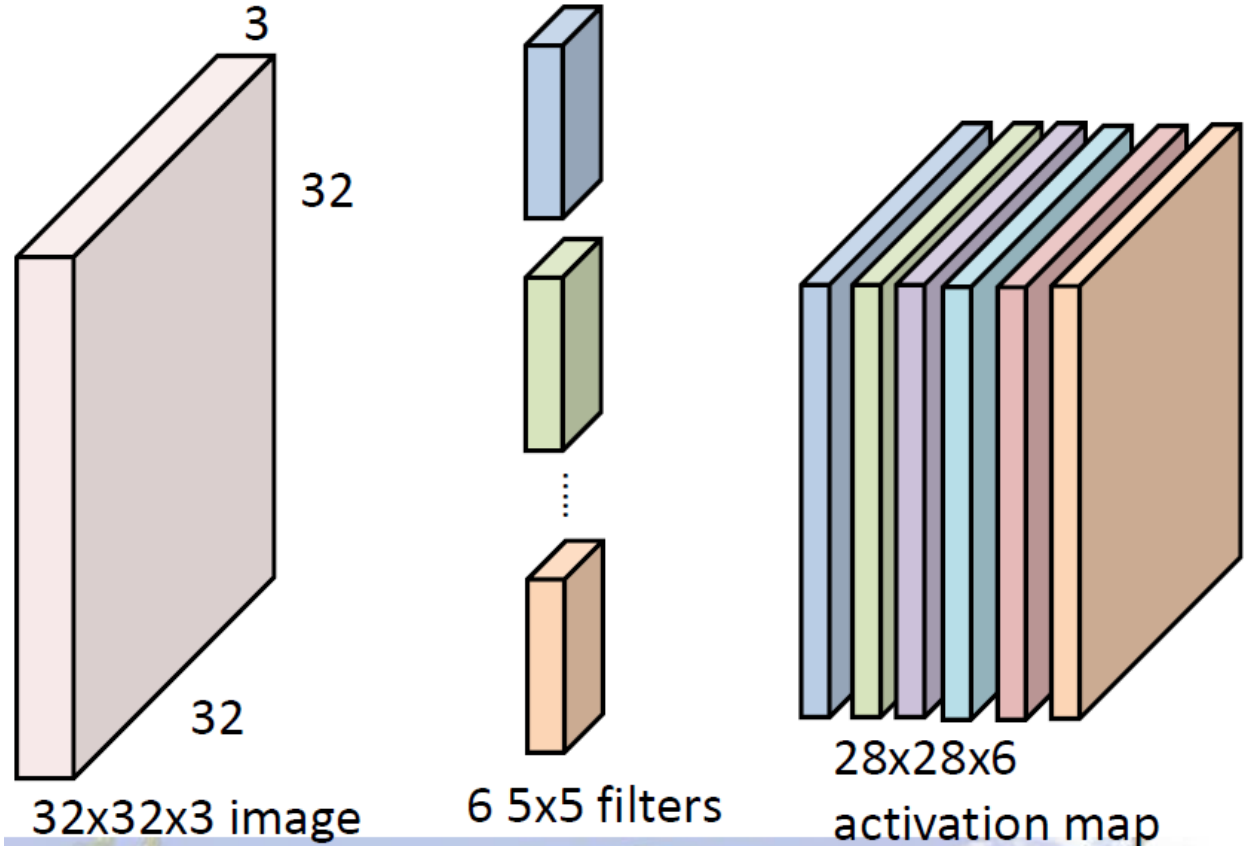
Convolutional Layer (1 output channel)

- each 2D filter is used for 1 input channel (input feature maps)
  - e.g., 5x5 filter
- 3D convolution = sum of 2D convolutions for all input channels
  - e.g. convolution with 5x5x **3** filters for **3** input channels
- each output pixel is connected to a small region (called **receptive field**) of input pixels
  - e.g. 5x5 receptive field for each output pixel
- output channel size is slightly reduced if no padding is used
  - (e.g., from 32x32 to 28x28 via 5x5 filtering)



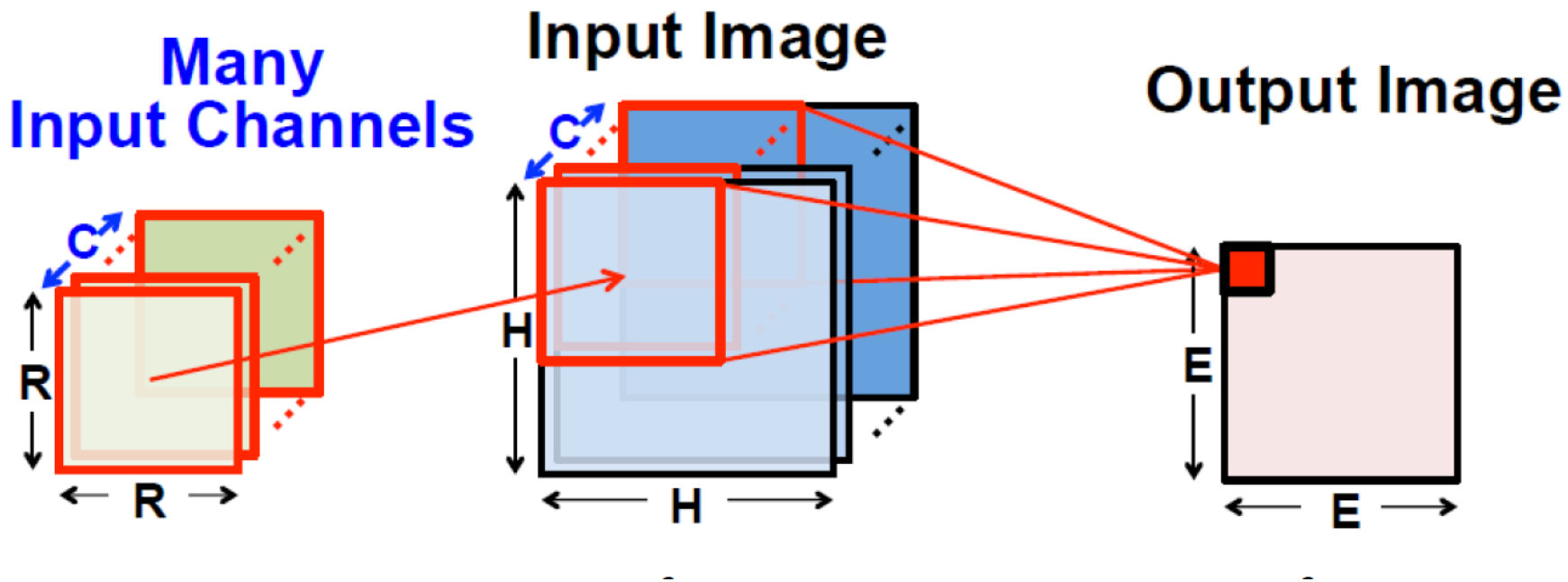
Convolutional Layer (6 output channels)

- each filter is used for 1 pair of input and output
- e.g., **6** 5x5x**3** filters for **3** input channels and **6** output channels
- a Conv layer generates output pixels (neurons) in 3D grid



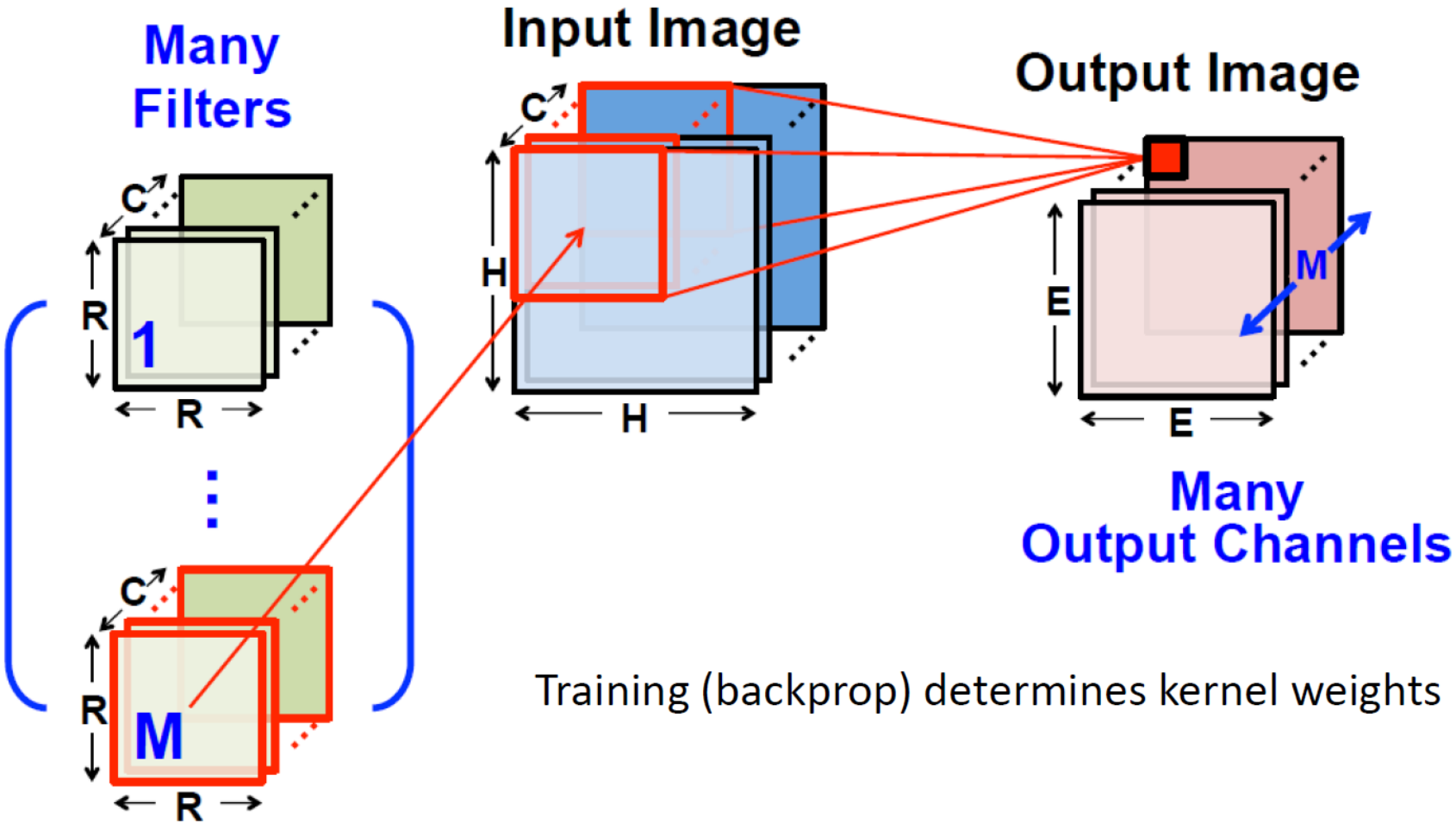
3D Convolution with a 2D output feature map

- a receptive field in input feature maps gives an output neuron
- each output feature map has its own set of kernel weights
- e.g., C input feature maps (input channels) of size HxH, C filter kernels of size RxR, one output feature map (output channel) of size ExE



3D Convolution with 3D Output

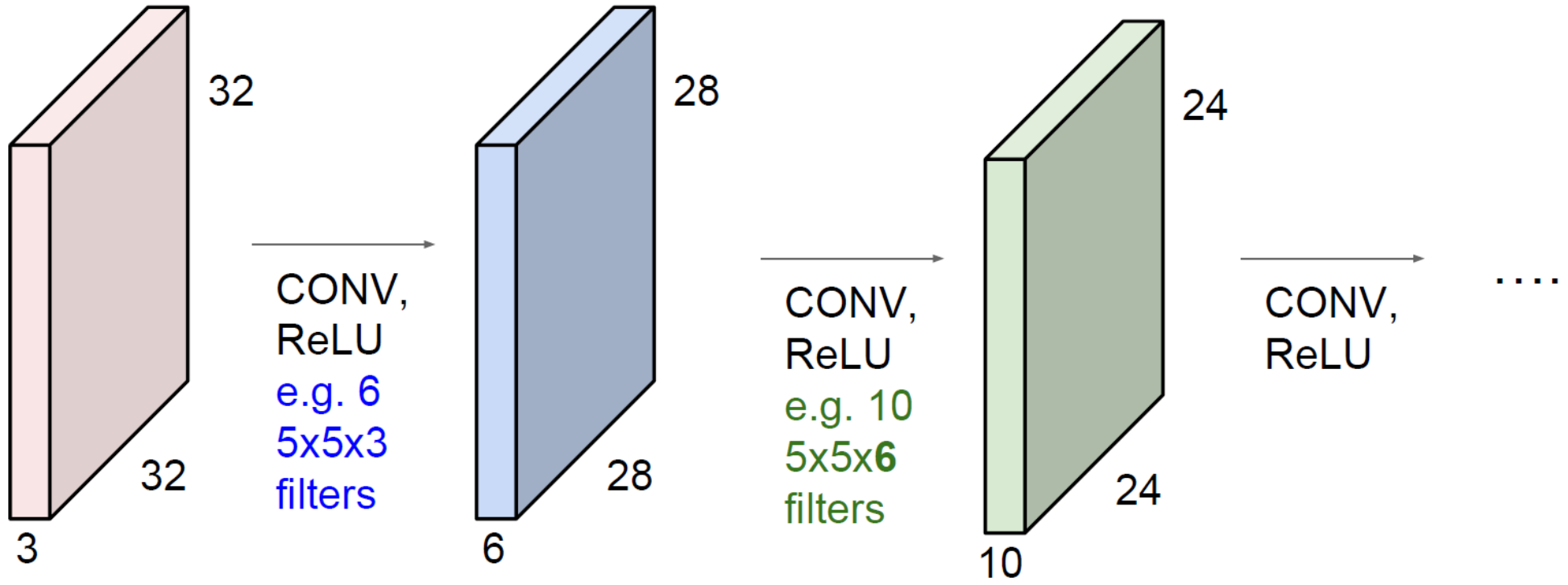
- C input features => M output feature maps
- need CxM filter kernels, C filter kernels for each output channel



Training (backprop) determines kernel weights

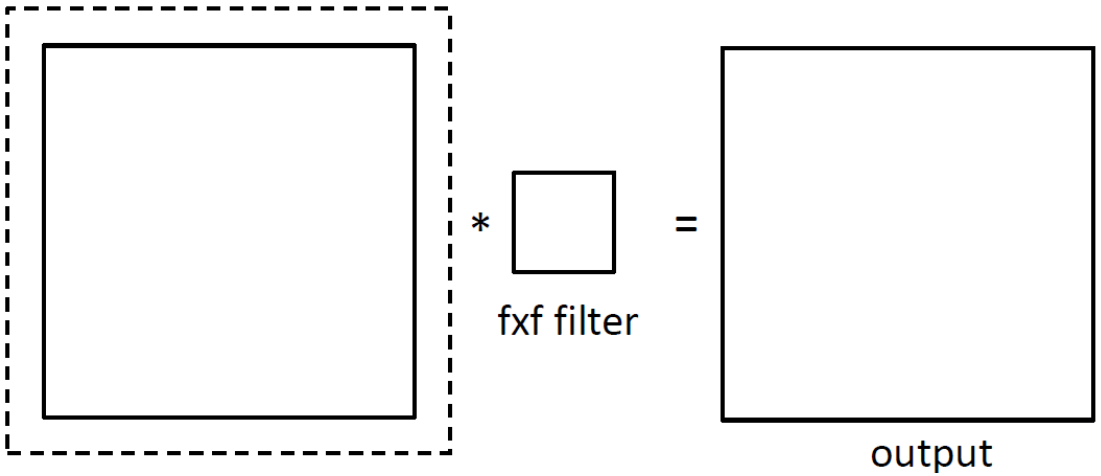
Conv+ ReLU

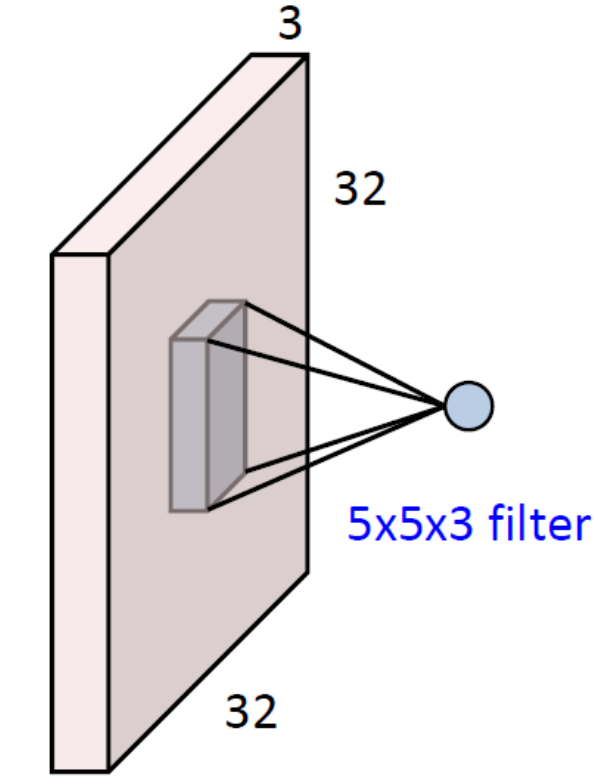
- output neurons of Conv layer are connected to non-linear activation function (usually ReLU in CNN)
- What is the receptive field in the two Conv layers below? (Ans.: 9x9)

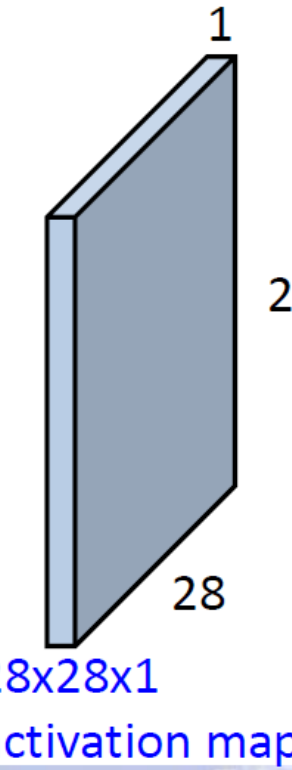


Padding

- input feature map size: 32x32
- Convolution with 5x5 filter and stride=1
- output feature map size: 28x28
- what is the size of output feature map with fxf filter?
- how to keep output feature map size fixed?
  - padding of input feature map with two borders
- what size of padding is required for fxf filter?





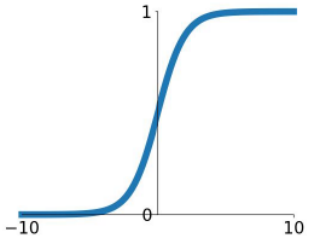


Activation Functions

- without activation functions, the output is just linear combination of inputs!
- activation functions create non-linearity

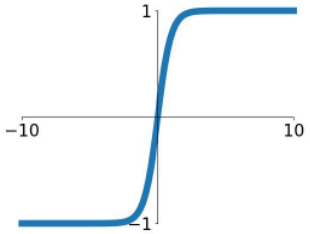
Sigmoid

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



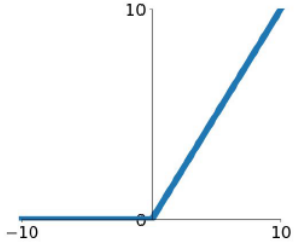
tanh

$$\tanh(x)$$



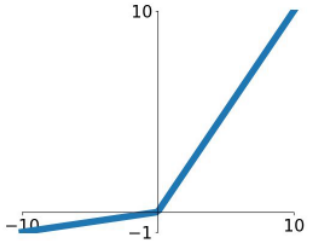
ReLU

$$\max(0, x)$$



Leaky ReLU

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

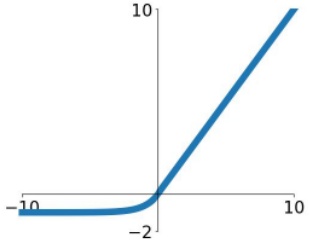


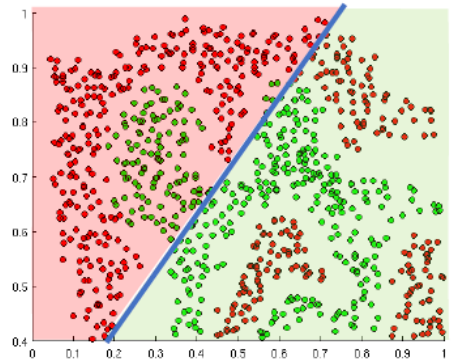
Maxout

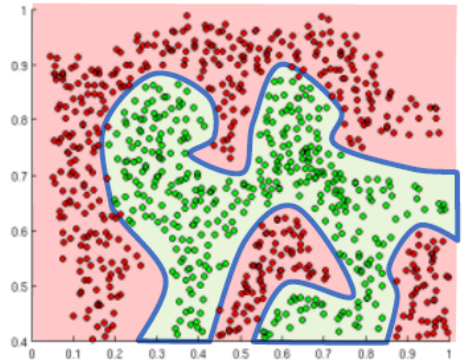
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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



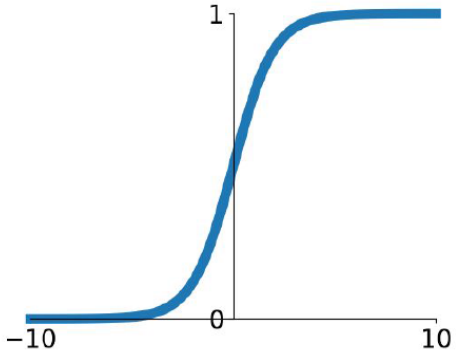




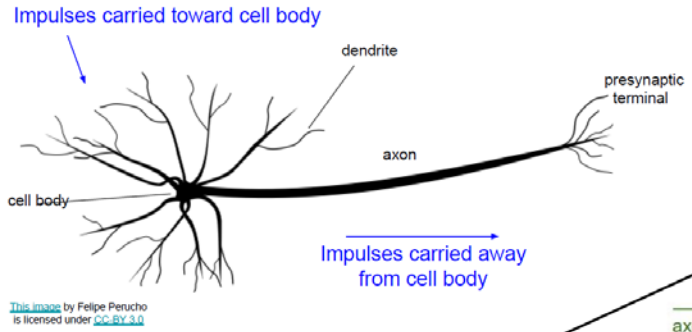
sigmoid

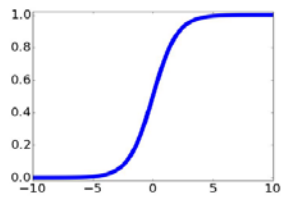
$$\sigma(x) = 1/(1+e^{-x})$$

- output range in [0,1]
  - not zero-centered
- historically popular
  - nice interpretation as a saturating “firing rate” of a neuron
- problems
  - saturated neurons “kill” gradients
  - output are not zero-centered
  - exponential function is compute-expensive



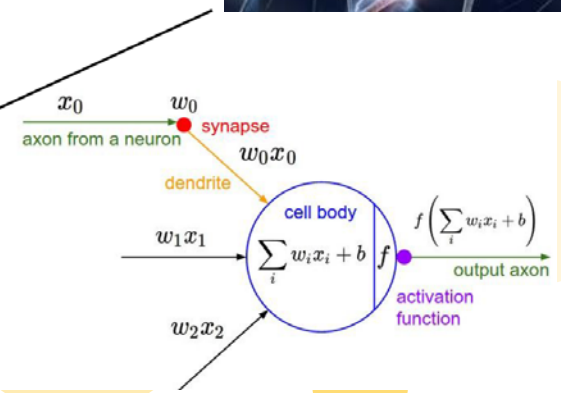
Sigmoid





sigmoid activation function

$$\frac{1}{1+e^{-x}}$$

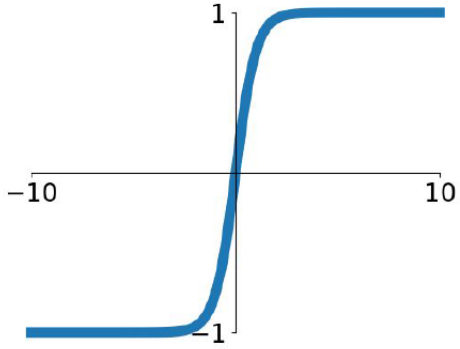


tanh(x)

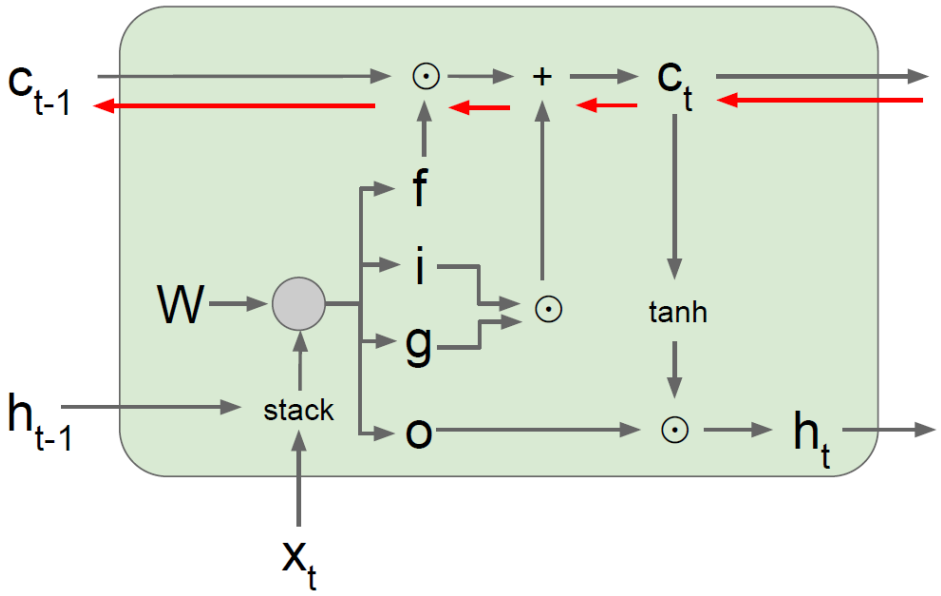
$$\tanh(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$$

$$= 1 - 2e^{-2x} / (1 + e^{-2x})$$

- output range in [-1, 1]
  - zero-centered
- still kills gradients during saturation regions
- used in LSTM (Long Short Term Memory)



tanh(x)



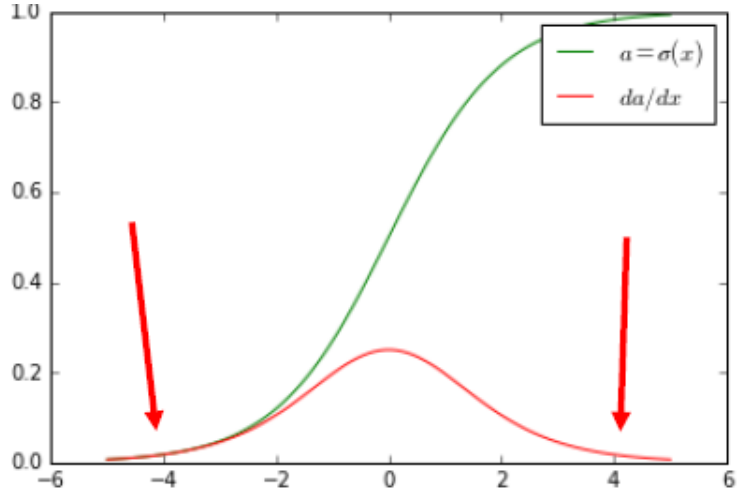
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

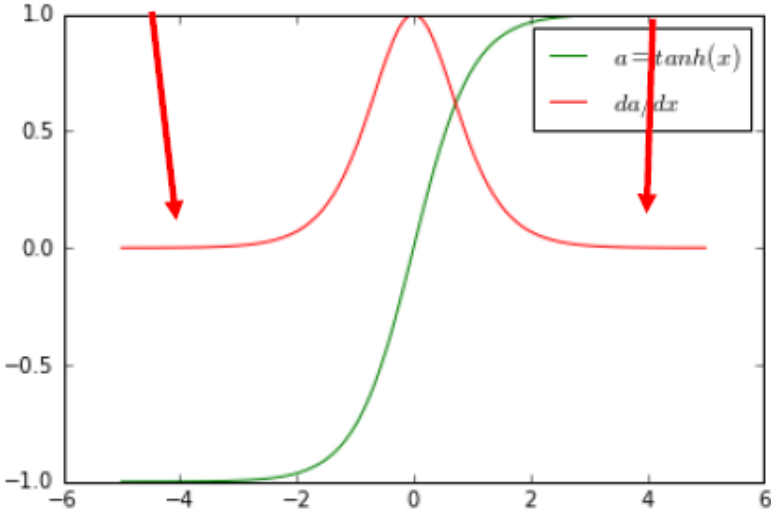
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

tanh vs. sigmoid

- Functional form is very similar: $tanh(x) = 2\sigma(2x) - 1$
- $tanh(x)$ has better output $[-1, +1]$ range
  - Stronger gradients, because data is centered around 0 (not 0.5)
  - Less "positive" bias to hidden layer neurons as now outputs can be both positive and negative (more likely to have zero mean in the end)
- Both saturate at the extreme → 0 gradients
  - "Overconfident", without necessarily being correct
  - Especially bad when in the middle layers: why should a neuron be overconfident, when it represents a latent variable
- The gradients are < 1, so in deep layers the chain rule returns very small total gradient
- From the two, $tanh(x)$ enables better learning
  - But still, not a great choice

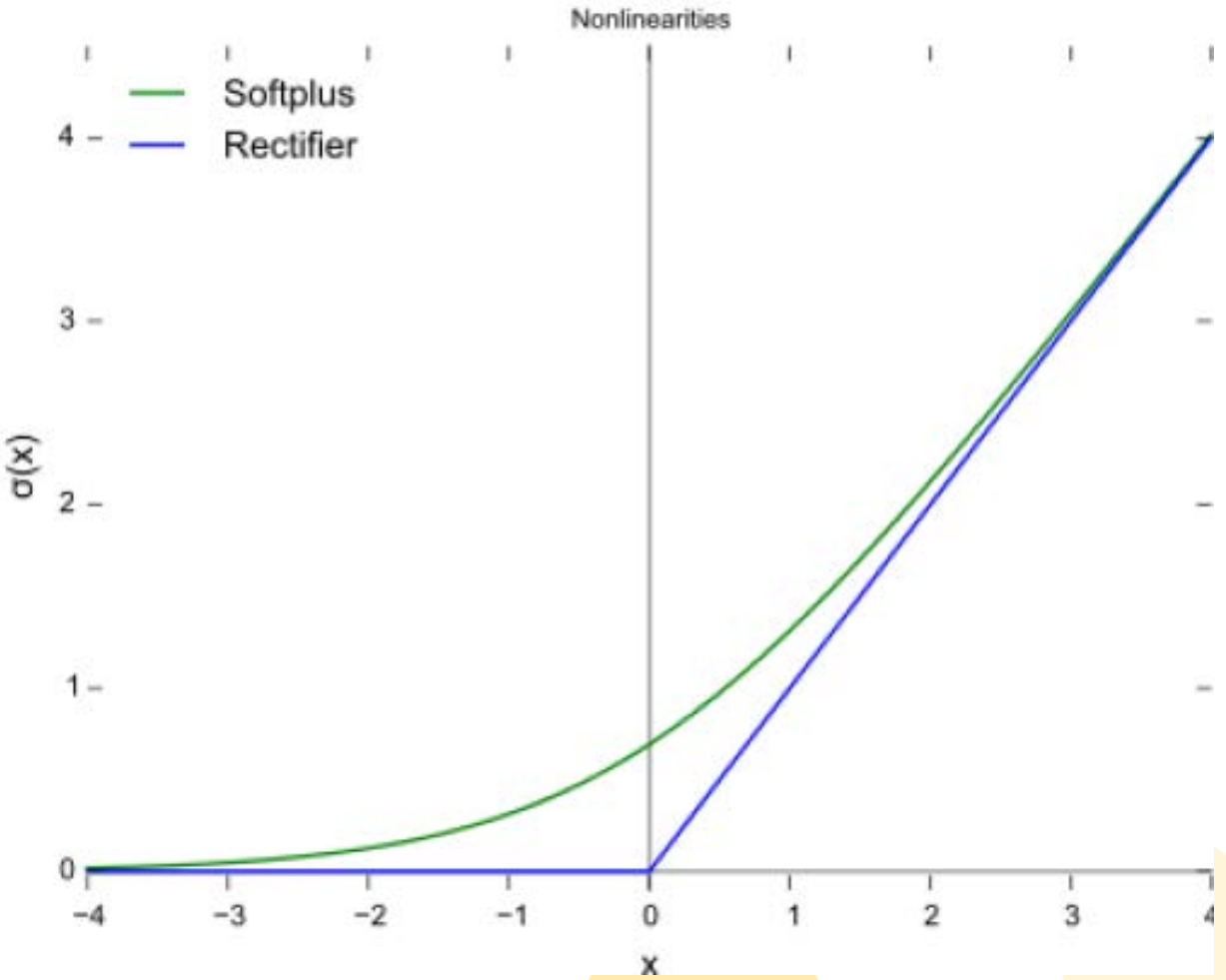




ReLU

$$\mathrm{ReLU}(x) = \max(0, x)$$

- does not saturate in positive x
- computation efficient
- converge faster (6x) than sigmoid/than in CNN
- actually more biologically plausible than sigmoid
- not zero-centered
- what is gradient in negative x?



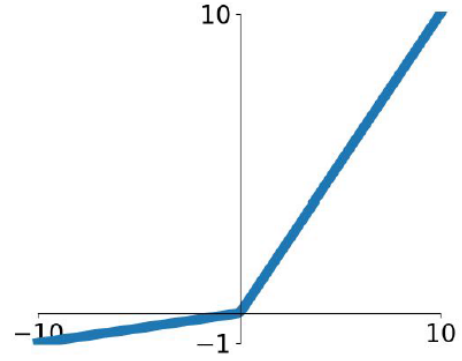
- softplus vs. ReLU

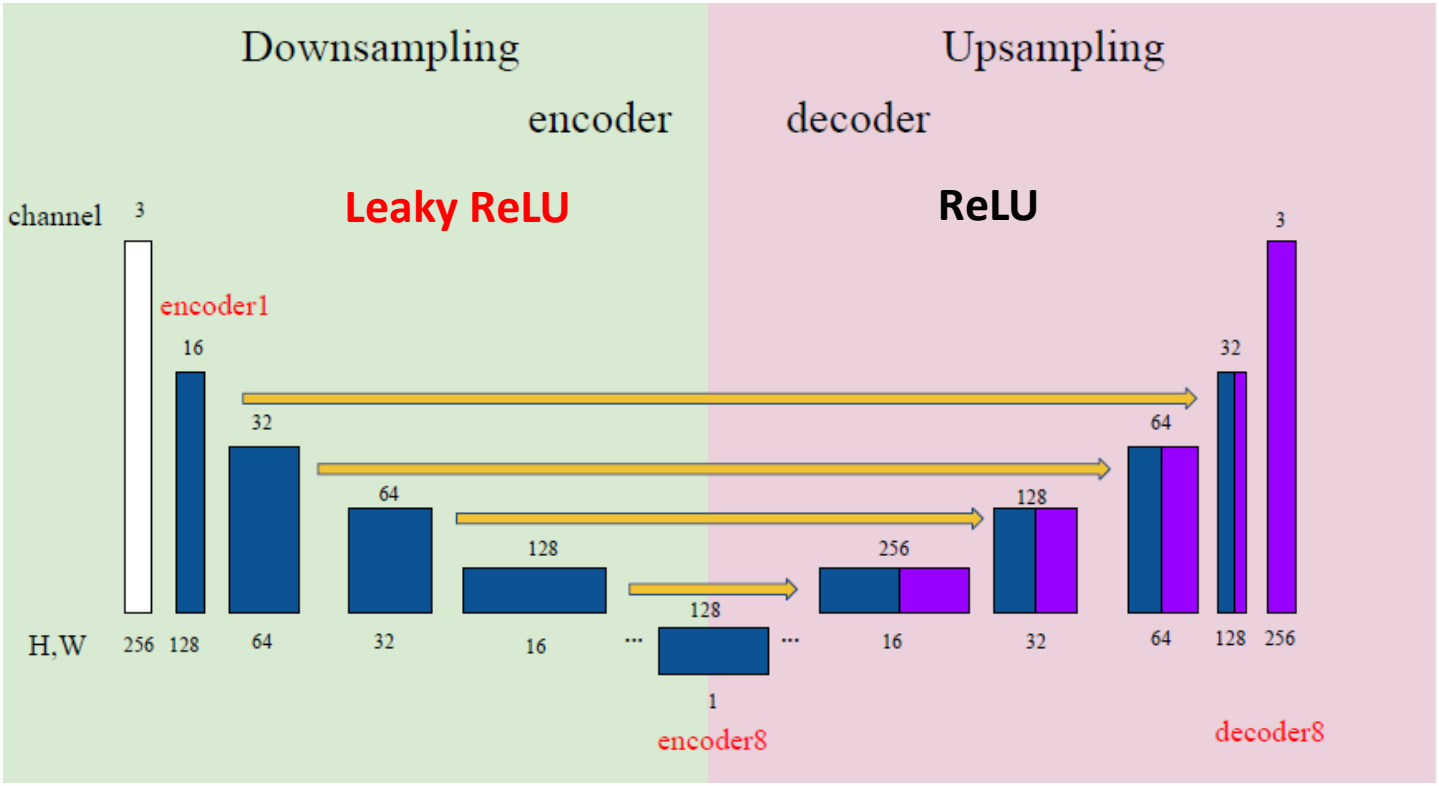
$$\mathrm{softplus} = \ln(1 + e^{x})$$

Leaky ReLU

$$\text{LeakyReLU}(x) = \max(\alpha x, x)$$

- does not saturate
- computation-efficient
- will not “die”
- also called Parametric ReLU (PReLU)
  - back-prop with gradient of alpha in negative x
- Leaky ReLU is used in encoder (Conv.) part of U-Net
  - decoder (DeConv) part still uses ReLU

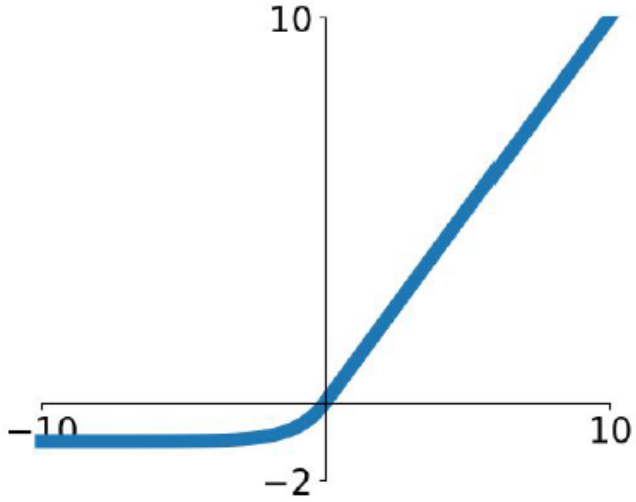




Exponential Linear Unit (ELU)

- all benefits of ReLU
- closed to zero mean outputs
- negative saturation region compared to leaky ReLU
- add some robustness to noise

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha \times (e^x - 1) & \text{if } x \leq 0 \end{cases}$$



Maxout

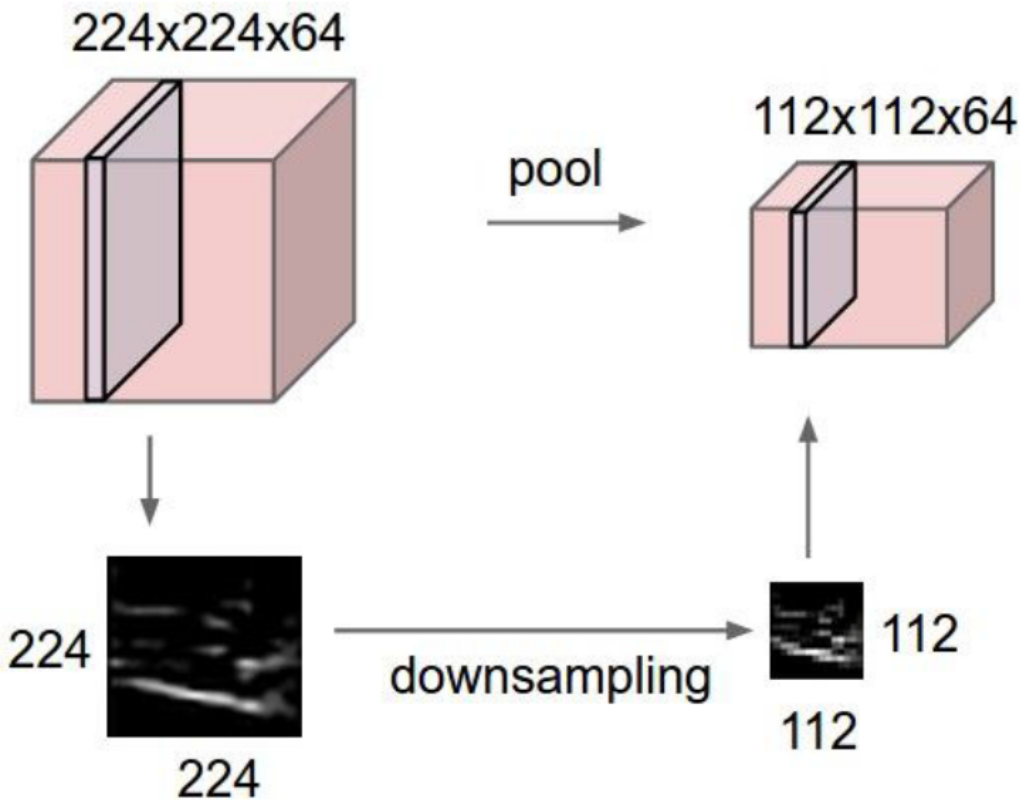
$$\text{maxout}(x) = \max(w_1^T x + b_1, w_2^T x + b_2)$$

- instead of applying an element-wise function, a maxout unit divides x into groups of k values. Each maxout unit outputs the maximum element of one of these groups
- does not have basic form of dot product
  - non-linear function
- generalizes ReLU and leaky ReLU
- does not saturate
- does not die
- but double parameters / neuron-group

Goodfellow, et al., “maxout networks,” ICML, 2013.

Pooling

- invariant to small translations of the input
- reduce size of output channels
  - maxpooling, avgpooling
  - larger stride could have same reduction effect

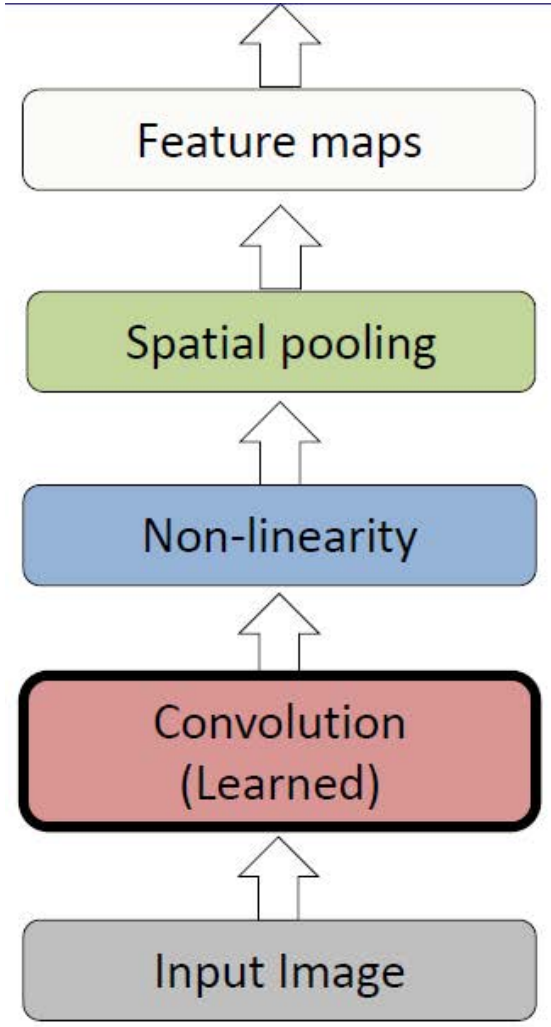


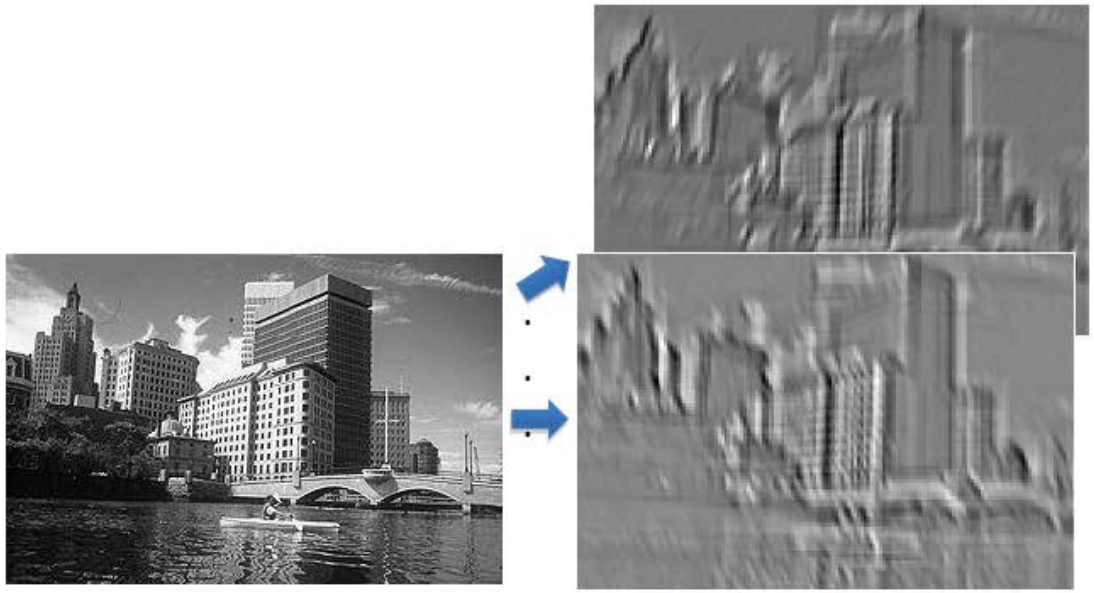
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pooling

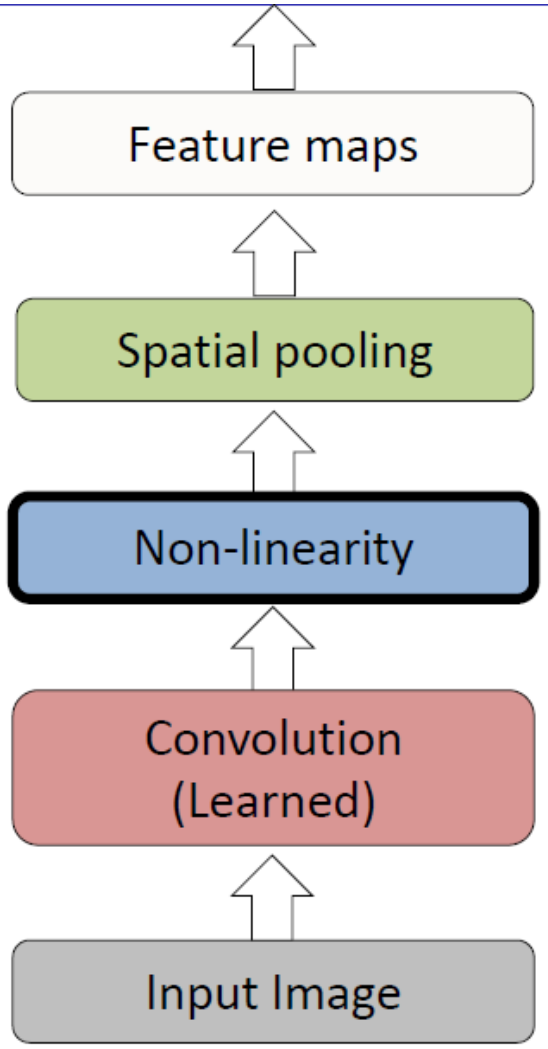
6	8
3	4

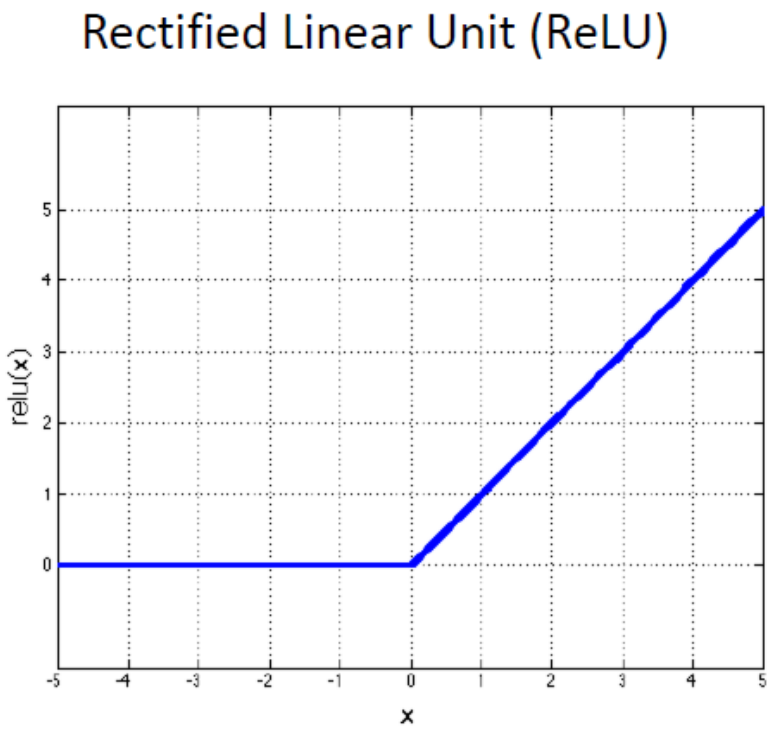
Complete Operations in Conv Layer (1/3): 3D Convolution



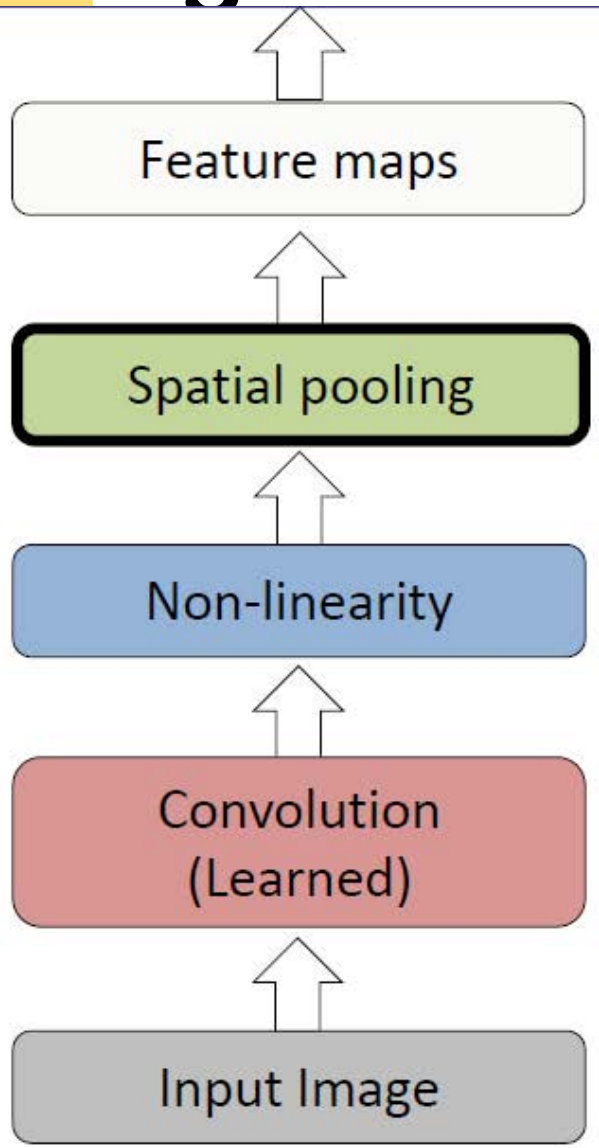


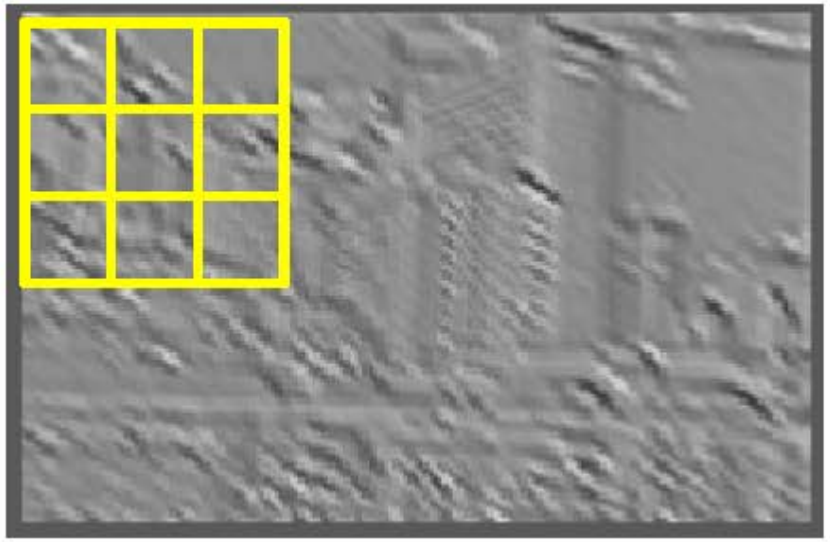
Complete Operations in Conv Layer (2/3): Non-Linear Activation



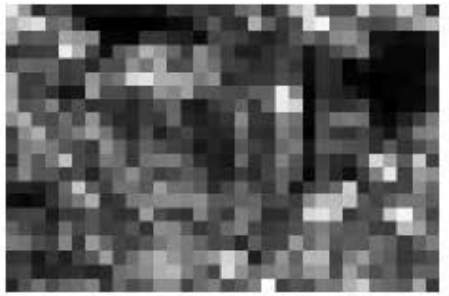


Complete Operations in Conv Layer (3/3): Pooling



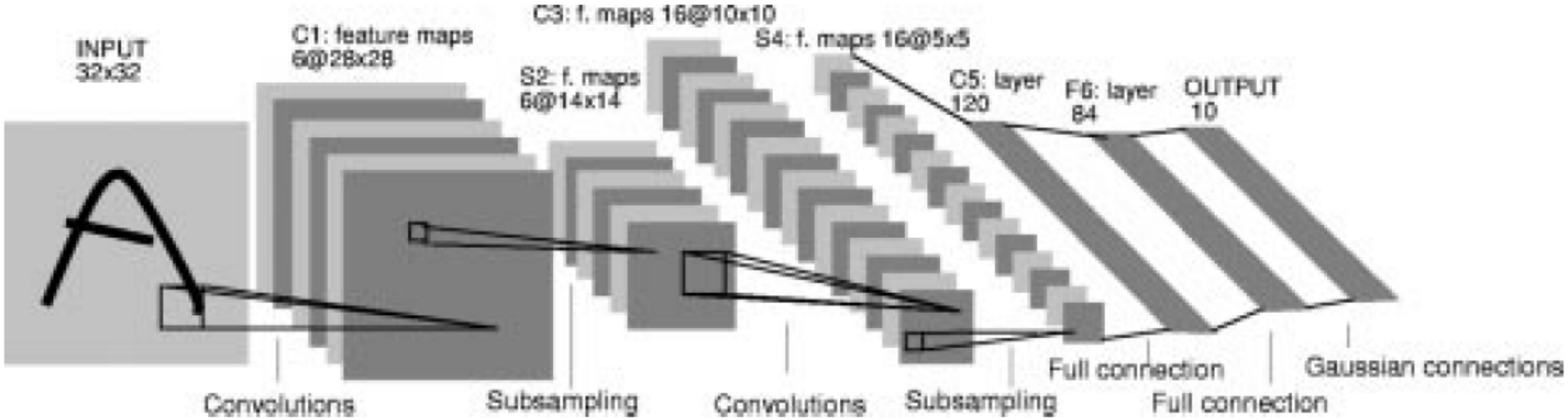


Max



LeNet-5 [1998]

- first CNN for digit recognition
- trained on Modified NIST (MNIST) dataset
- three convolution layers (C1, C3, C5)
- two down-sampling (avg. pooling) layers (S2, S4)
- one fully connected layer (F6)



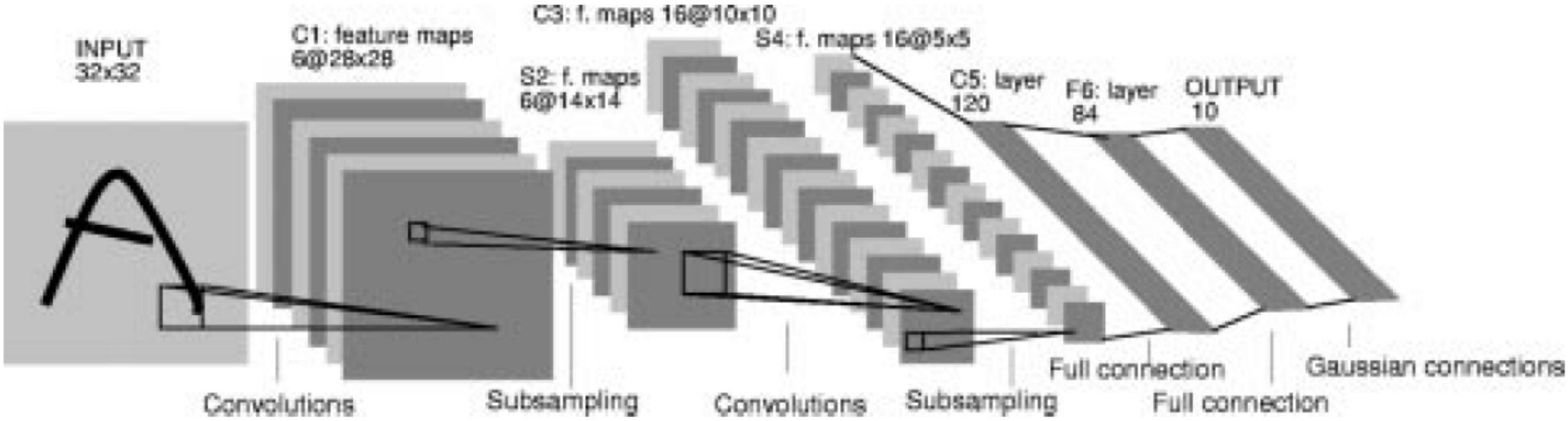
Y. LeCun, et al., “Gradient-Based Learning Applied to Document Recognition,” Proc. IEEE, 1998.

LeNet (2/3)

- C1: 5x5x1x6 convolution (32x32 -> 28x28)
- S2: 2x2 avg. pooling with stride=2 (28x28 -> 14x14)
- C3: 5x5x**{3,4,6}**x16 convolution (14x14 -> 10x10)
- S4: 2x2 avg. pooling with stride=2 (10x10 -> 5x5)
- C5: 5x5x16x1x120 convolution (5x5 -> 1x1)
- F6: 120x1 fully-connected (120 -> 84)

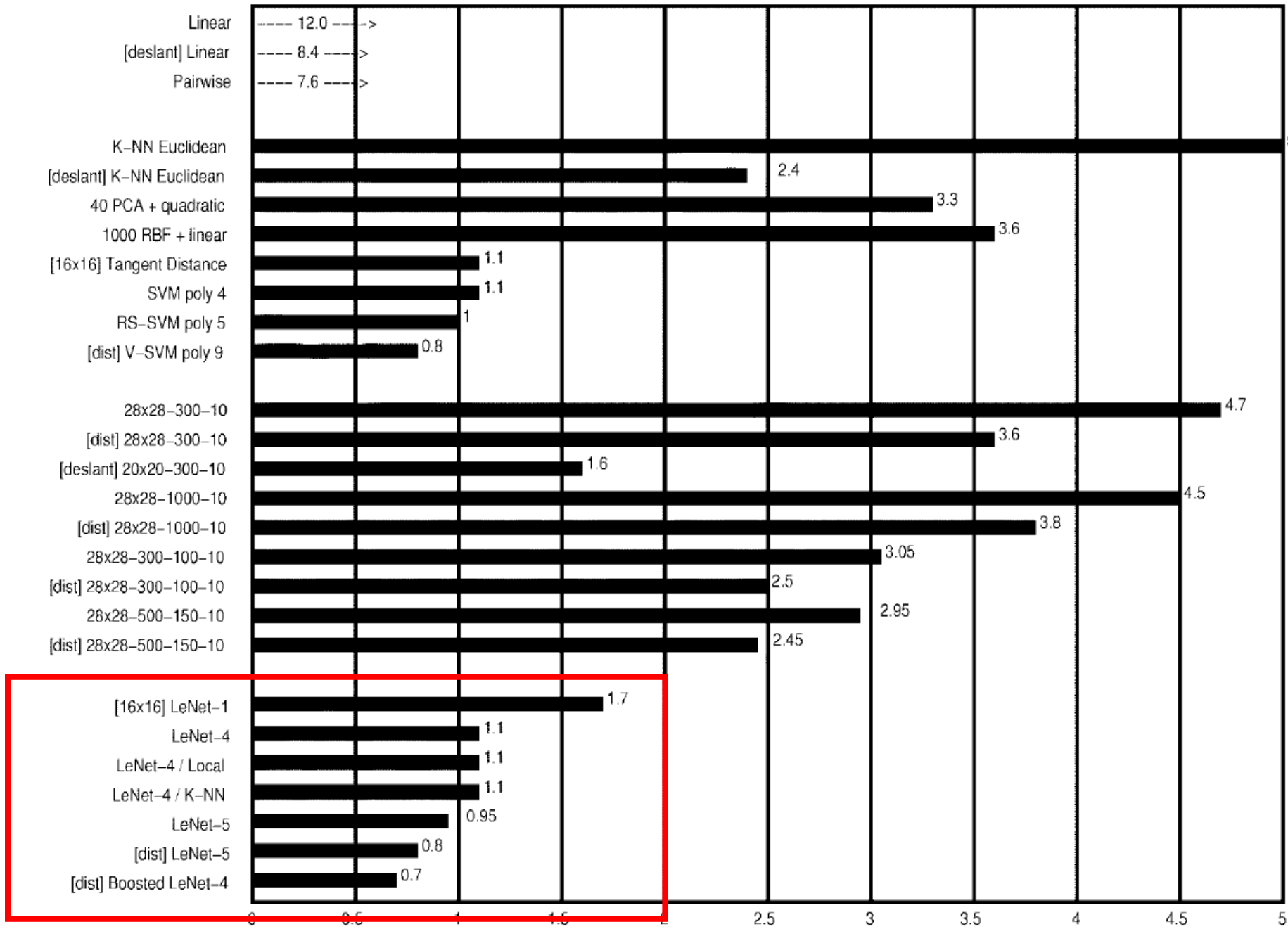
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

combination of input feature maps in C3



LeNet (3/3)

- compared with
  - linear classifier
  - kNN
  - PCA
  - SVM
  - ANN (with various hidden layers)
  - LeNet-5 and its variants



Feature Extraction with Convolution

- different filters extract different features from images
  - filter coefficients (weights) are determined during training in machine learning (instead of handcrafted feature extraction)

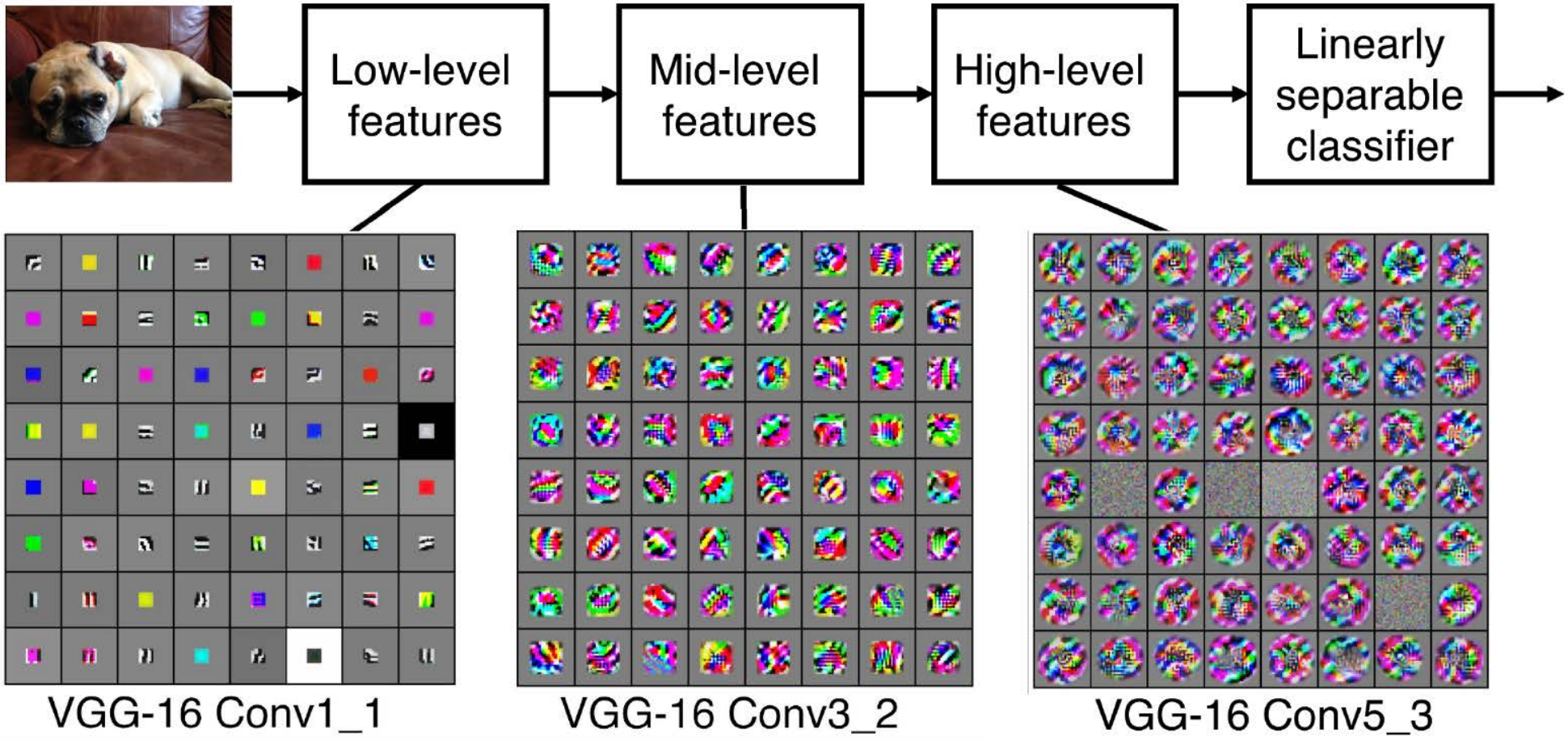
Original

Sharpen

Edge Detect

"Strong" Edge Detect

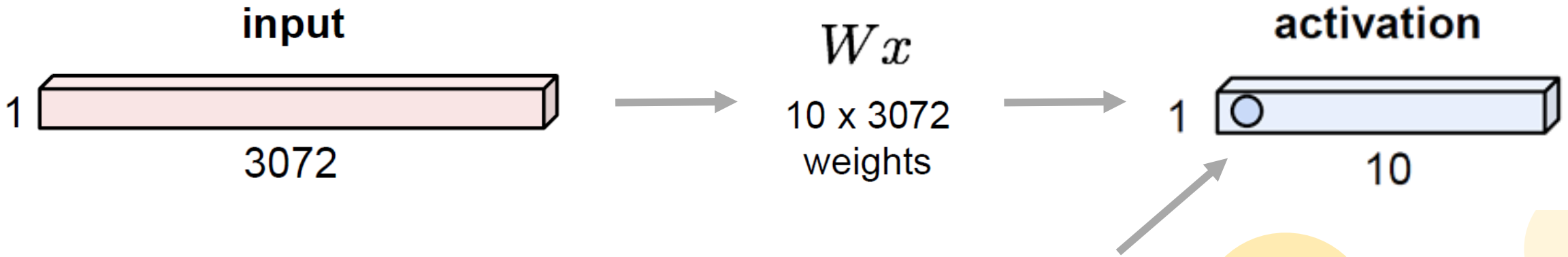
Conv Extracts Features at Different Levels



Fully Connected (FC) Layer

- each neuron looks at the full input volume
- the output of last Conv layer is flatten
  - each pixel corresponds to an input neuron in first FC layer
- FC layer operation can be represented as matrix-vector Wx multiplication
  - matrix W denotes the weights of a FC layer; vector x represents the input neurons

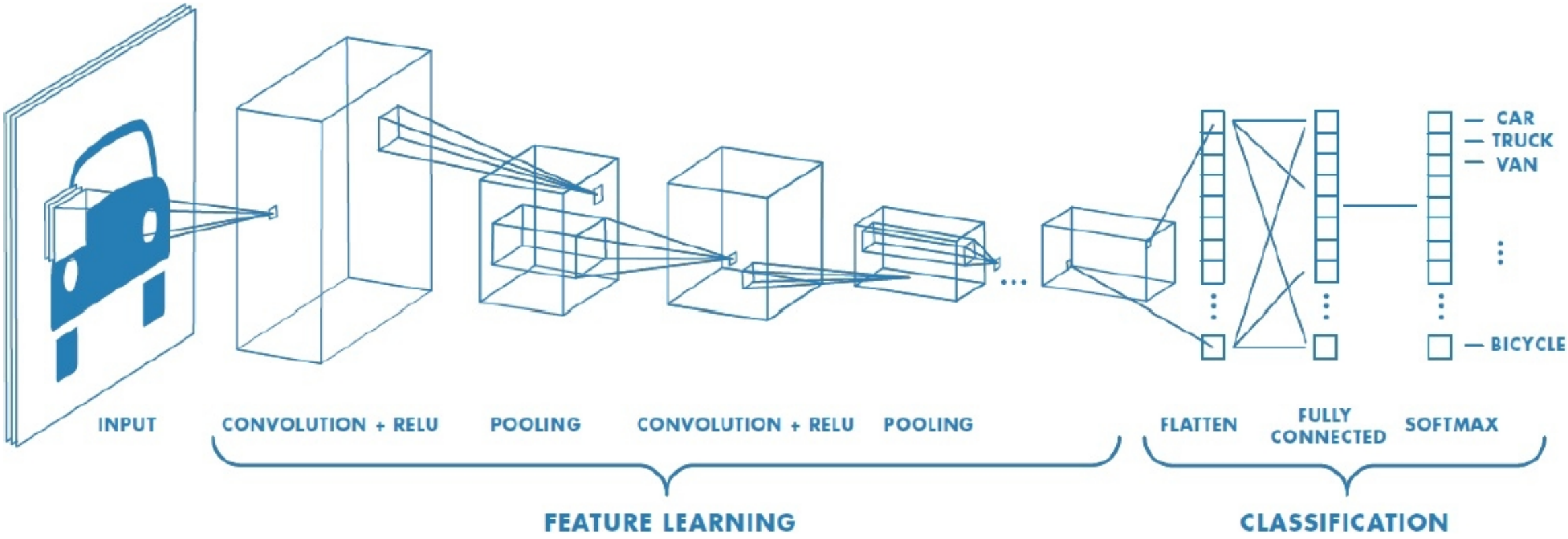
32x32x3 image -> stretch to 3072 x 1



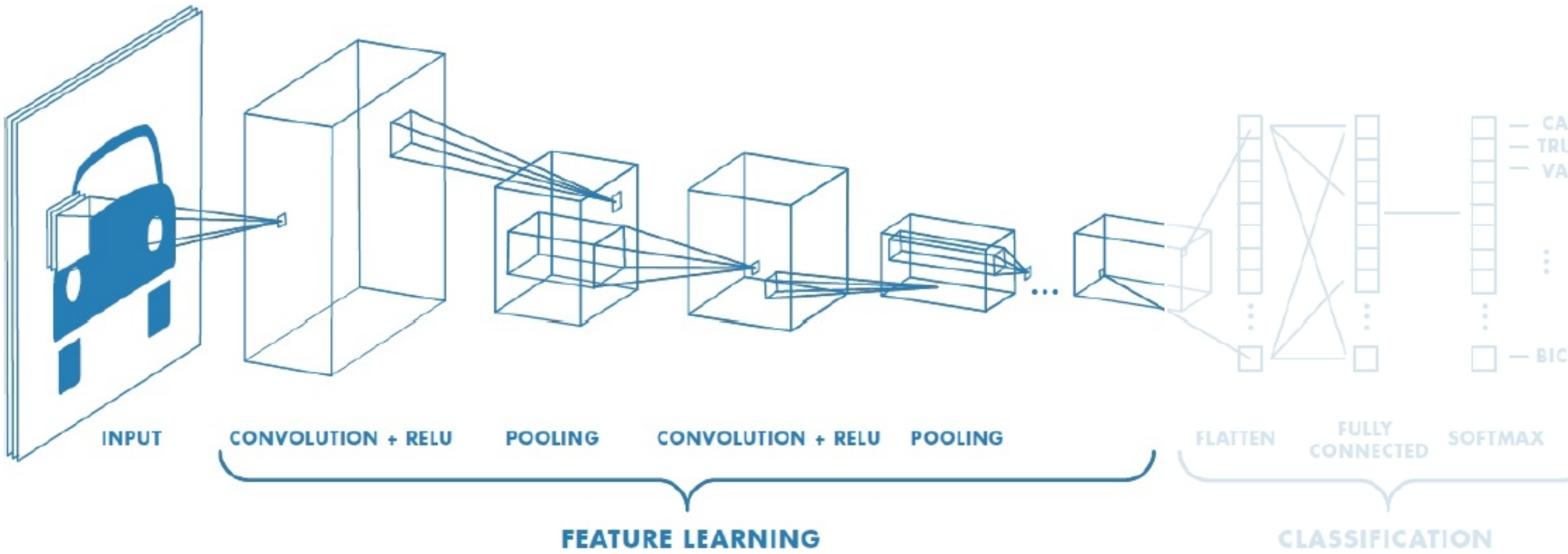
1 number :
The result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Complete CNN for Classification

- Conv layers to extract features of different levels
- FC layers to classify
  - express output as a probability via softmax

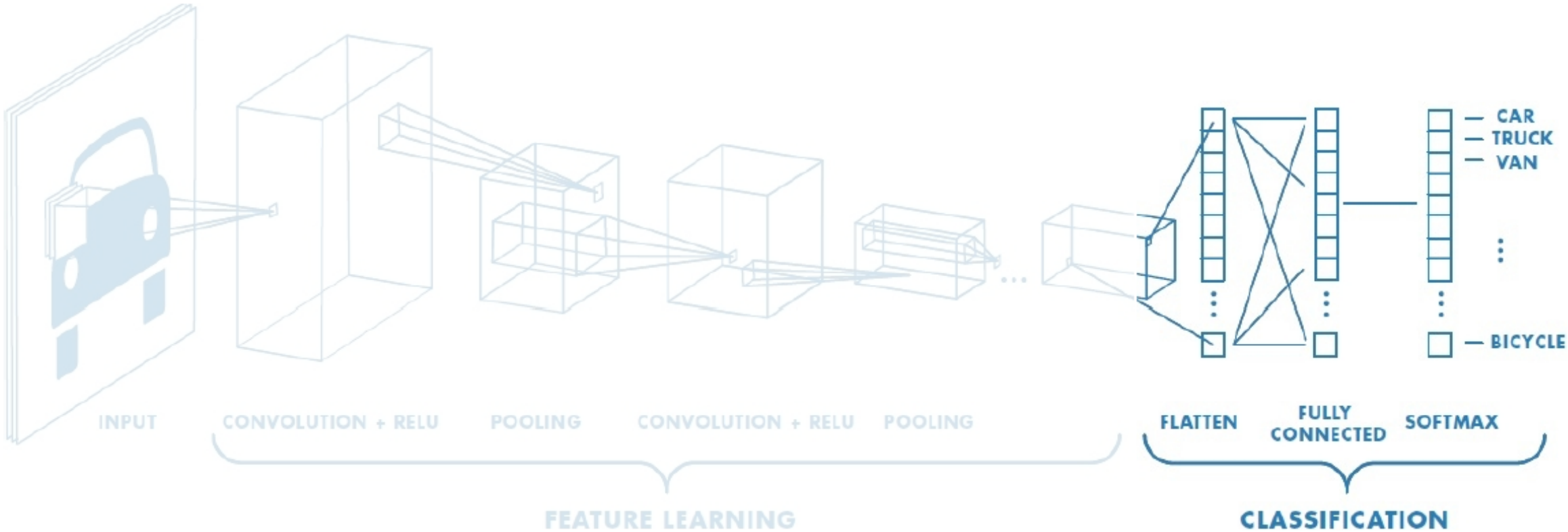


CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

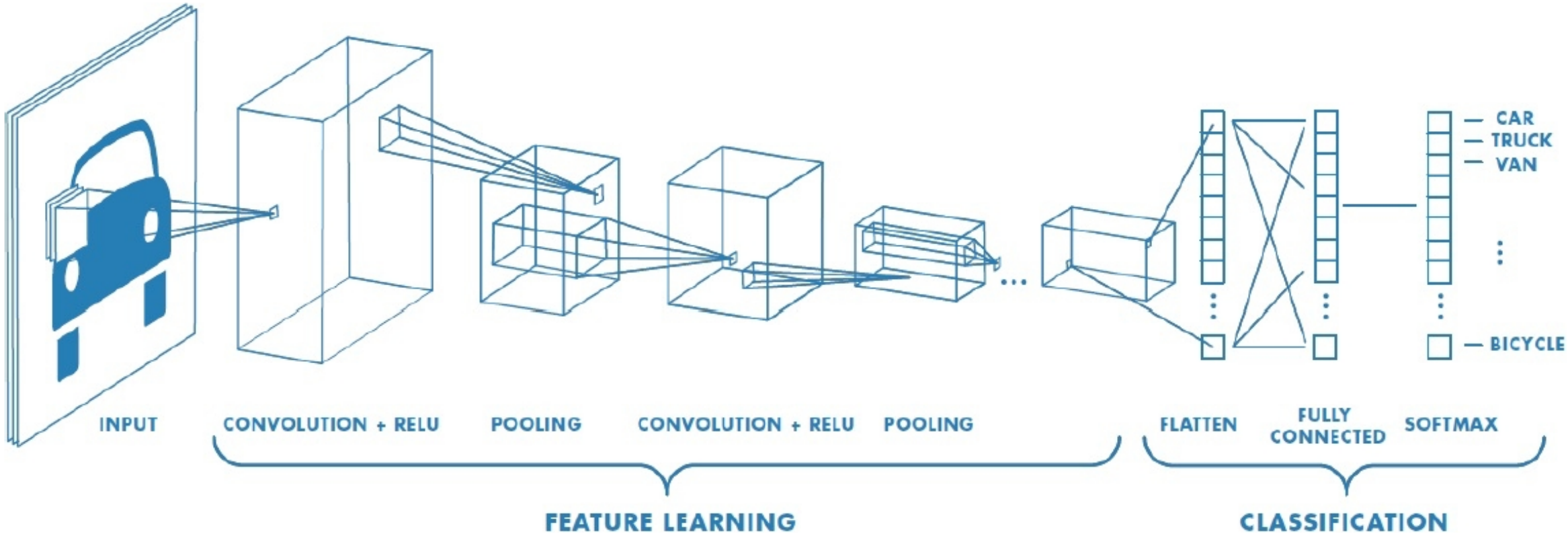
CNNs for Classification: Class Probabilities



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

CNNs: Training with Backpropagation

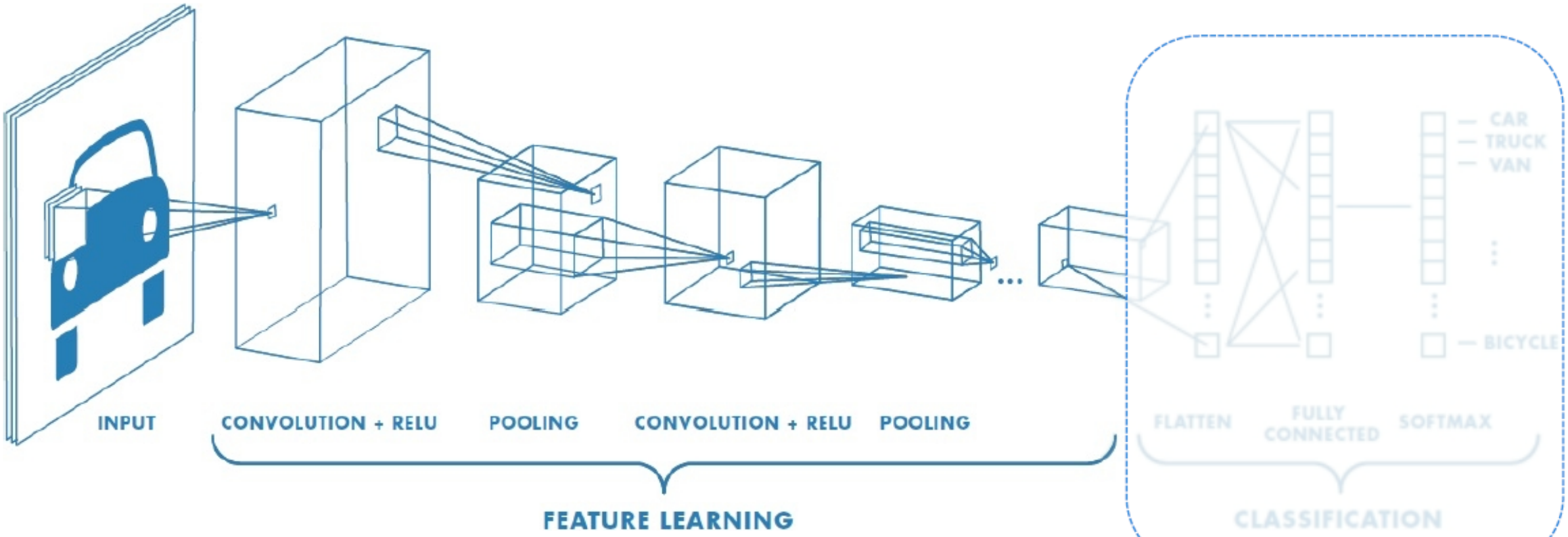


Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

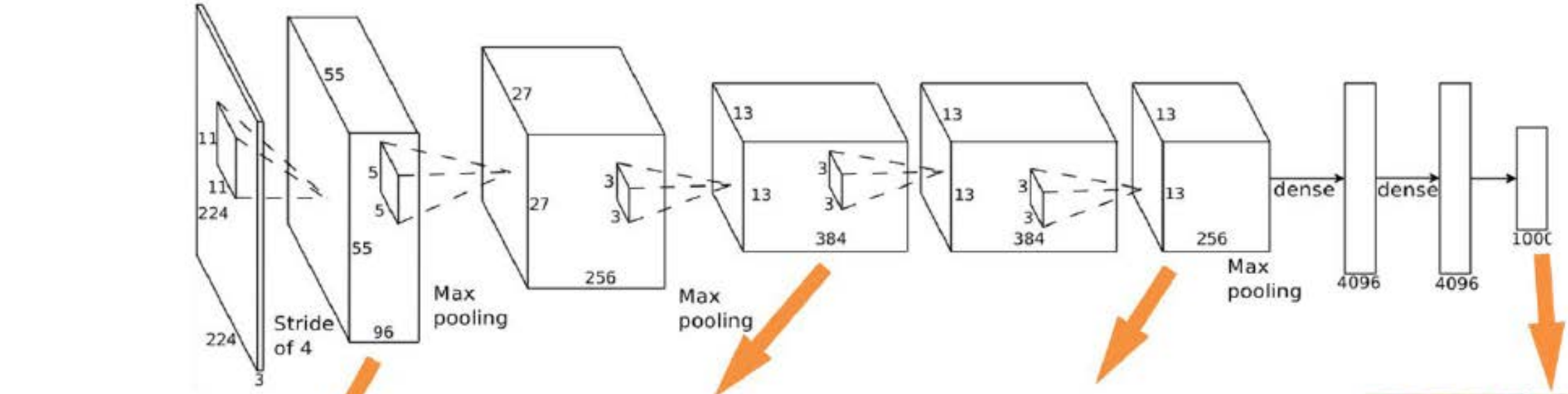
$$J(\boldsymbol{\theta}) = \sum_i y^{(i)} \log\left(\hat{\boldsymbol{y}}^{(i)}\right)$$

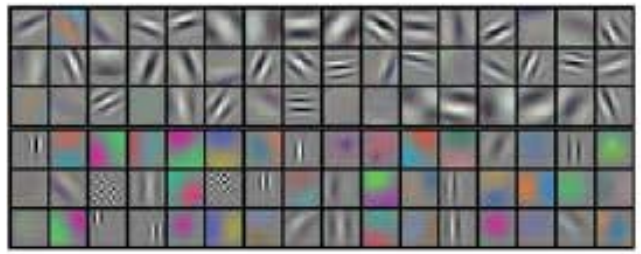
An Architecture for Many Applications



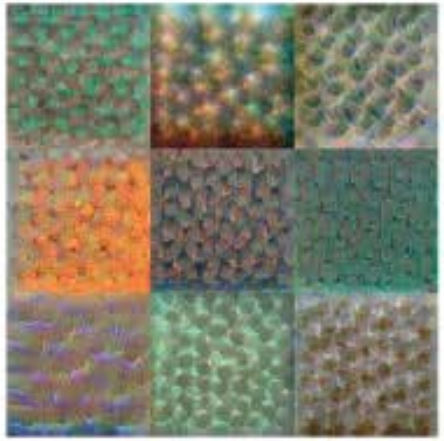
Different image classification domains
Object detection with R-CNNs
Segmentation with fully convolutional networks
Image captioning with RNNs

Extracted Features in AlexNet





Conv 1: Edge+Blob



Conv 3: Texture

Numerical

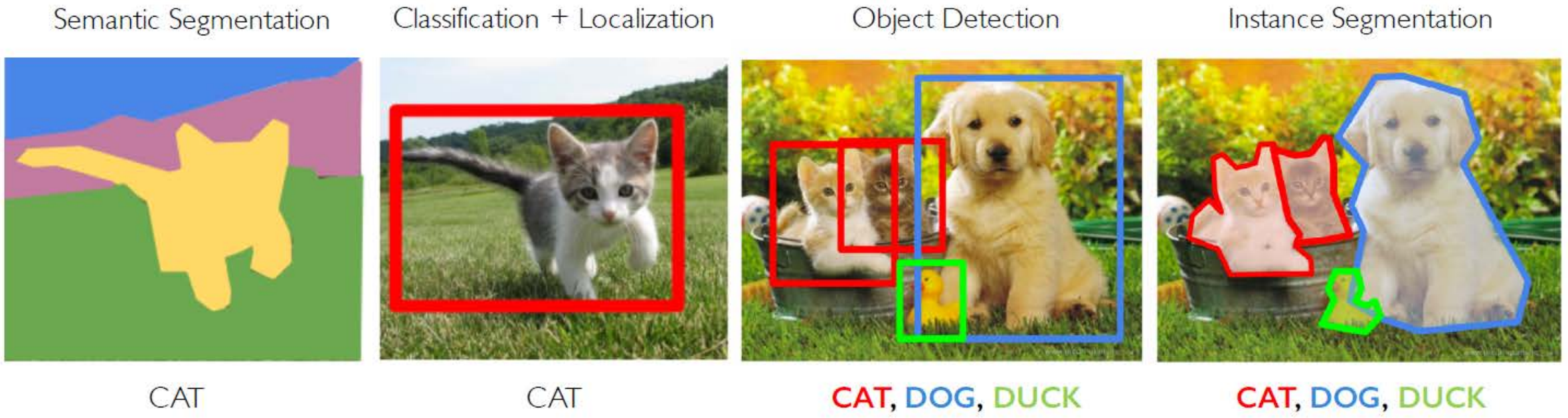
Data-driven

Conv 5: Object Parts

Fc8: Object Classes

Beyond Classification

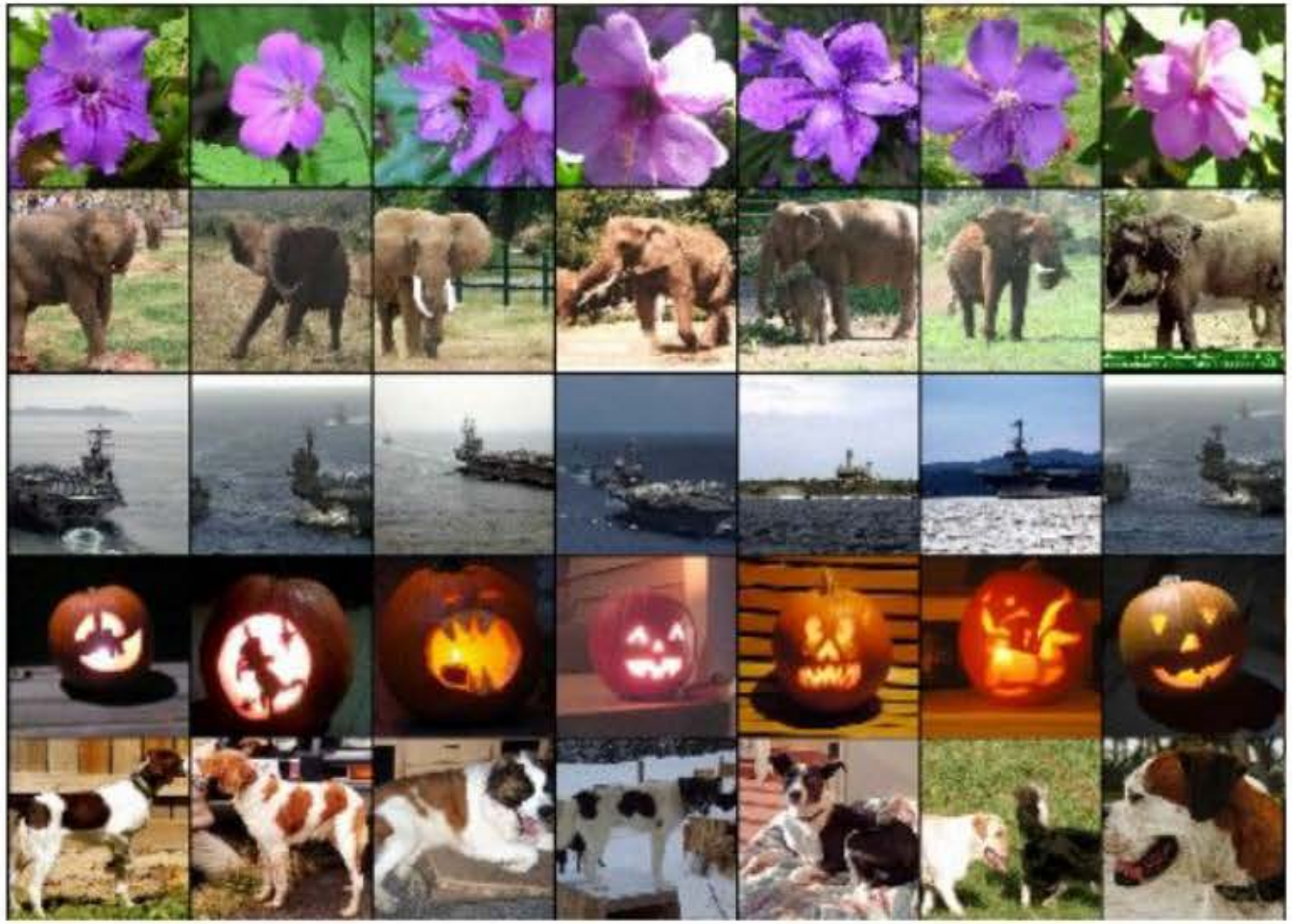
- classification + localization
  - detection
- segmentation
- possibly different models other than CNN
  - RNN (for image caption), GAN (for segmentation and style transfer)



CNN Applications (1/5): Classification

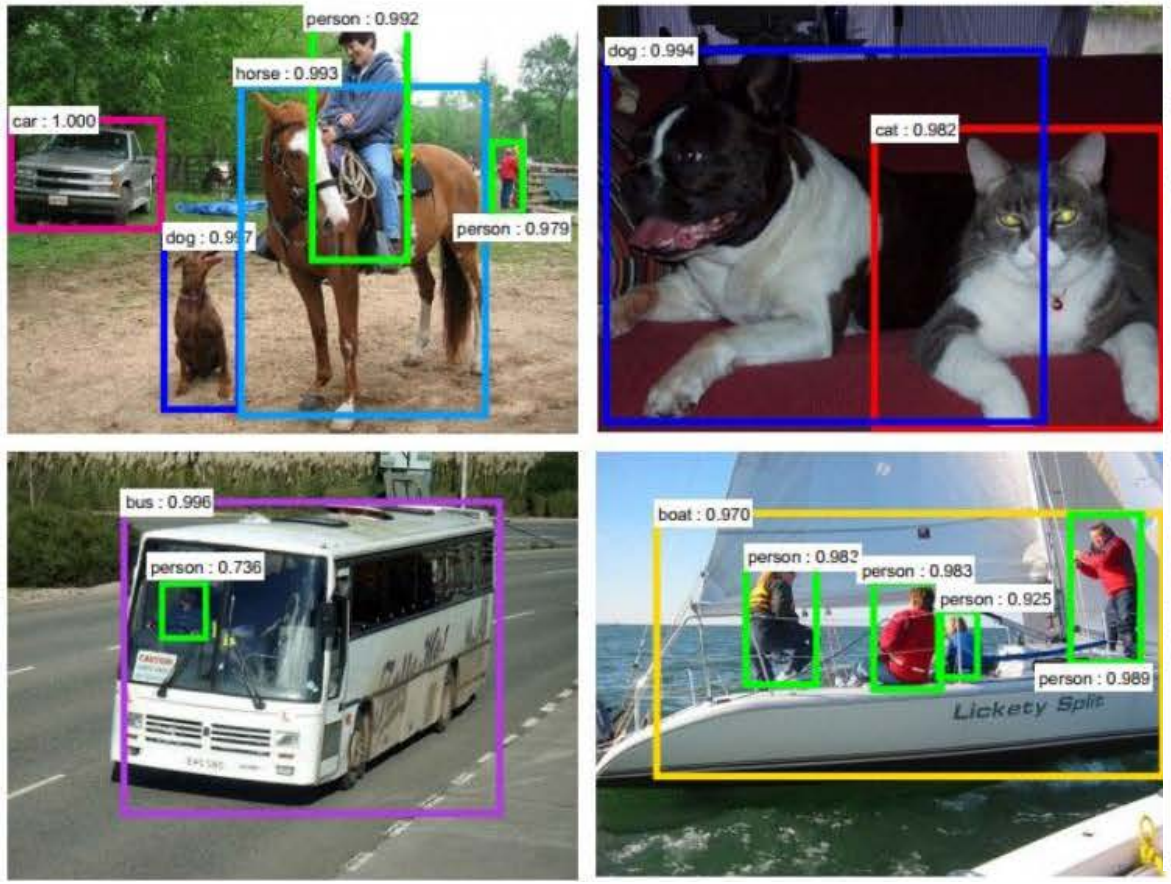
Classification

Retrieval



CNN Applications (2/5): Detection

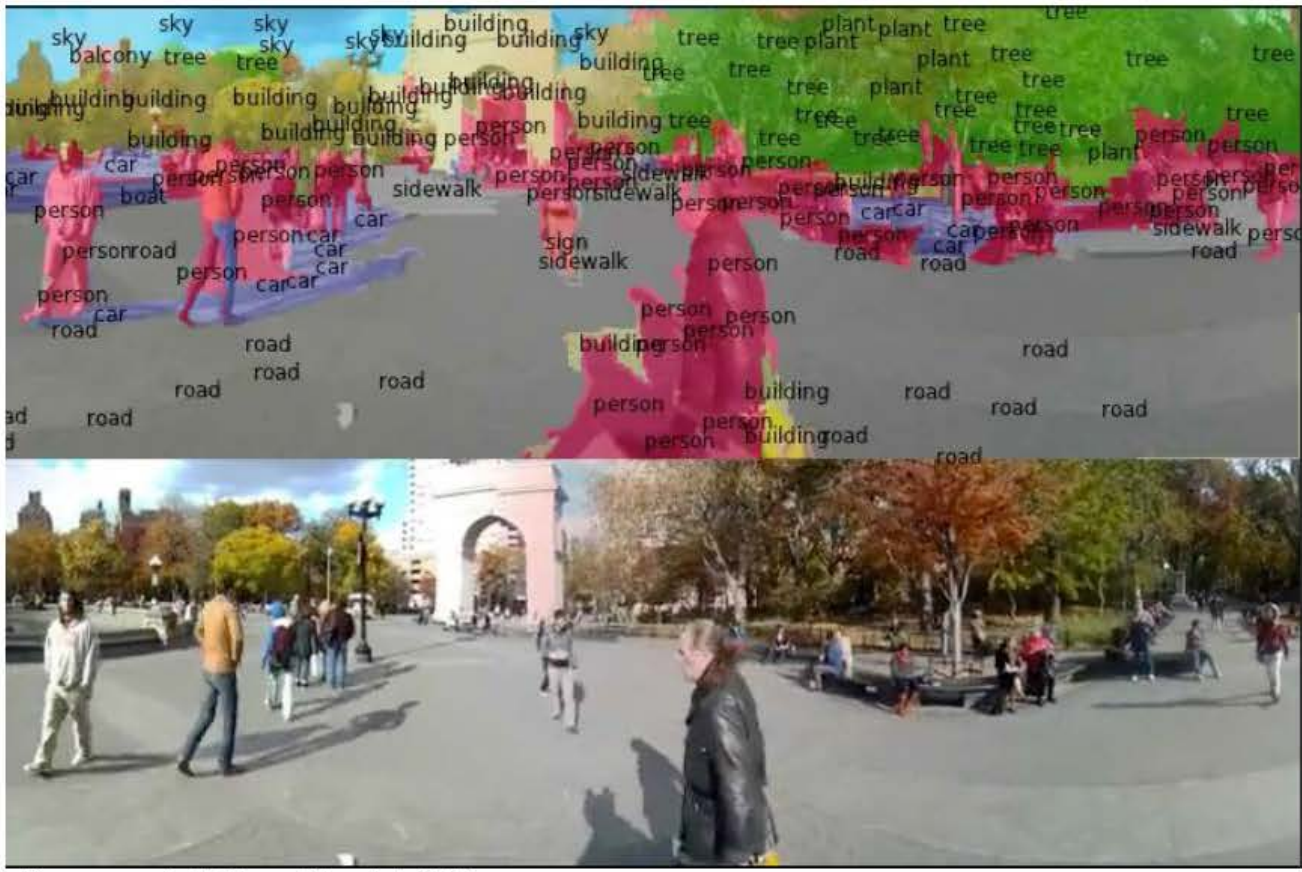
Detection



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

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

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

CNN Applications (3/5): Captioning

No errors

A white teddy bear sitting the grass

Minor errors

A man in a baseball uniform throwing a ball

Somewhat related

A woman is holding a cat in her hand

Image Captioning

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

A man riding a wave on top of a surfboard

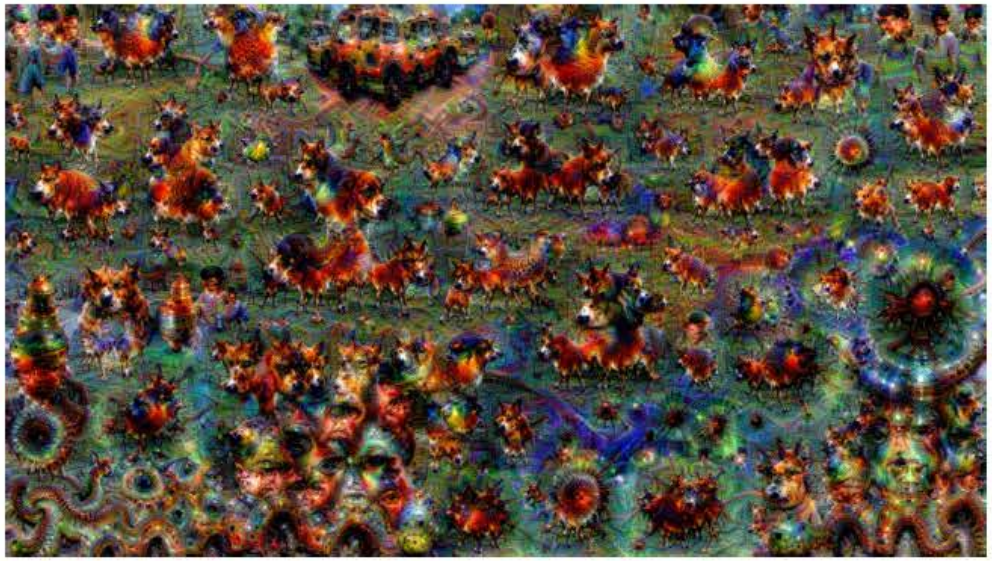
A cat sitting on a suitcase on the floor

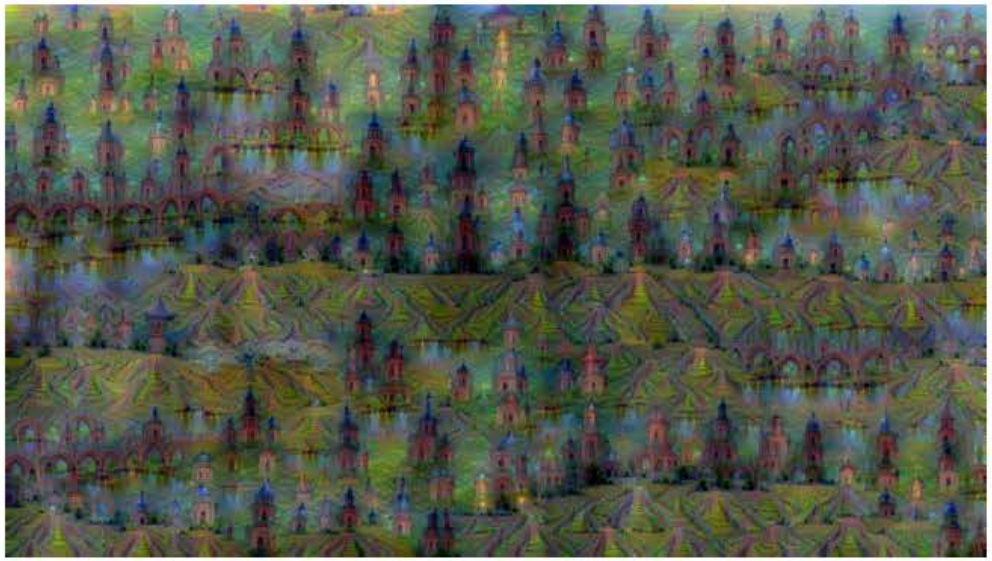
A woman standing on a beach holding a surfboard

All images are CCO Public domain:
https://pixabay.com/en/luggage-antique-cat-164301/
https://pixabay.com/en/teddy-DIUsh-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-soort-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-plaver-shortstop-infield-1045263/

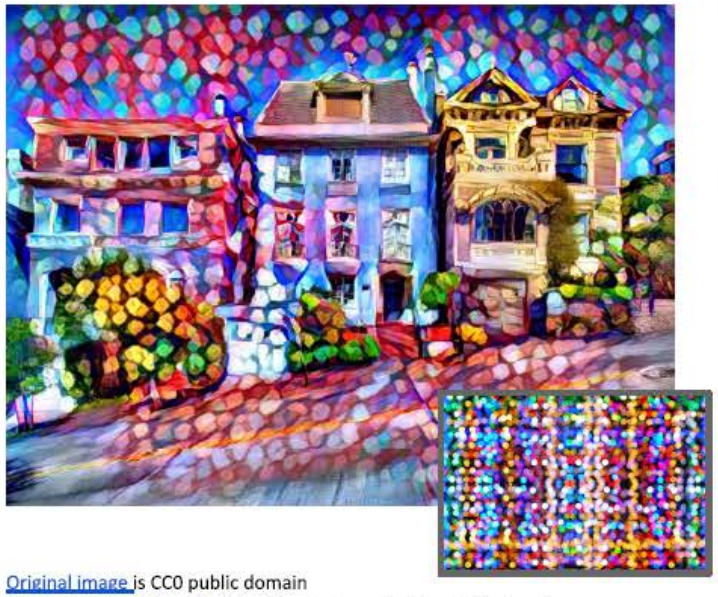
Captions generated by Justin Johnson using Neuraltalk2

CNN Applications (4/5):Style Transfer

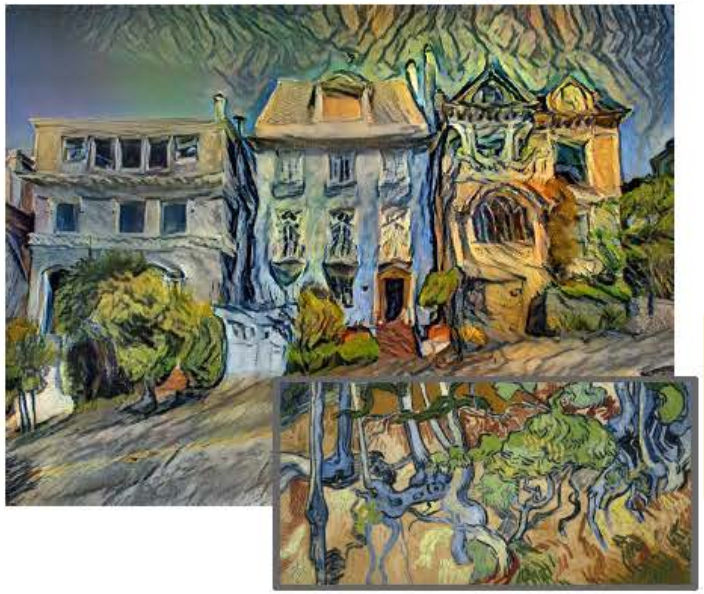




Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.



Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;



Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

CNN Applications (5/5): Photo Enhancement

Datasets

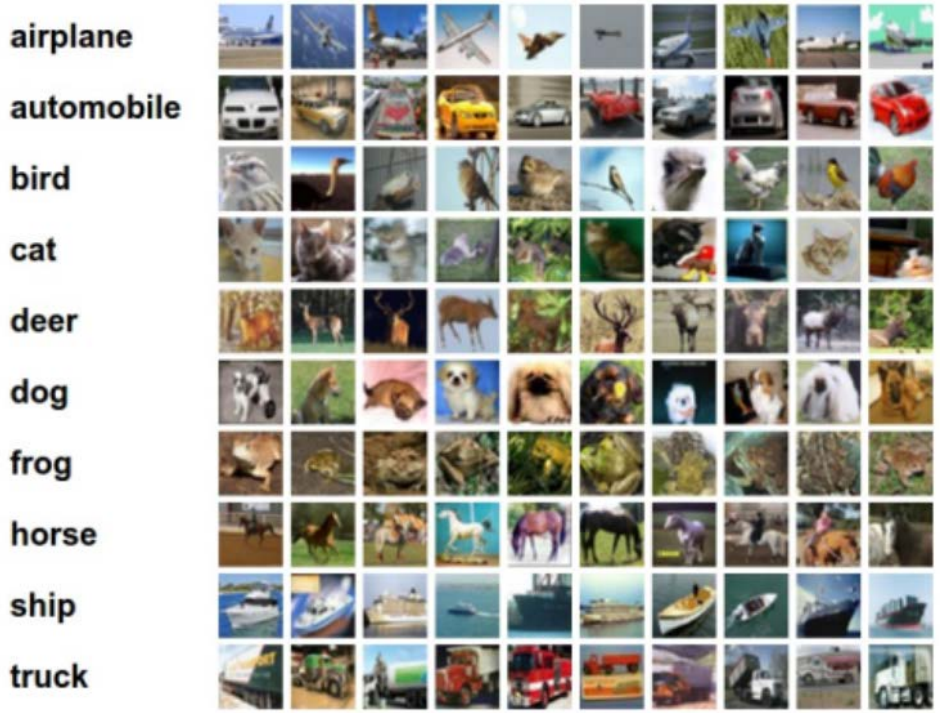
- MNIST
- CIFAR-10/100
- ImageNet
- VOC
- COCO
- SVHN
- ...

dataset	image size	# of samples	# of categories	image types	training set	test set
MNIST	28x28 gray	70K	10	digits	60K	10K
ImageNet	256x256 color	> 15 M	> 22,000	vision	1.2M	150K
Caltech-101	300x200 color	>9K	101	vision	5K	3K
CIFAR-10/100	32x32 color	6 K	10, 100	vision	60K	10K
PASCAL VOC	500*334 color	>10K	21	vision	10K	5K
LabelMe (MIT)	-	>2K	7	vision	2K	1K
SVHN	32x32 color	>70K	10	s	600K	26K
COCO	-	330K	80	vision	10K	5K
CompCars	-	136K	-	cars	-	-
Stanford Cars	-	16K	196 car classes	cars	8K	8K
KITTI	64*32 color	>15K	5	pedestrian	>7K	>7K
INRIA	64*128 color	>0.8K	1	pedestrian	614	288

MNIST, CIFAR-10, ImageNet Datasets

- MNIST: 10 categories, 60,000 training images of 28x28, 10,000 test images
- CIFAR-10: 10 catogories, 60,000 training images of 32x32x3, 10,000 test images
- ImageNet: 1000 categories, 1,200,000 training images of 256x256x3, 150,000 test images

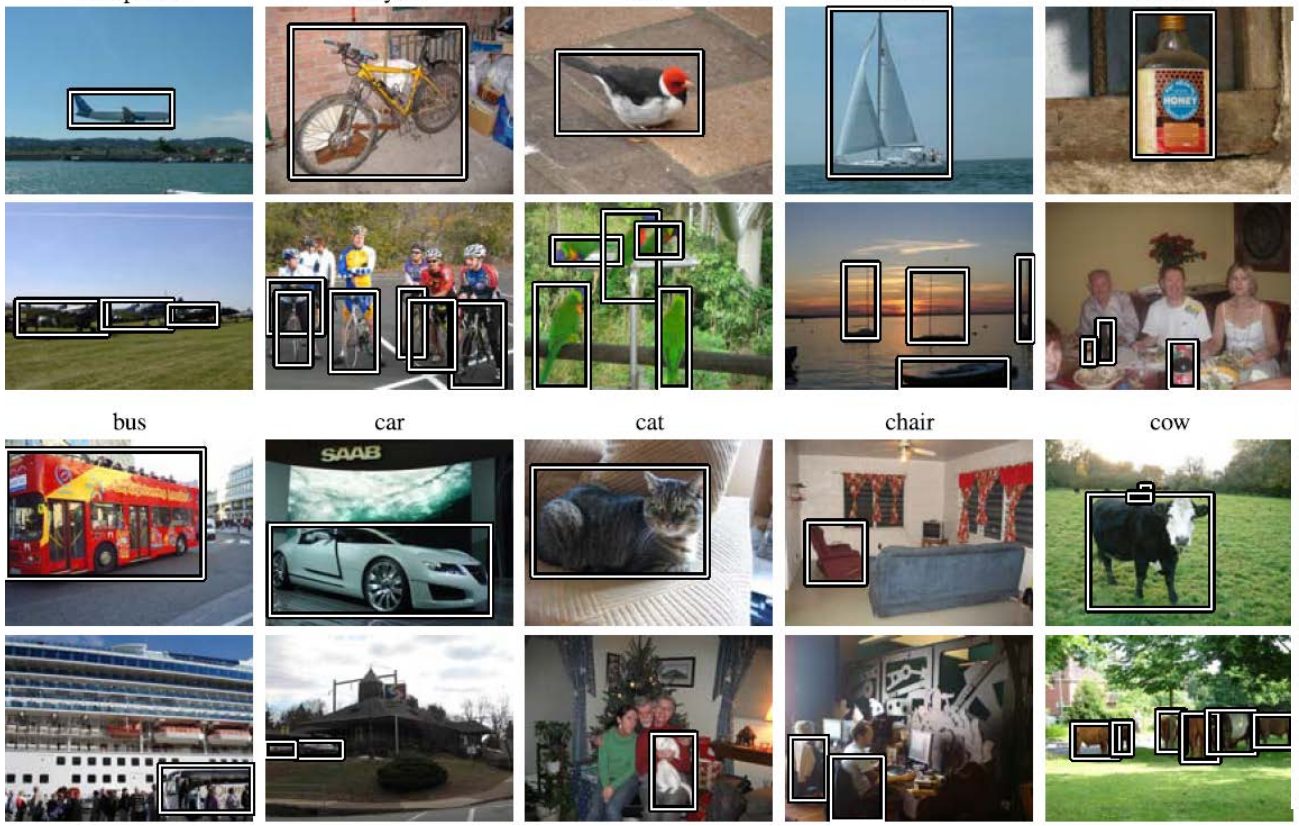
MNIST: handwritten digits

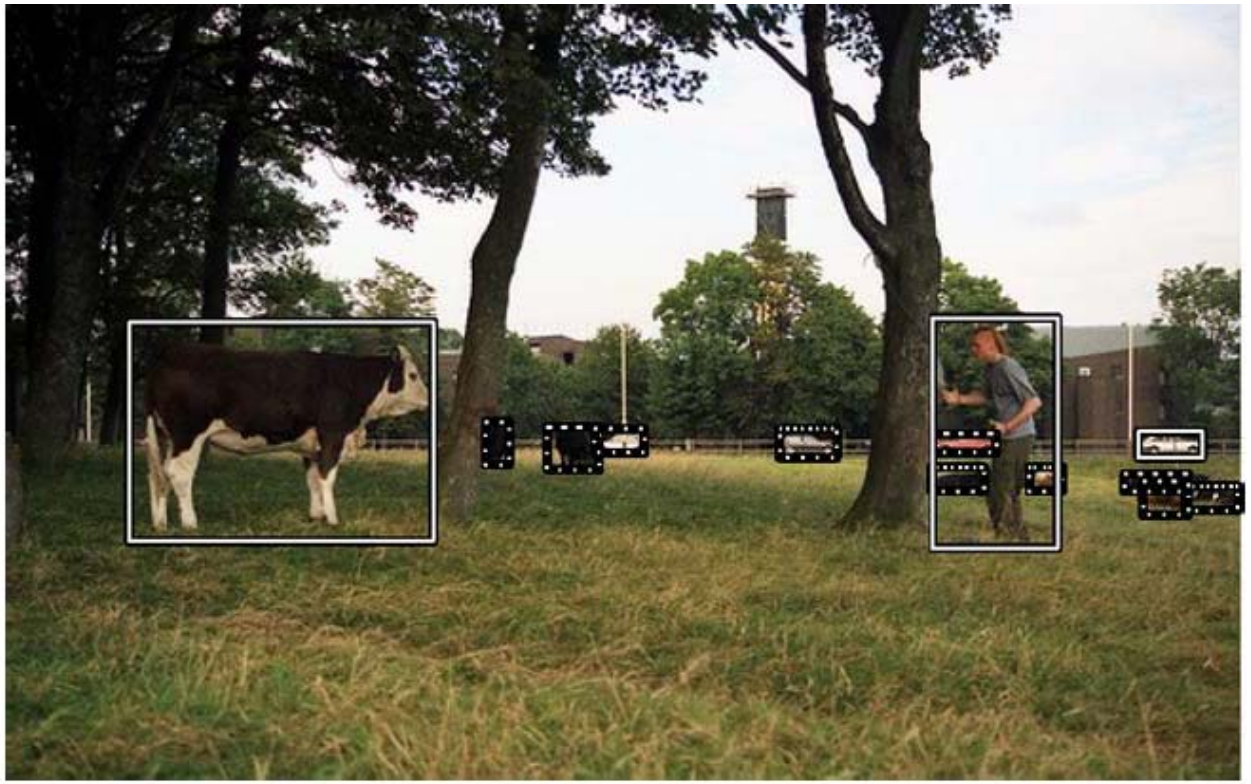


ImageNet:
22K categories. 14M images.

PASCAL VOC*

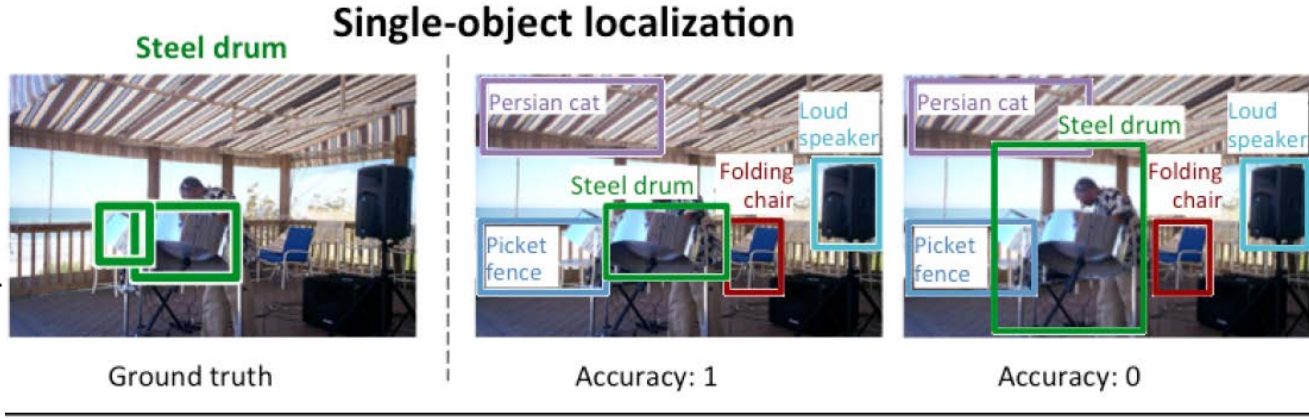
(b) **chair**: highest ranked negative images

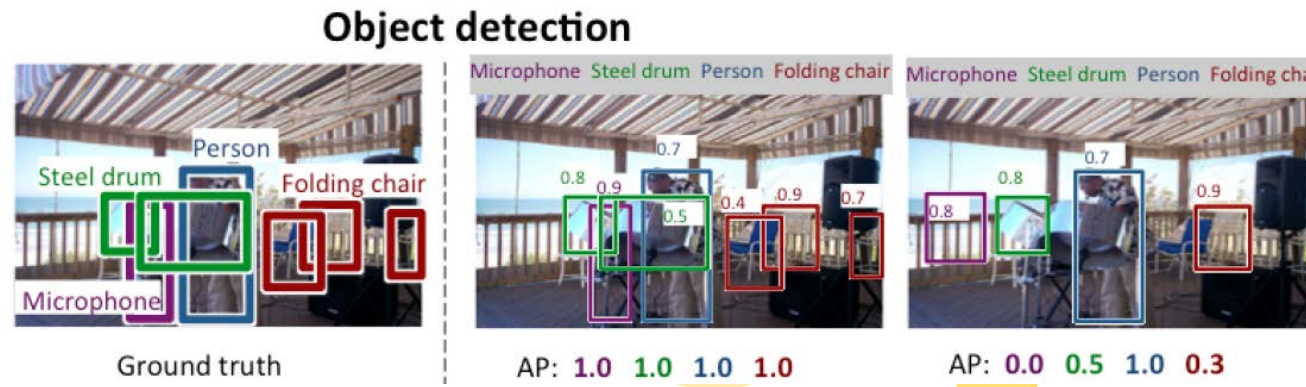




* M. Everingham, et al., "The PASCAL Visual Object Classes (VOC) Challenge", IJCV, 2010.

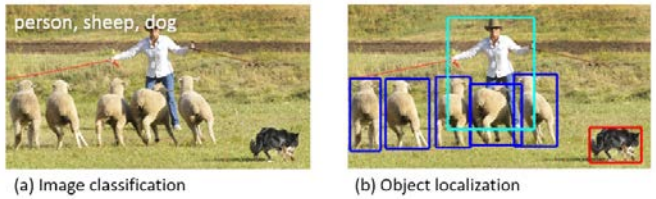
ImageNet*





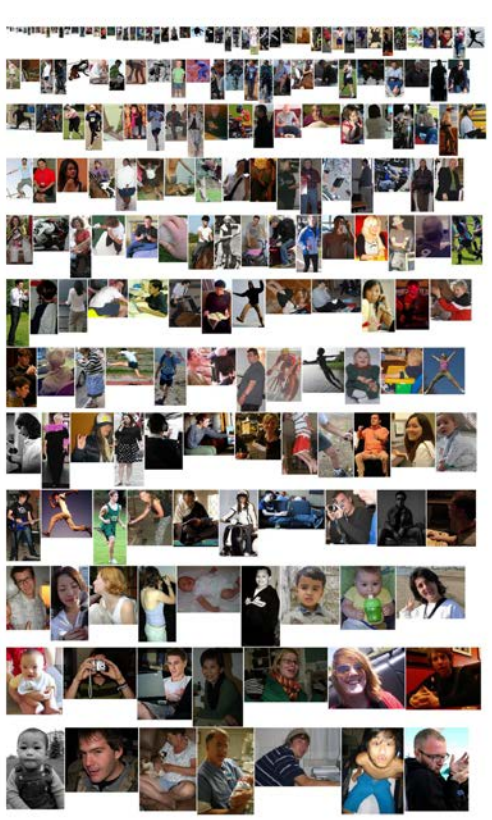
* O. Russakovsky, et al., "ImageNet Large Scale Visual Recognition Challenge," 2015.

COCO



(a) Image classification

(b) Object localization



(a) PASCAL VOC.

(b) MS COCO.

* T.-Y. Lin, et al., “Microsoft COCO: Common Objects in Context,” ECCV, 2014.

LeNet Demo

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

CIFAR-10 demo

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