

# Convolutional Neural Networks

# Recap: Goal of ML

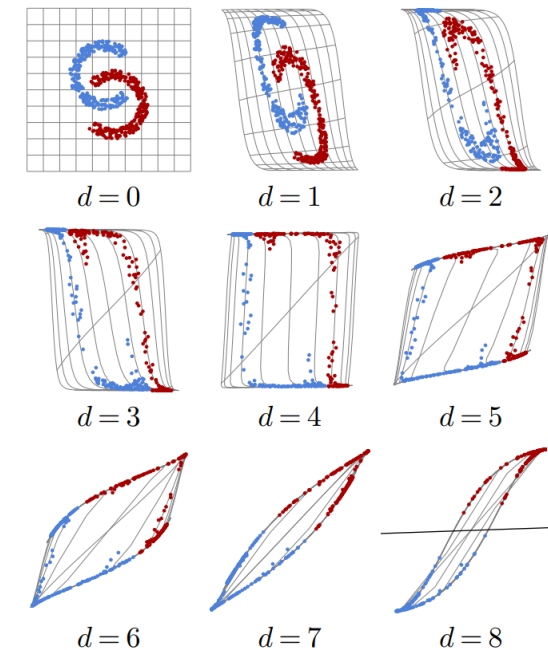
generalization from optimization on training data set  
(approximation of true data generating probability distribution by empirical risk minimization)

- fitting: complex function approximation
- for generalization: learning of good abstraction/representation of data/concepts

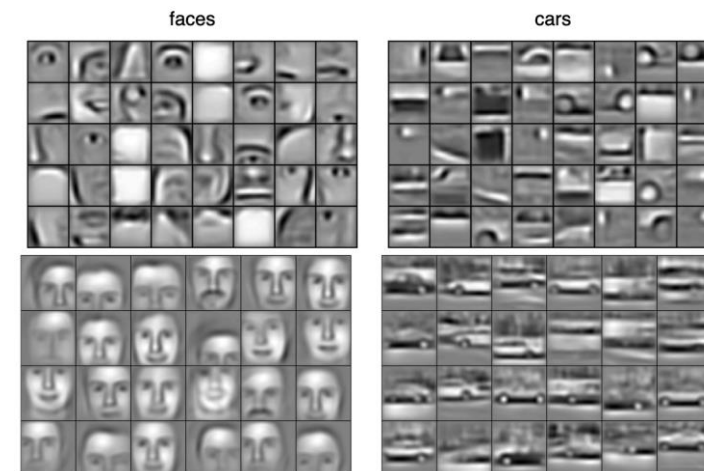
→ deep learning methods (MLP, CNN, ...) optimal candidates

e.g., CNNs can learn hierarchical representation by means of many convolutional and pooling layers

the deeper the better (accuracy, hierarchical representation)



[source](#)



[source](#)

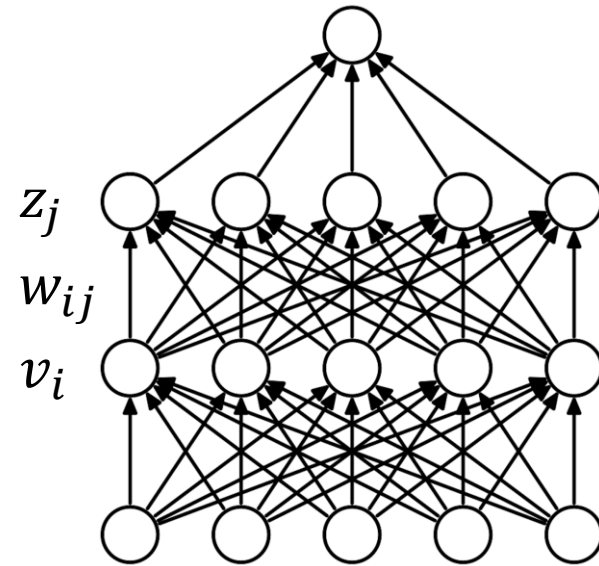
# Recap: Feed-Forward Neural Networks

computation in usual feed-forward network:

scalar input values  $z_j$  to activation function of nodes  $j$  in hidden or output layer as matrix multiplication of scalar output values  $v_i$  from activation function of nodes  $i$  from previous layer with connecting weights  $w_{i,j}$

$$z_j = \sum_i v_i w_{i,j}$$

dropping dimension of different training observations in this view → loading full batch or mini-batches



# Grid-Like Data

data with grid-like topology (spatial structures), e.g.:

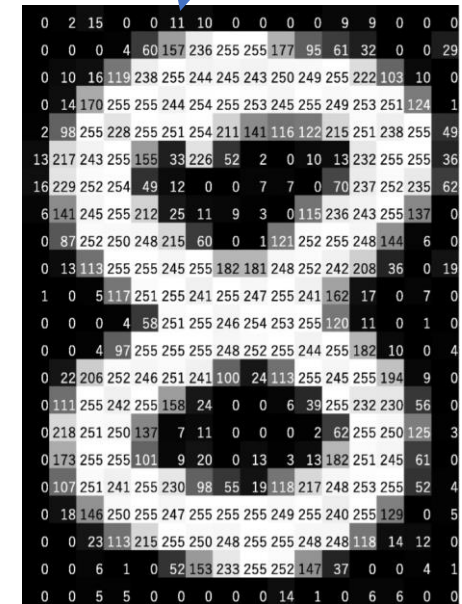
- time-series data: 1-D grid of data taken at regular time intervals (can also be done with recurrent neural networks or transformers)
- image data: 2-D grid of pixels (→ computer vision)

convolutional networks:

neural networks using convolutions with kernels (local groups of values, e.g., pixels from an object, highly correlated) in place of general matrix multiplications in at least one of their layers

→ highly regularized feed-forward networks

scalar value (like in usual feed-forward network)



[source](#)

# Convolution Operation

to be exact, usually rather cross-correlation instead of convolution operation (what would have  $-$  here):

$$Z_{i,j} = (K * V)_{i,j} = \sum_{m,n} V_{i+m,j+n} K_{m,n}$$

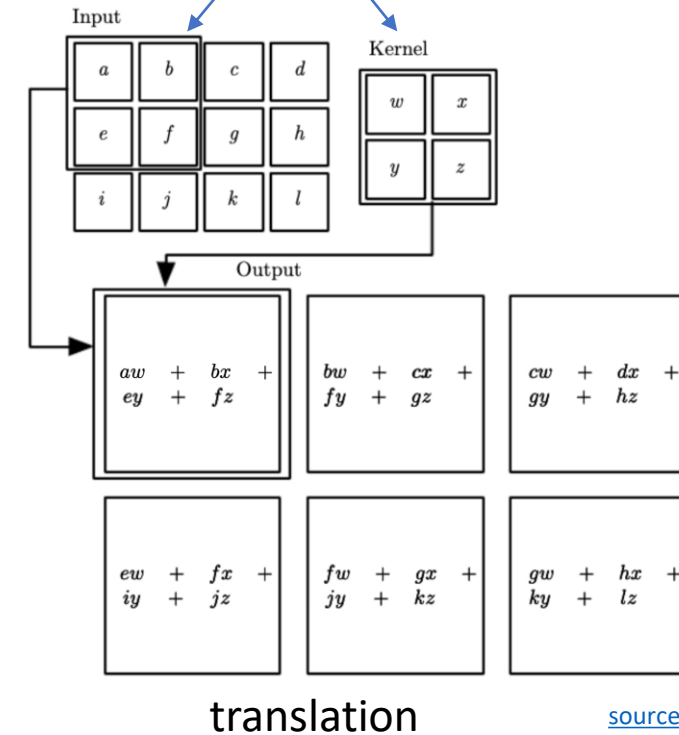
feature map
input (matrix)
kernel (matrix)

again, dropping dimension of different training observations  
 matrices  $\rightarrow$  tensors: several input channels  $c$  (e.g., RGB) and  
 several output channels  $f$  (different feature maps, e.g., vertical  
 edge, nose, ear, ...)

$$Z_{f,i,j} = \sum_{c,m,n} V_{c,i+m,j+n} K_{f,c,m,n}$$

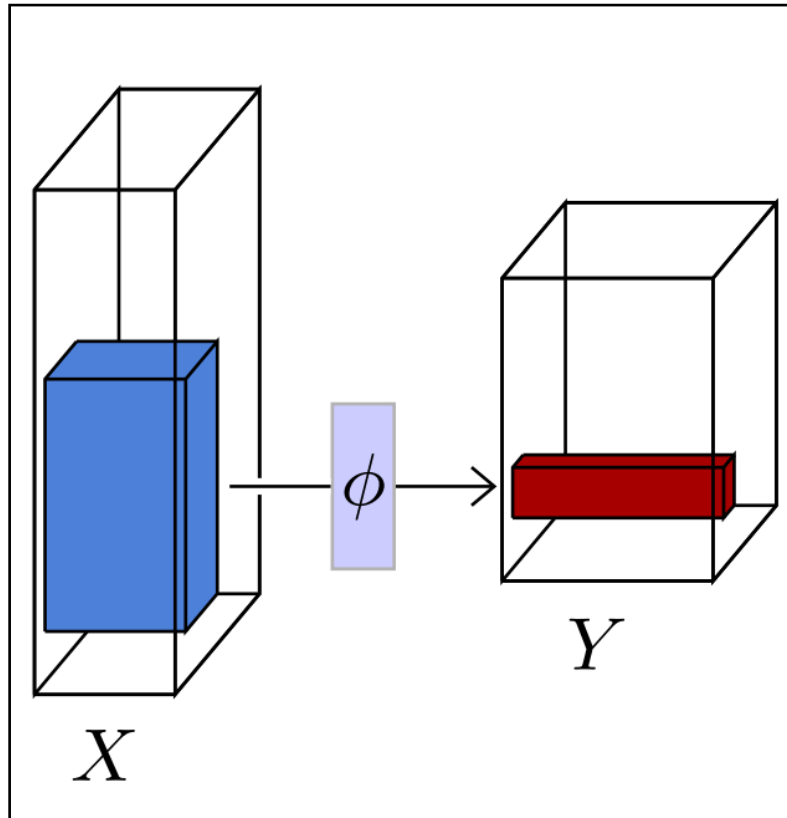
**learned parameters:**

- connection to **local patches** of previous layer's feature maps
- shared over entire image (common learning across image)

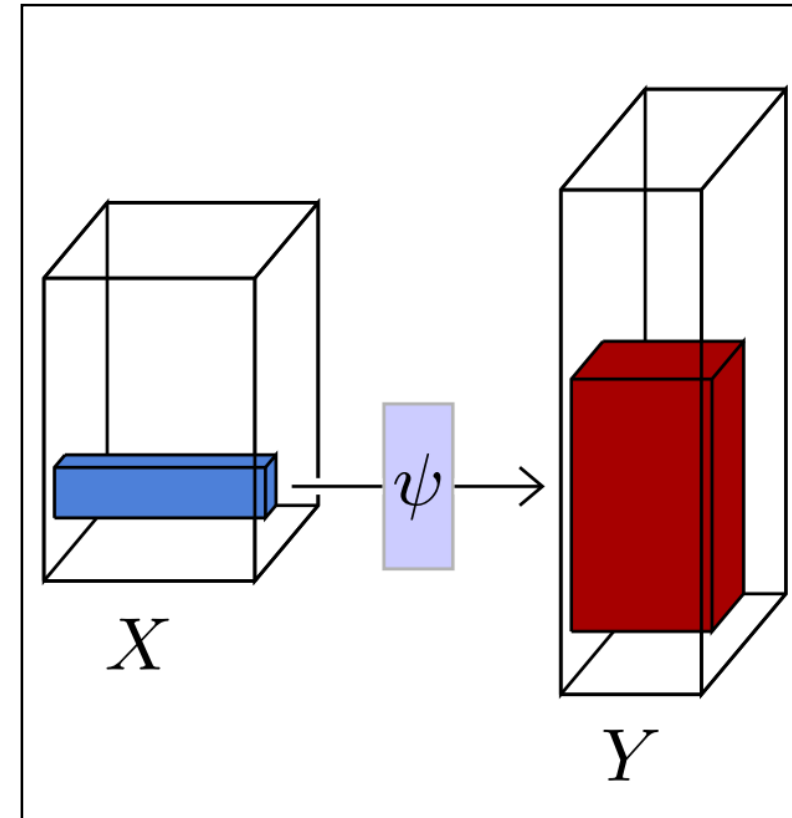


# Decrease or Increase Dimensionality

typical use cases  
discussed here



*2D convolution*



*2D transposed  
convolution*

e.g., decoder side  
of autoencoders  
or U-nets in  
diffusion models

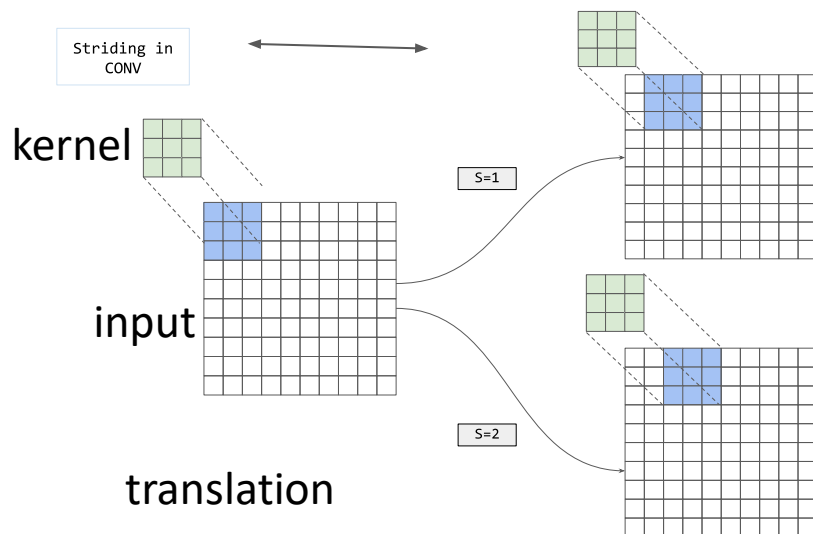
# Important Details: Striding and Padding

need to define how to stride over image

$$Z_{f,i,j} = \sum_{c,m,n} V_{c,i \times s + m, j \times s + n} K_{f,c,m,n}$$

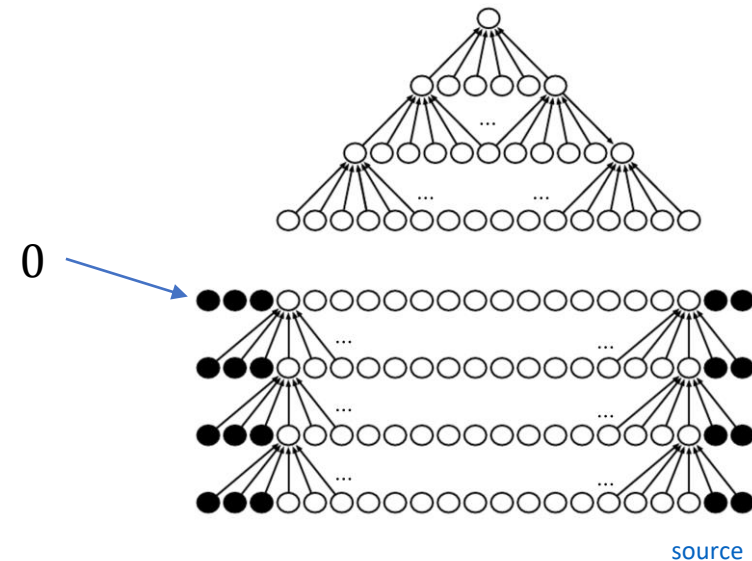
$s > 1$  corresponds to down-sampling

→ fewer nodes after convolutional layer



zero-padding of input to make it wider:

otherwise shrinking of representation with each layer (depending on kernel size) → allowing large kernels and slow shrinkage

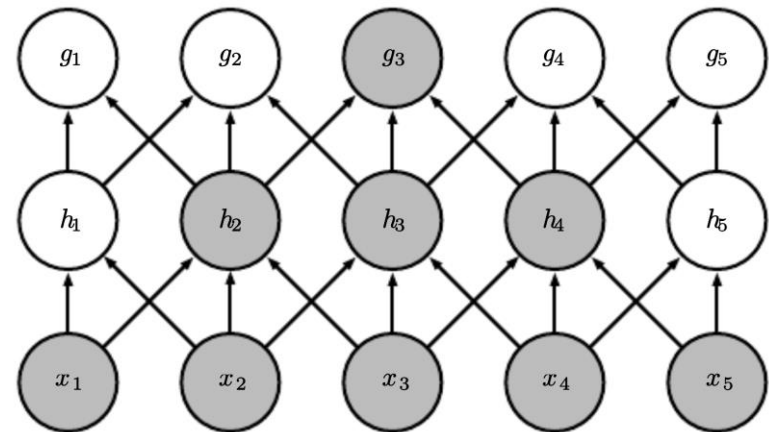
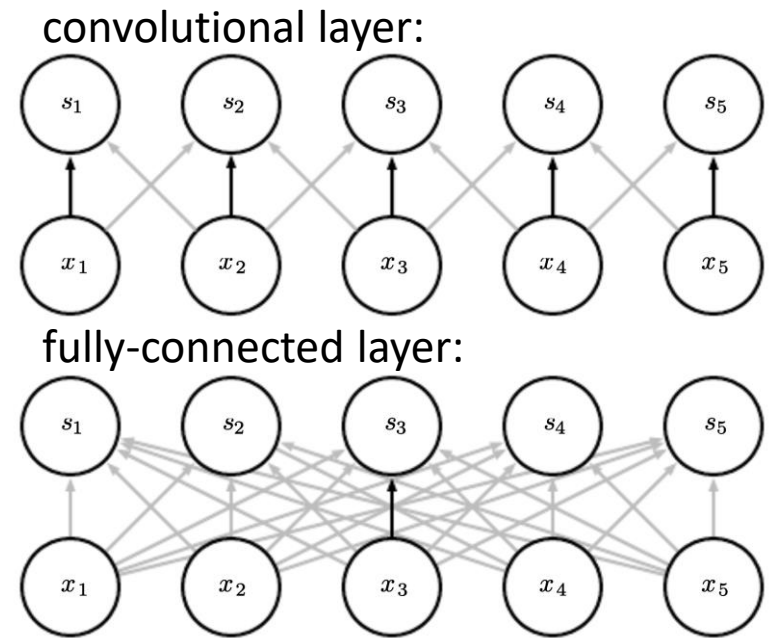


# Regularization Effects

- sparse interactions: much less weights
- parameter sharing: use same weights for different connections

effect of receptive field over several layers:

- consider only locally restricted number of input values from previous layer
  - grows for earlier layers (indirect interactions)
- hierarchical patterns from simple building blocks (many aspects of nature hierarchical)





# Another Ingredient: Pooling

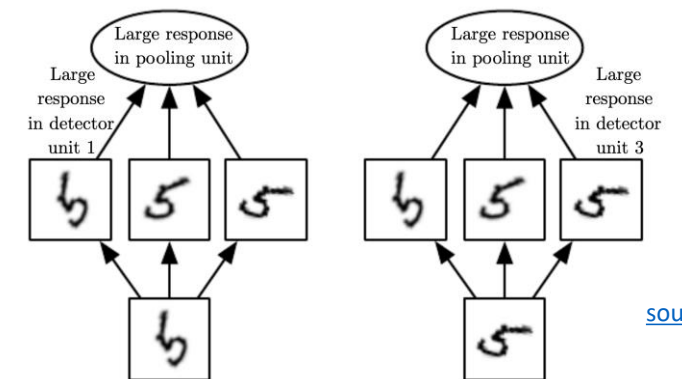
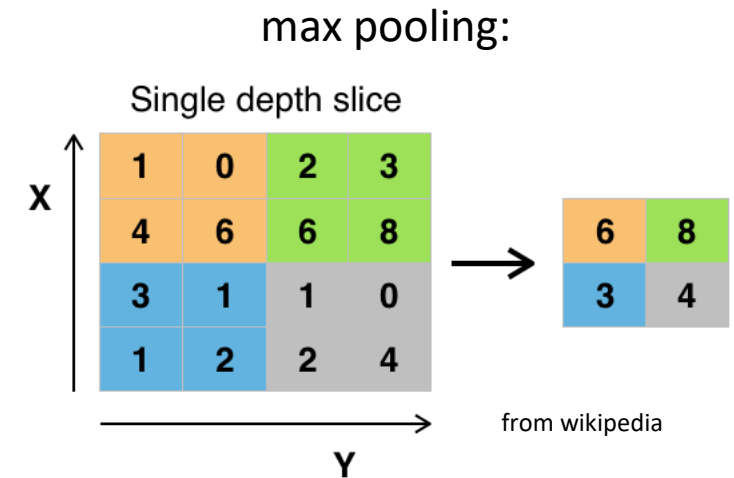
replacing outputs of neighboring nodes with summary statistic (e.g., maximum or average value of nodes)

→ non-linear down-sampling (regularization)

pooling is translation invariant: no interest in exact position of, e.g., maximum value

pooling over features learned by separate kernels (cross-channel pooling) can also learn other transformation invariances, like rotation or scale

(convolutions can detect same translated motif across entire image, but not rotated or scaled versions of it)

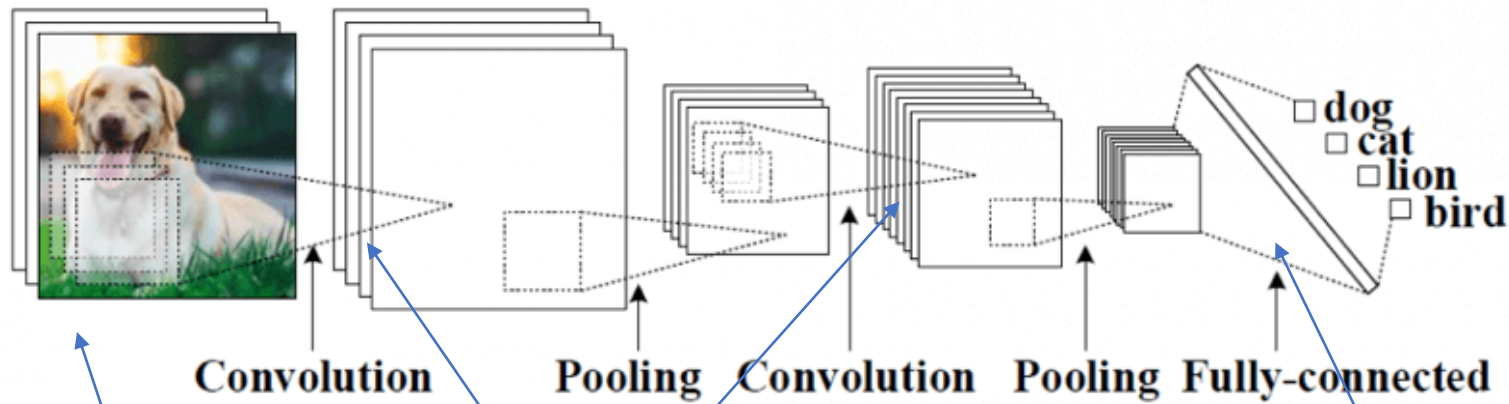


# Putting It All Together

CNN in short:

local connections, shared weights, pooling, many layers

kernels

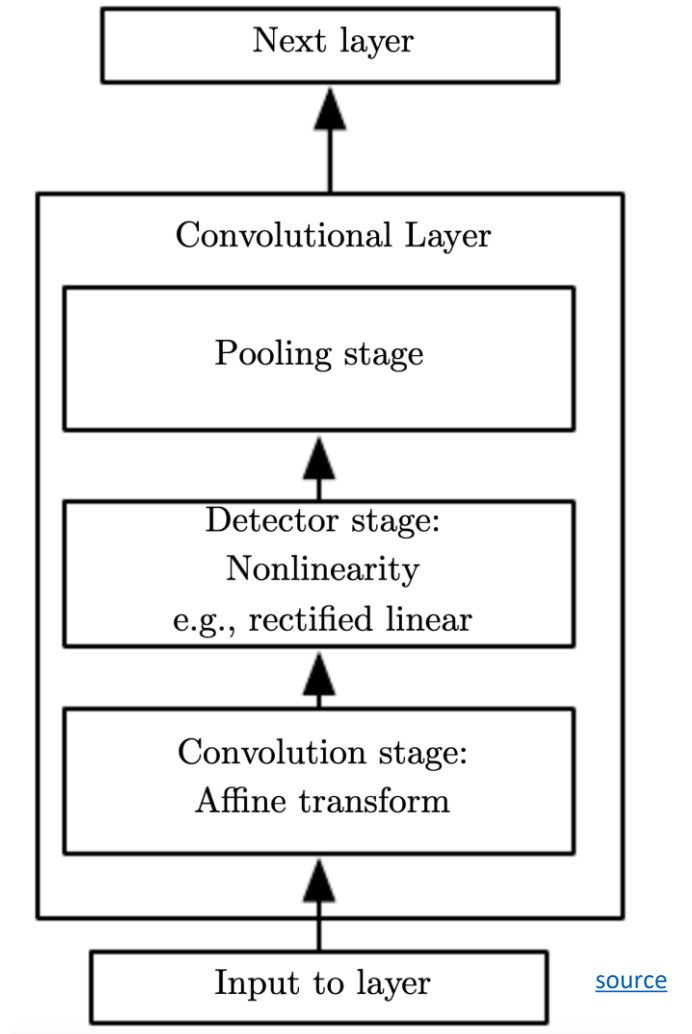


many images  
(training examples),  
potentially with  
several channels

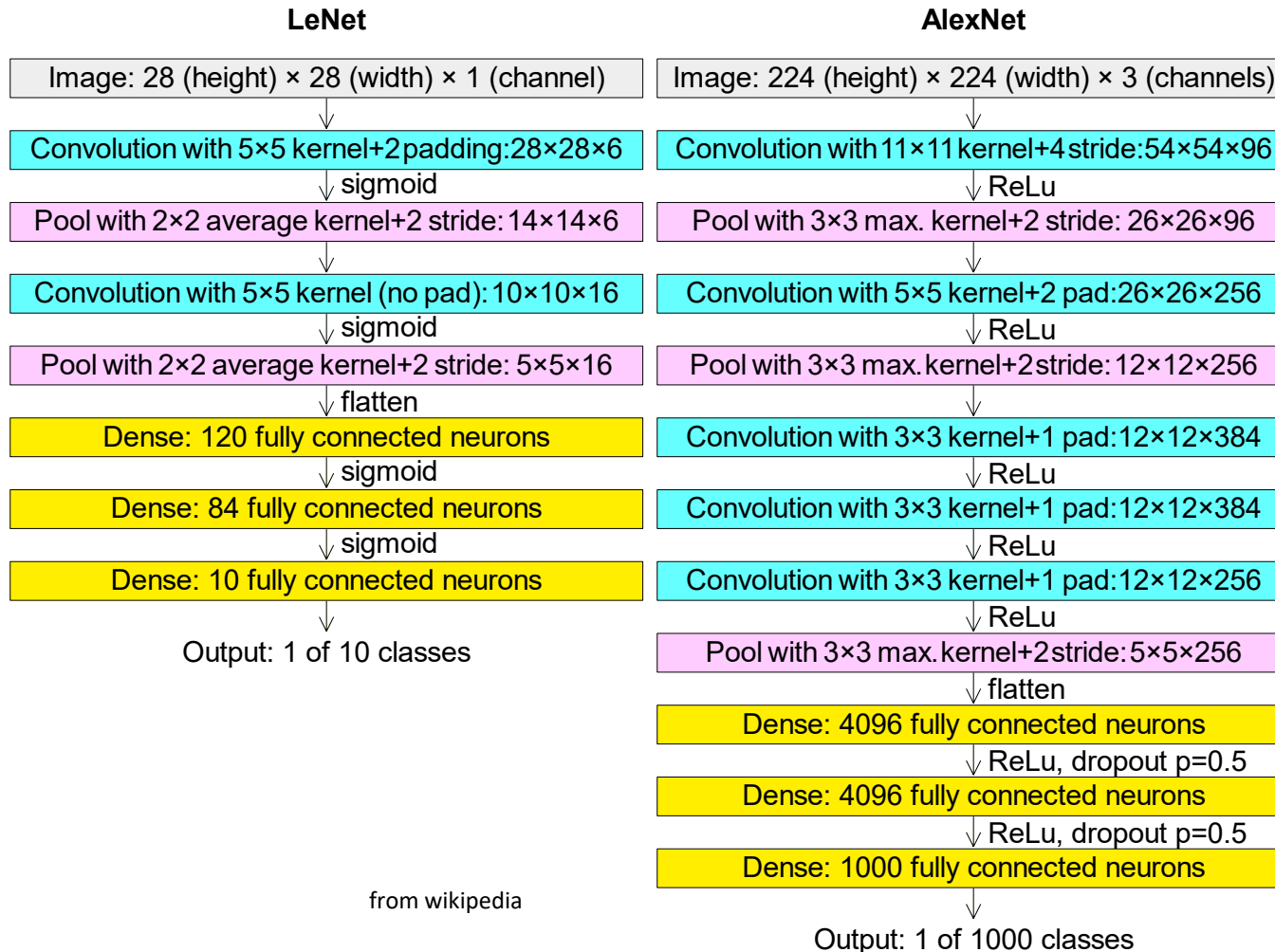
several kernels  
producing several  
feature maps

down-sampling  
by convolutions  
and pooling

flatten dimensions  
for final classification  
or regression



# Going Deeper



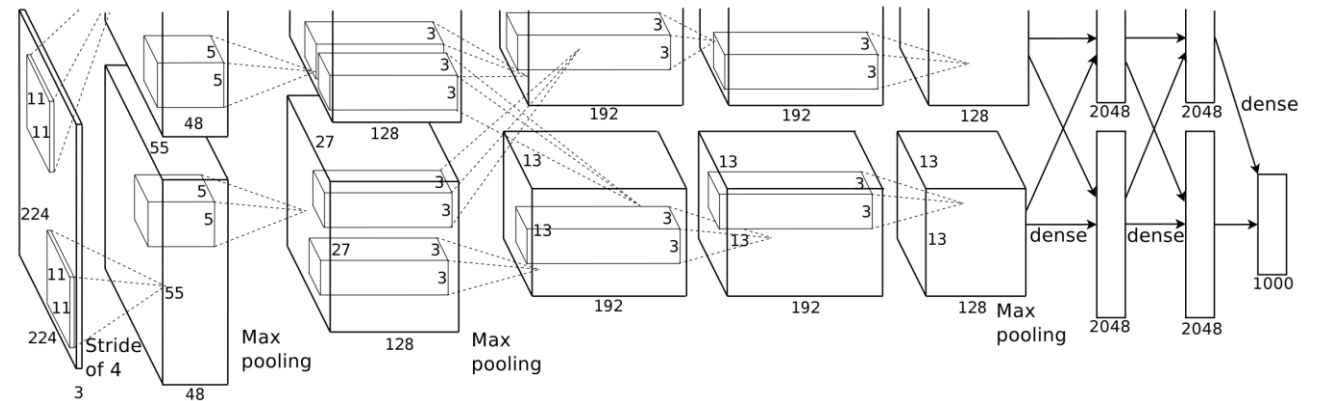
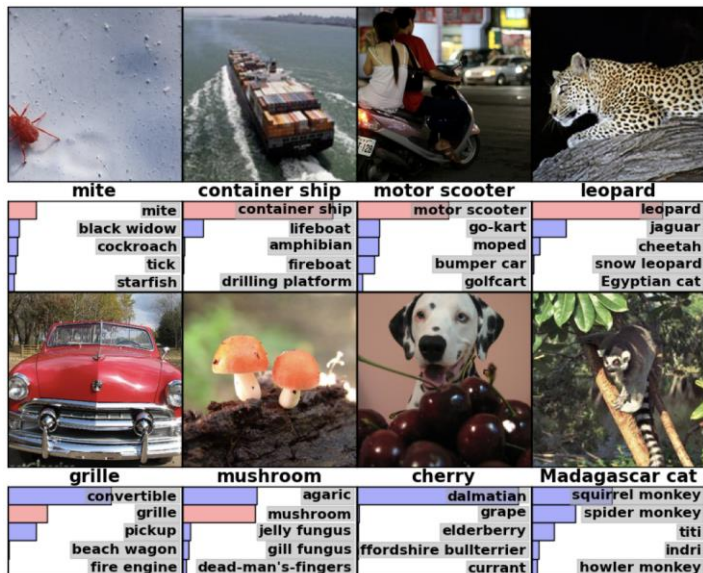
AlexNet finally started the deep learning hype.  
(winning the ImageNet challenge in 2012)

# Rise of Deep Learning

a little bit oversimplified:

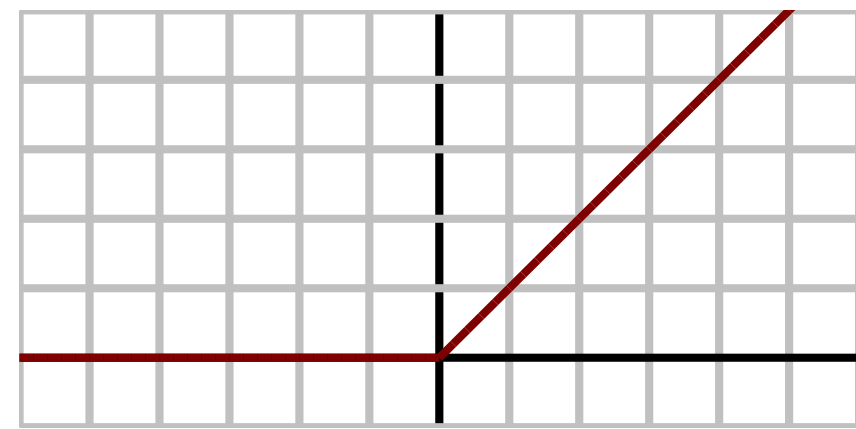
deep learning = lots of training data + parallel computation + smart algorithms

AlexNet: ImageNet (with data augmentation) + GPUs + ReLU, dropout, SGD



[source](#)

# Rectified Linear Unit (ReLU)



reminder: activation function non-linear transformation of summed weighted input of a node (linear), output to be used as input for nodes of subsequent layer

needs to be differentiable for back-propagation (ReLU at 0 no issue, just set to 0 or 1)

neural network model with [ReLU activation](#) can be interpreted as exponential number of linear models that share parameters

main advantages (leading to enablement of deeper networks by better optimization):

- unlike sigmoid or tanh (predominantly used before) activation, no issue with vanishing gradients from saturation effects
- very efficient computation: constant gradients of 0 and 1 below and above input of zero
- sparse activation: many hidden nodes deactivated (output 0) → information disentangling

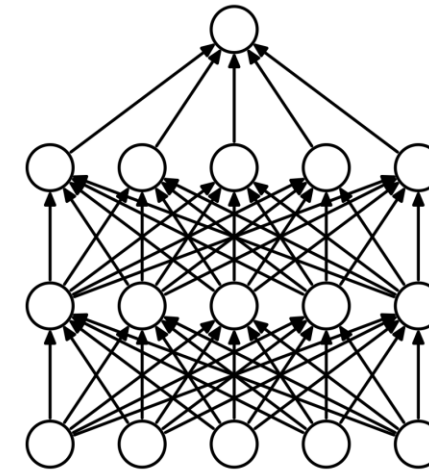
# Dropout in Neural Networks

goal: prevent overfitting of large neural networks

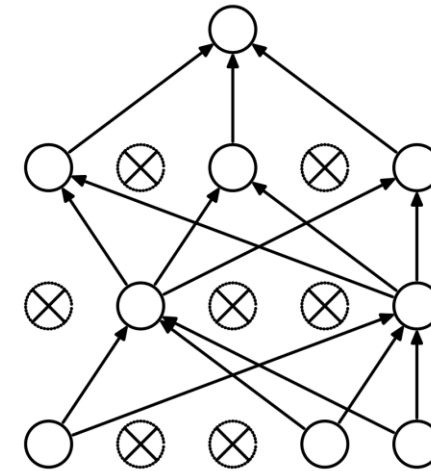
idea: randomly drop non-output nodes (along with their connections) during training (not prediction)

→ adaptability: regularizing each hidden node to perform well regardless of which other hidden nodes are in the model

- for each mini-batch, randomly sample independent binary masks for the nodes
- much less computation than bagging (training many different neural networks)
- destroying extracted features rather than input values



(a) Standard Neural Net



(b) After applying dropout.

[source](#)

dropout for 2D structures, such as images, usually drops entire channels instead of individual nodes (because locality nullifies the effect of standard dropout → neighboring nodes step in)

# Inductive Bias (aka Learning Bias)

set of assumptions that a learning algorithm uses to predict outputs of inputs that it has not encountered during training

examples: linear response (linear regression), maximum margin (SVM), nearest neighbors (kNN), spatial structure (CNN)

but also: different regularization and optimization methods

crucial piece of generalization

data in disguise (replacement for missing information on specific situations in limited training data sets)

# No Free Lunch Theorem

All optimization/ML algorithms (both sophisticated and simple ones) perform equally well when their performance is averaged across all possible problems.

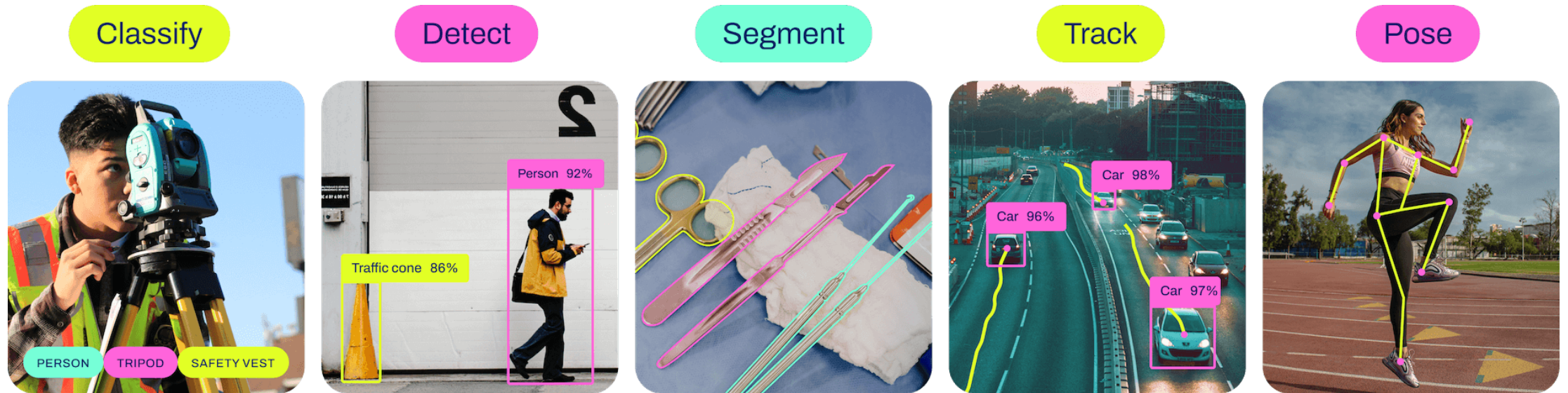
(But deep learning is trying to solve many problems with very general-purpose forms of regularization.)

Particularly, model complexity doesn't reflect if inductive bias is appropriate for problem at hand.

→ choose right ML method for learning task at hand



# Object Detection and Image Segmentation



[source](#)

coding example: object detection with [YOLO](#) and subsequent image segmentation with Segment Anything Model ([SAM](#))

# Assignments

- handwritten digit classification: [Kaggle Digit Recognizer](#)  
(build PyTorch CNN)
- image classification: [Kaggle Intel Image Classification](#)  
(fine-tune a [torchvision](#) model, e.g., AlexNet)