Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava

Neural Networks

Lecture 10

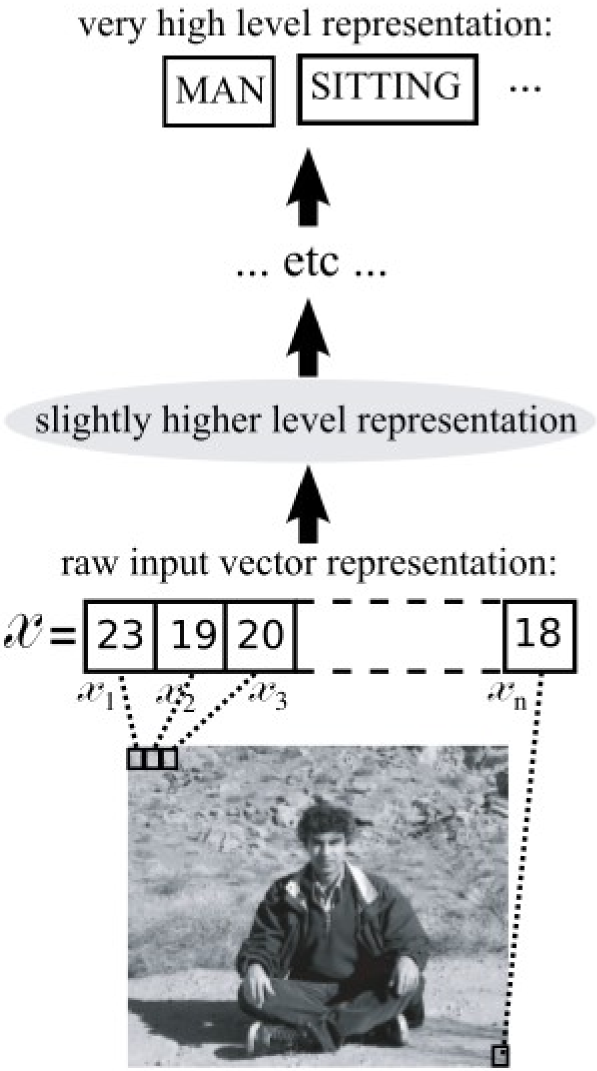
Deep learning and convolutional nets

Igor Farkaš

2024

Deep learning

- multi-layer architectures (>2 hidden layers)
- increasing abstractness
- with distributed representations emerging
- current discussion (connectionist ML) about the origins of DL
- Breakthrough: Deep Belief Networks (Hinton, Osindero &Teh, 2006)
- unsupervised + supervised learning possible
- biological relevance
- Currently top results in various large-data domains: vision (object recognition), language tasks, speech, games



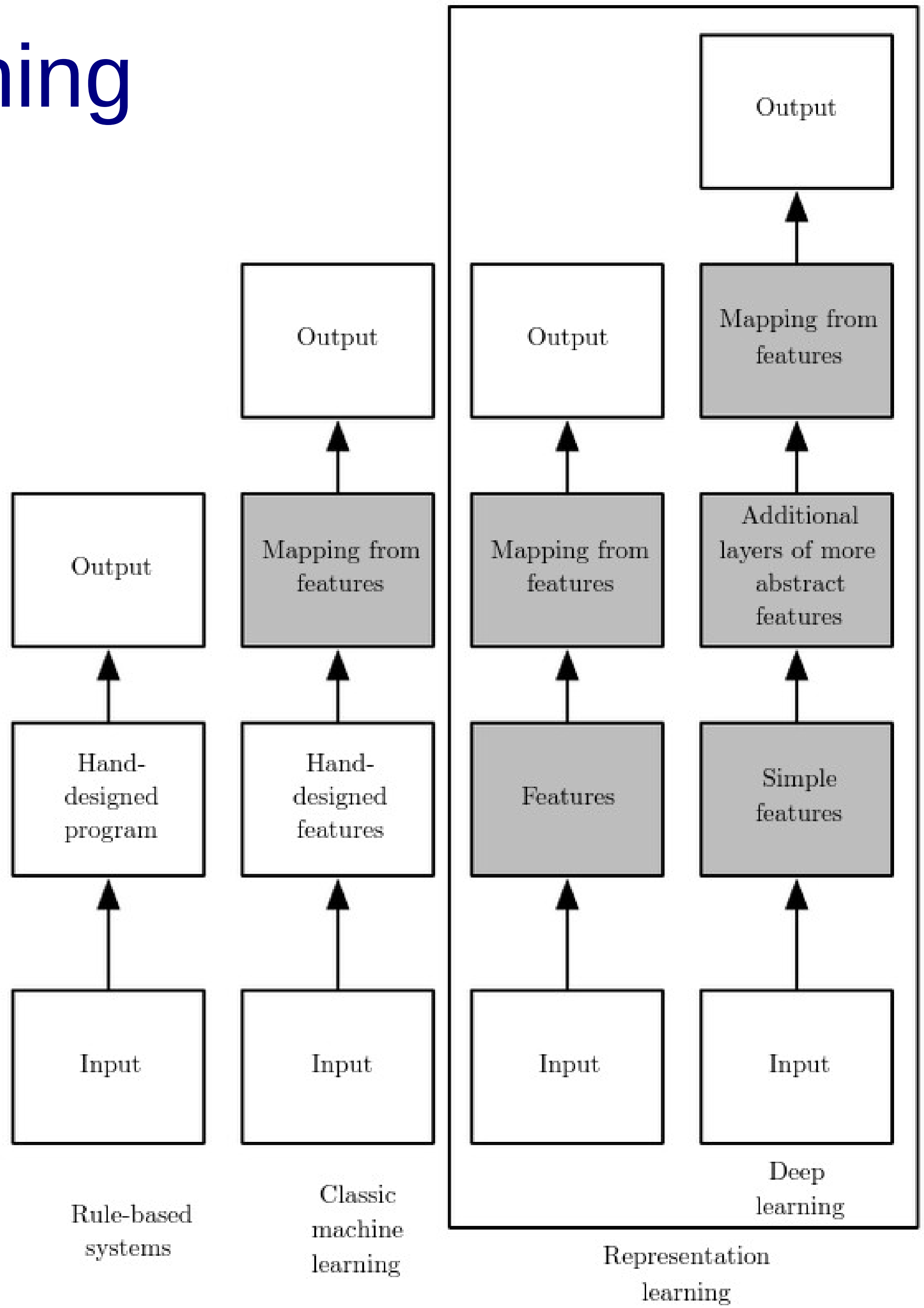
(Bengio, 2009)

Advantage of DNNs

- e.g. NN with 2 hidden layers: $y = f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$
- Depth of the model = #hidden layers, width = #neurons
- DNN = extension of linear models (which cannot capture interactions b/w any two input variables (e.g. XOR)
- depth more effective than width:
- universal approximation theorem: The hidden layer may be infeasibly large (growing exponentially with input dimension) and may fail to learn and generalize correctly.
- In many cases, deeper models can reduce the number of units required to represent the desired function and improve generalization.
- DNN = representation learning

Representation learning

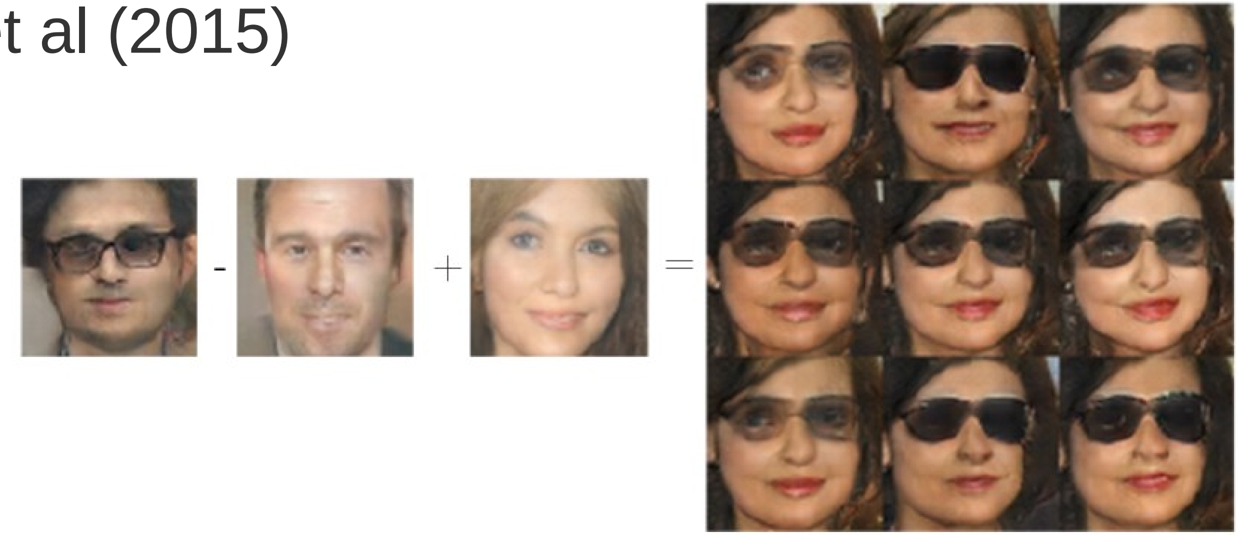
- DL – towards end-to-end learning
- Type of representation matters
- distributed reps have advantages
- greedy unsupervised layer-wise pretraining – often used at onset of DNNs (from 2006)
  - successful, but not inevitable for DNNs
  - before: only CNNs



(Goodfellow et al, 2015)

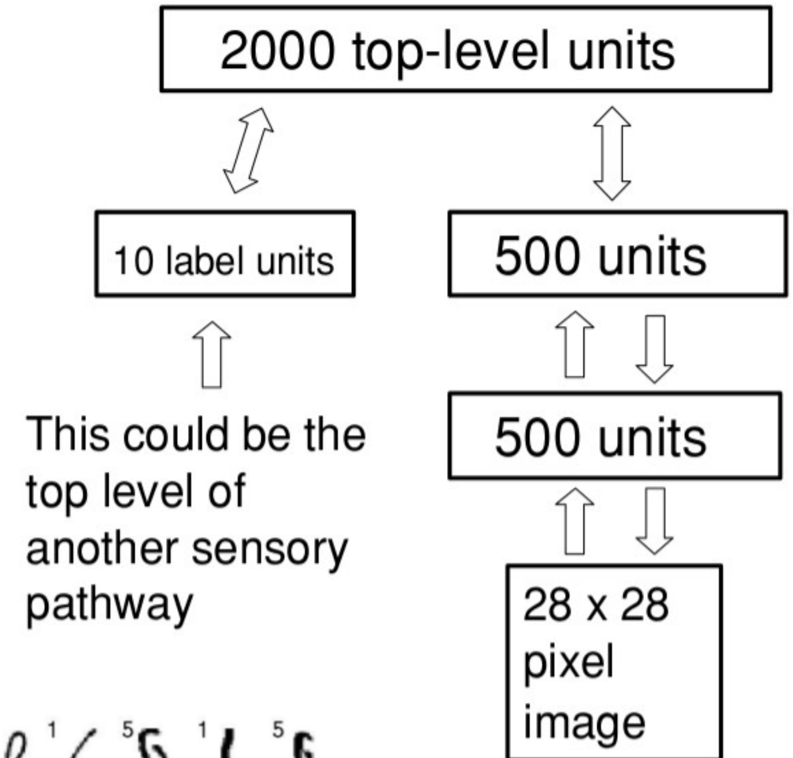
Distributed representations

- Powerful, since they are efficient and support generalization (unlike localist = symbolic representations)
- Distributed representations are combinatorially very powerful because they can use *n* features with *k* values to describe k^n different concepts.
- Redford et al (2015): a generative model can learn a distributed representation that disentangles two sources of variation:
- see Goodfellow et al (2015) for more details



Benchmark tests – example

- MNIST database – handwritten digits
- Deep Belief Network (Hinton et al, 2006): layer-wise unsupervised pretraining + learning joint distributions (image-label pairs)
  - also top-down weights, symmetric weight matrices, 1.25% errors



(Hinton et al, 2006)

errors made by DBN

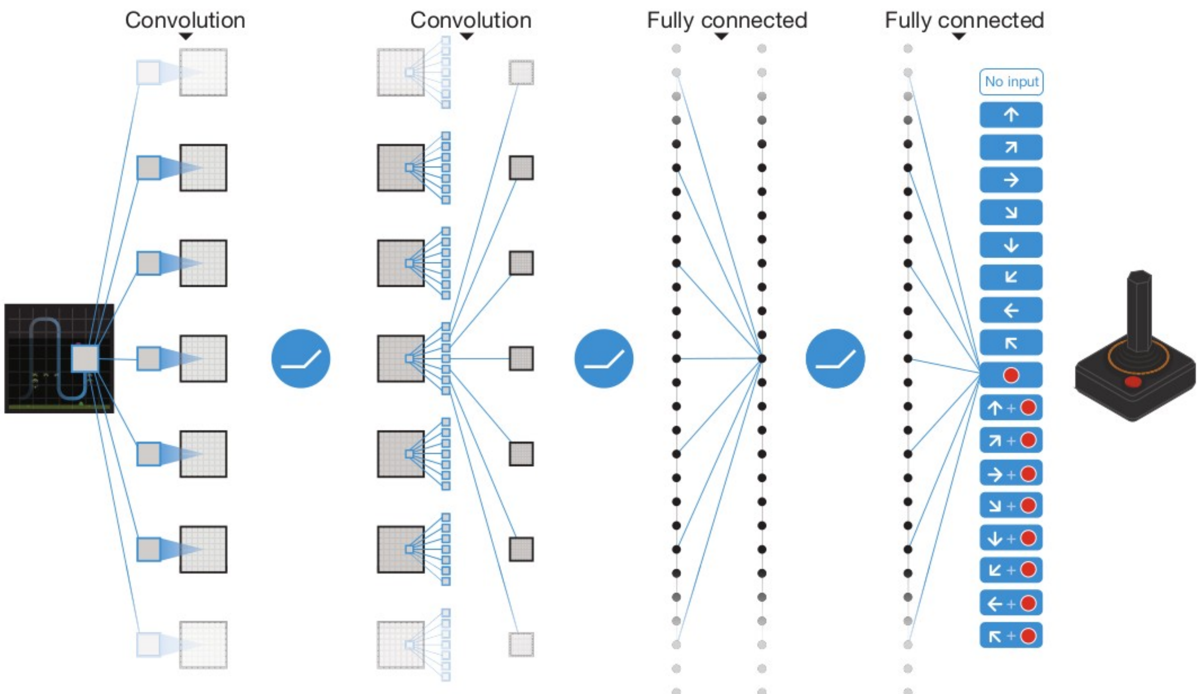
Success of convolutional DNN in vision

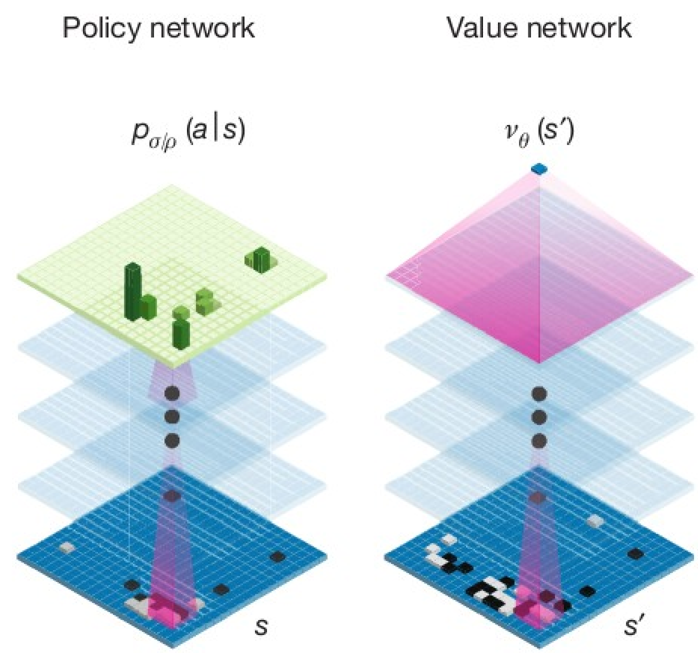
- MNIST handwritten digits using CNN (LeCun et al, 1998)
  - testing error <1%
- MNIST (Cireşan, Meier, and Schmidhuber, 2012),
  - near-human performance (0.23%)
  - committee of 35 deep convolutional networks
- German Traffic Signs (Cireşan, Meier, Masci, et al., 2012)
  - super-human performance (0.54% vs ~1%)

(Deep Mind‘s) success of DNN in games

- Convolutional (deep Q) NN (Mnih et al, 2015) – learns to win Atari games from *raw pixel* data (i.e. end-to-end)

- AlphaGo beats Lee Se-dol (Silver et al, 2016)
  - RL-based deep NN combined with tree search



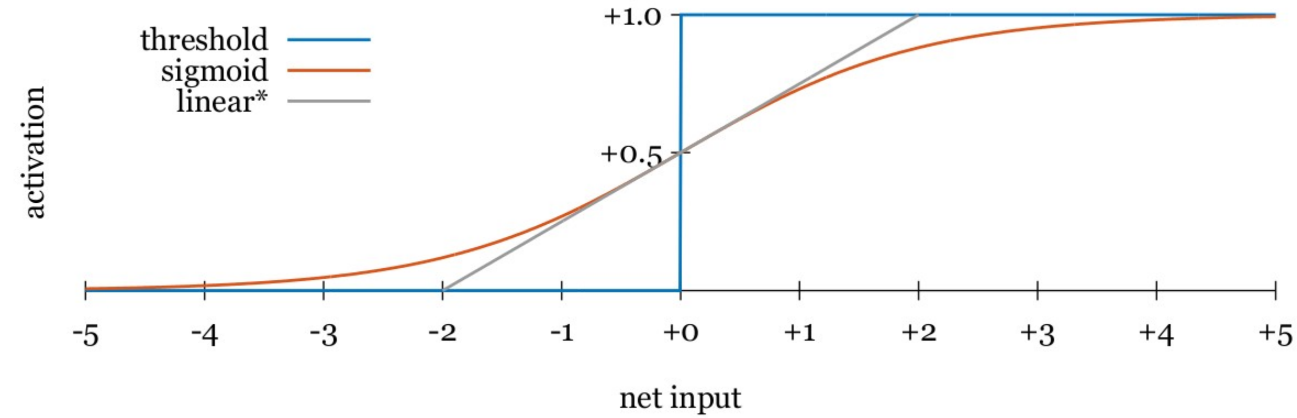


Steps to improve deep learning

- initialization
  - weights: uniform or gaussian distribution (naïve)
  - problem of weight saturation
  - unsupervised pretraining
- new activation functions
  - help avoid gradient vanishing problem
- regularization
  - improves generalization
- training
  - using improved versions of gradient descent
  - pretraining (in various models)

Evolution of activation functions

- Unipolar: logic threshold, logistic sigmoid, (thresholded) linear

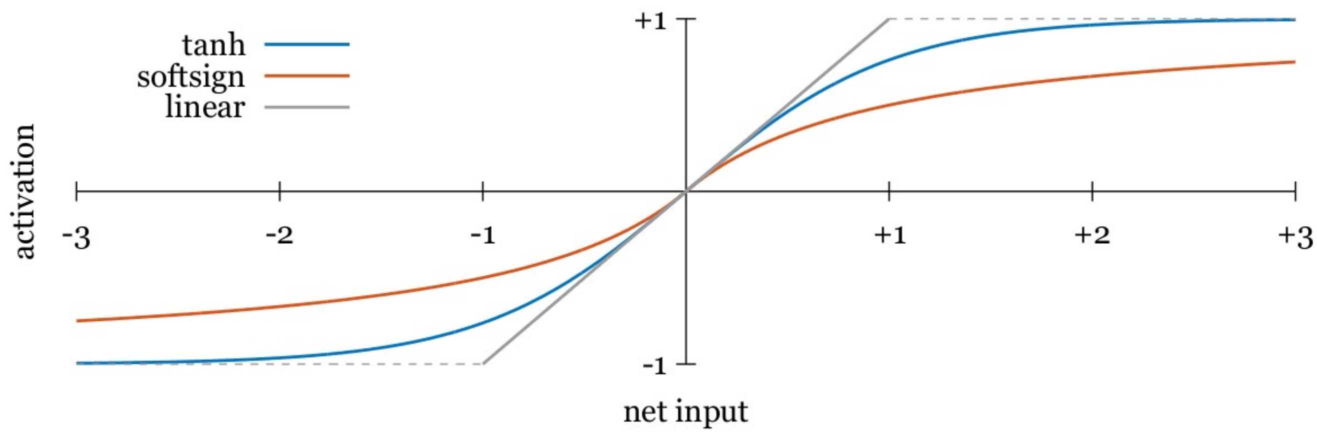


- Bipolar: softsign (Glorot and Bengio, 2010)

(Kuzma, 2016)

$$\tanh(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}$$

$$softsign(net) = \frac{net}{1 + |net|}$$



Activation functions – rectifiers

- asymmetric, with preserved nonlinearity
- introduced to prevent saturation problems
- ReLU – rectified linear unit

Leaky ReLU (Maass et al, 2015)

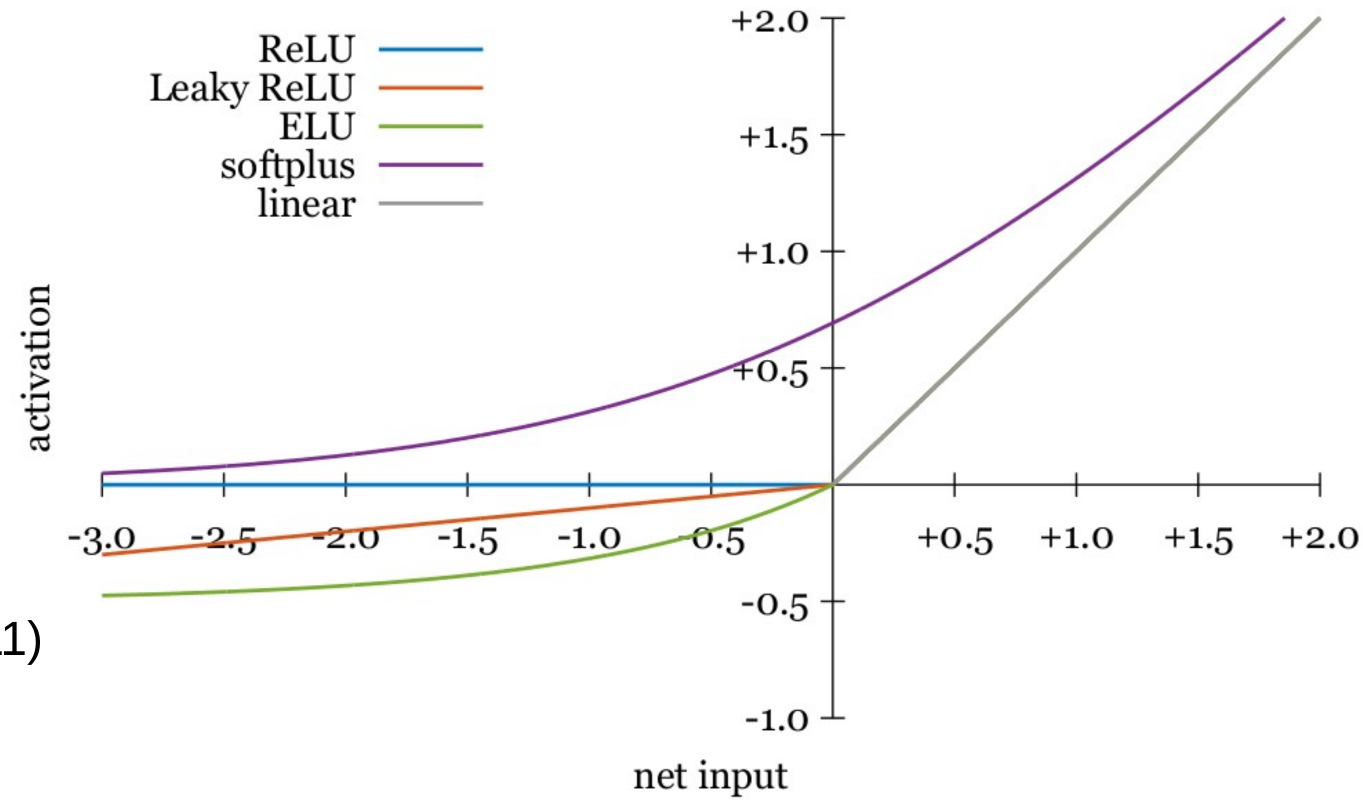
$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

Exponential ReLU
(Clevert et al, 2015)

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

Softplus (Glorot et al, 2011)

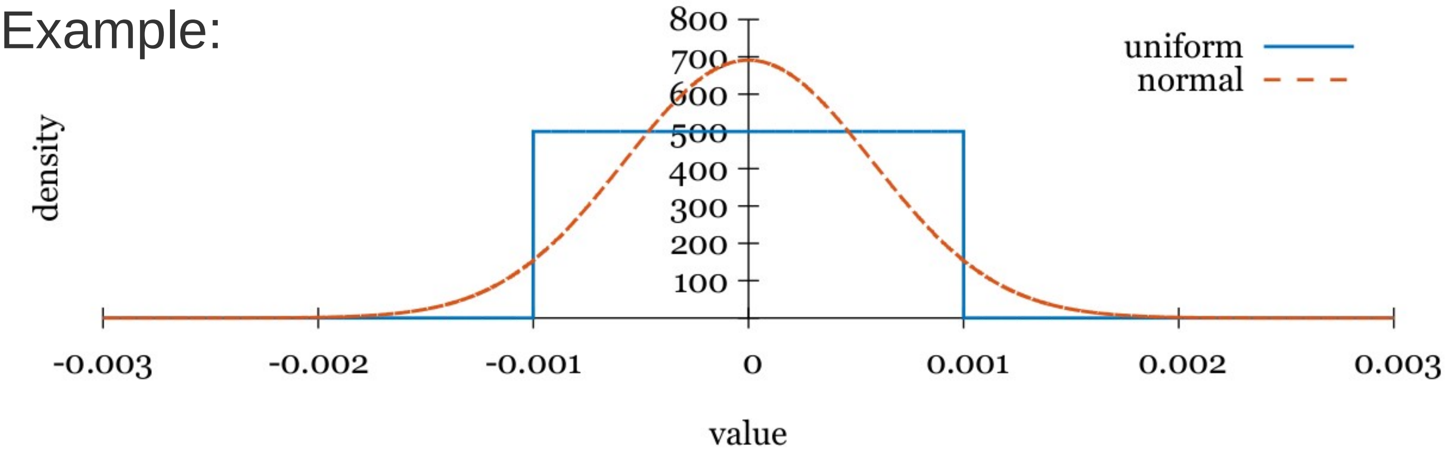
$$f(x) = \ln(1 + e^x)$$



Weight initialization

- Default – small random numbers, *Uniform*(-*m*,+*m*), *Normal*(0,*s*²)

Example:



$$Uni(-.0001, +0.001)$$

$$N(0, \sqrt{(1/3)} * 0.001)$$

Both distributions have the same mean and variance

- Normalized initialization – depends on network architecture:

$$\mathbf{W} \sim \mathrm{Uni}\left(\pm \frac{1}{\sqrt{\deg_{in}}}\right)$$

(Bradley, 2009)

$$\mathbf{W} \sim \mathrm{Uni}\left(\pm \sqrt{\frac{6}{\deg_{in} + \deg_{on}}}\right)$$

(Glorot & Bengio, 2010)

$$\mathbf{W} \sim \mathrm{N}\left(0, \sqrt{\frac{2}{\deg_{in}}}\right)$$

(He et al, 2015)

Convolutional networks

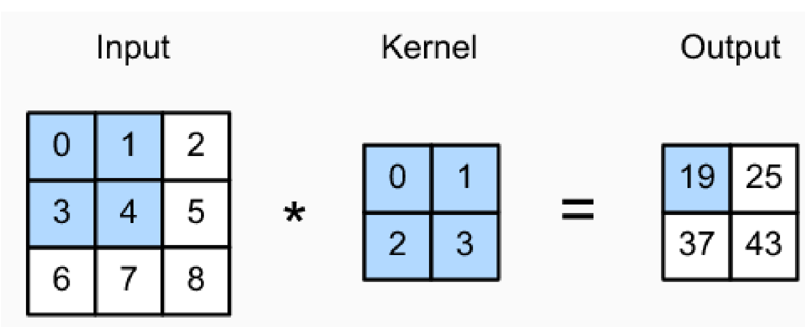
- a specialized kind of NN for processing data that has a known grid-like topology (1D, 2D, …)
- use a specialized kind of linear operation – convolution – in place of general matrix multiplication in at least one layer
- Convolution combines input x with (flipped) kernel w
- 1D: $s(t) = (x * w)(t) = \Sigma_a \, x(a) \, . \, w(t-a)$
- 2D: $S(i, j) = (I * K)(i, j) = \Sigma_m \Sigma_n I(m,n) \, . \, K(i-m, j-n)$

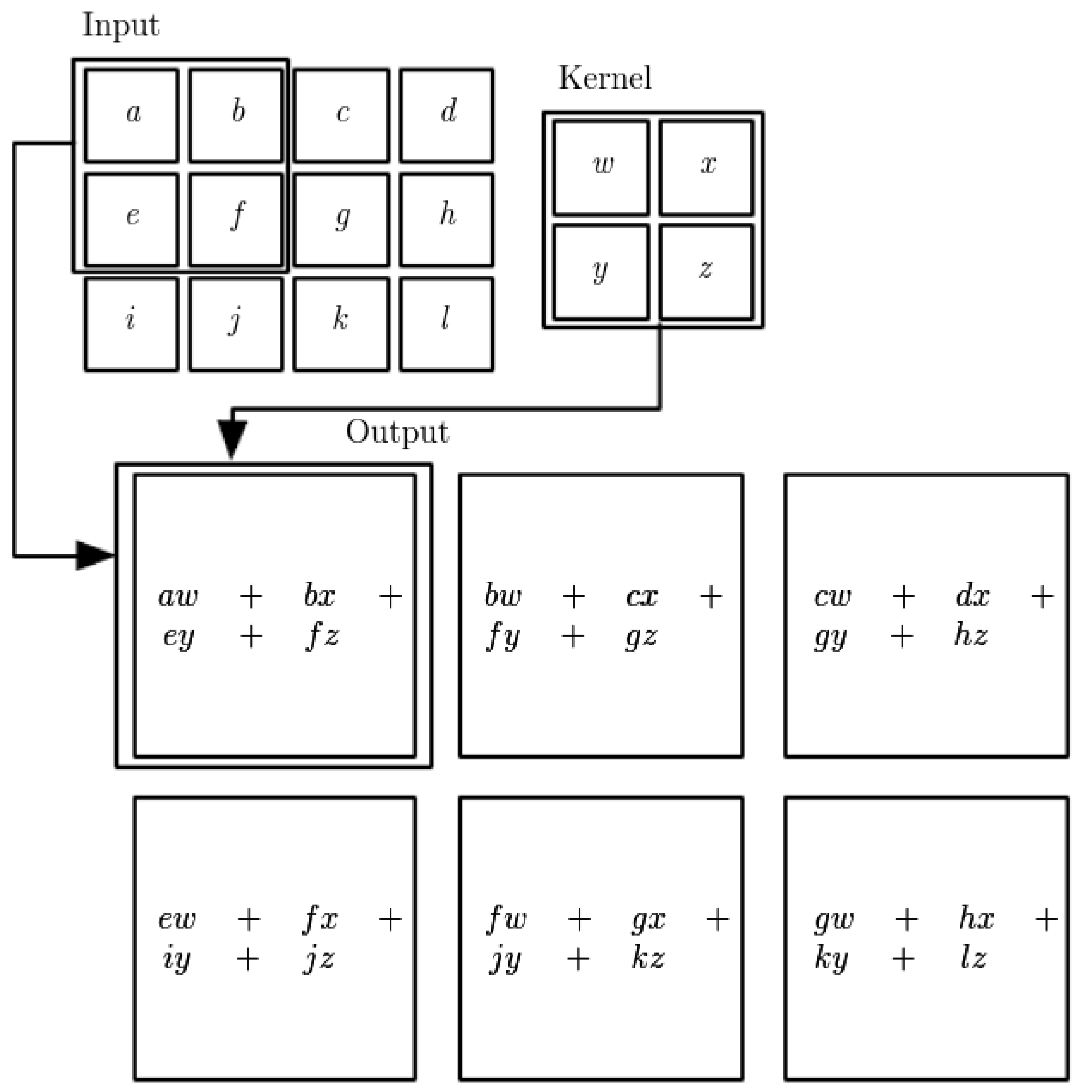
- Convolution is commutative = $S(i, j) = \Sigma_m \Sigma_n I(i-m, j-n) \, . \, K(m,n)$
- Cross-correlation: $S(i, j) = (I \circ K)(i, j) = \Sigma_m \Sigma_n I(i+m, j+n) \, . \, K(m,n)$

Example of a 2D convolution

- Kernel ($k_h \times k_w$) is usually much smaller than the input image ($n_h \times n_w$)
- Kernel is restricted to lie completely in the image
- Example on the right: Image shrinks from 3×4 to 2×3
- In general, the output size is $(n_h - k_h + 1) \times (n_w - k_w + 1)$.

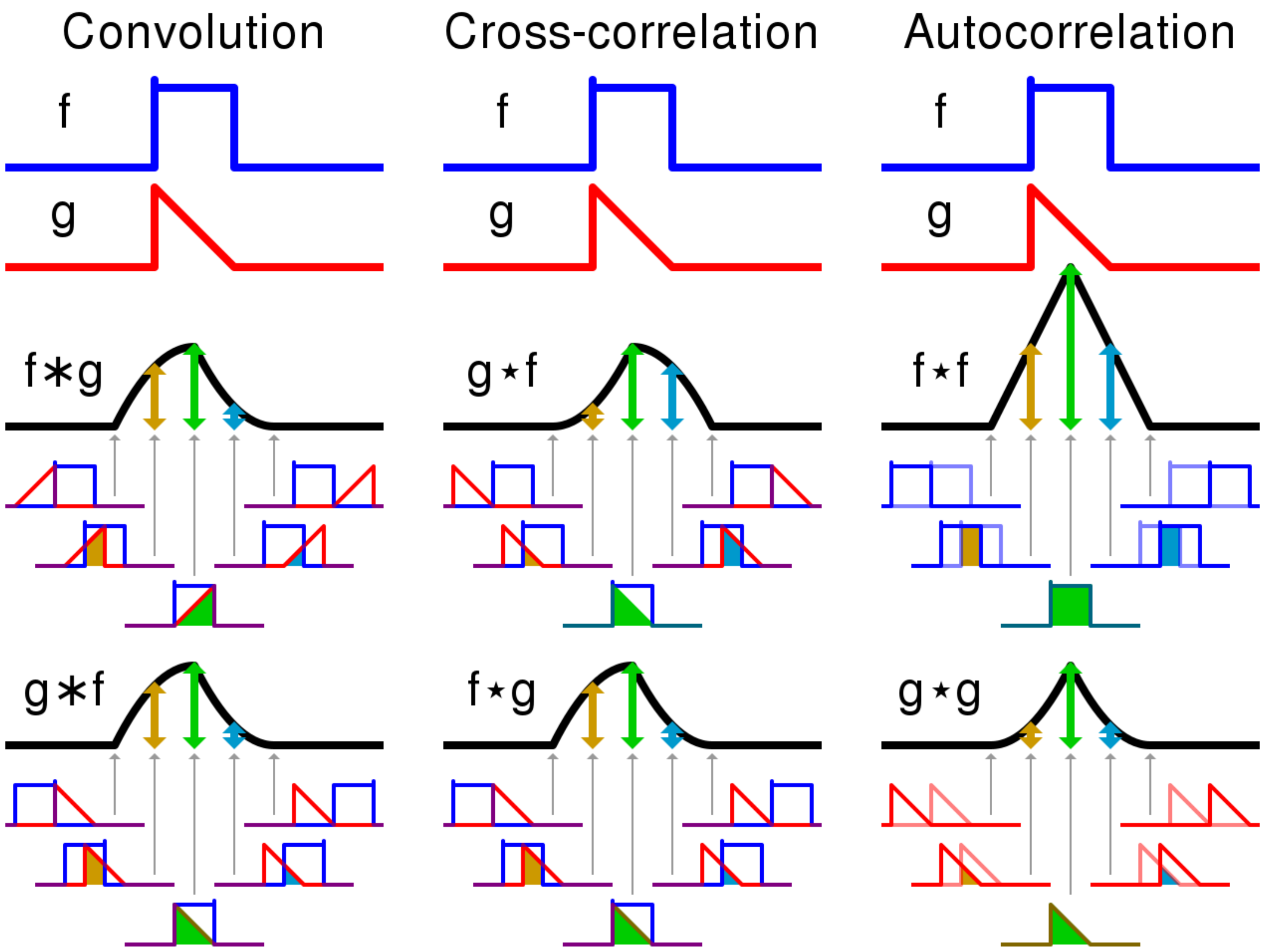
Another example with numbers:





(Goodfellow et al., 2015)

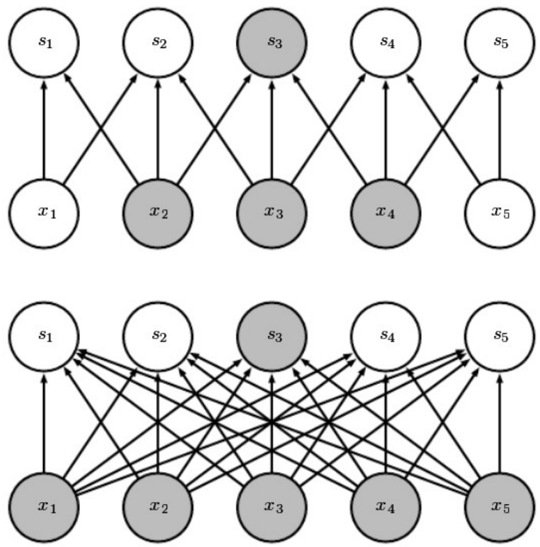
Graphical comparison in 1D



https://en.wikipedia.org/wiki/Convolution

Three advantages of convolution

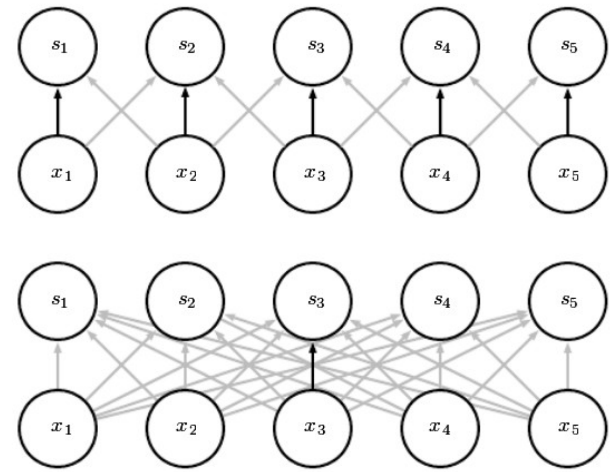
Sparse interactions (weights)



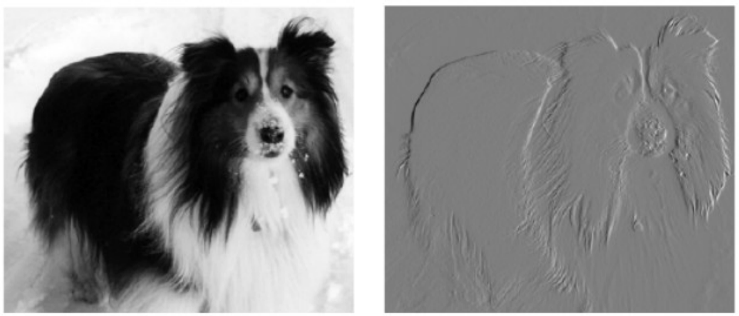
YES

NO

Parameter sharing



Example of edge detection:



Equivariant representations

– only to translation, other forms of equivariance (scale, rotation) require additional mechanisms.

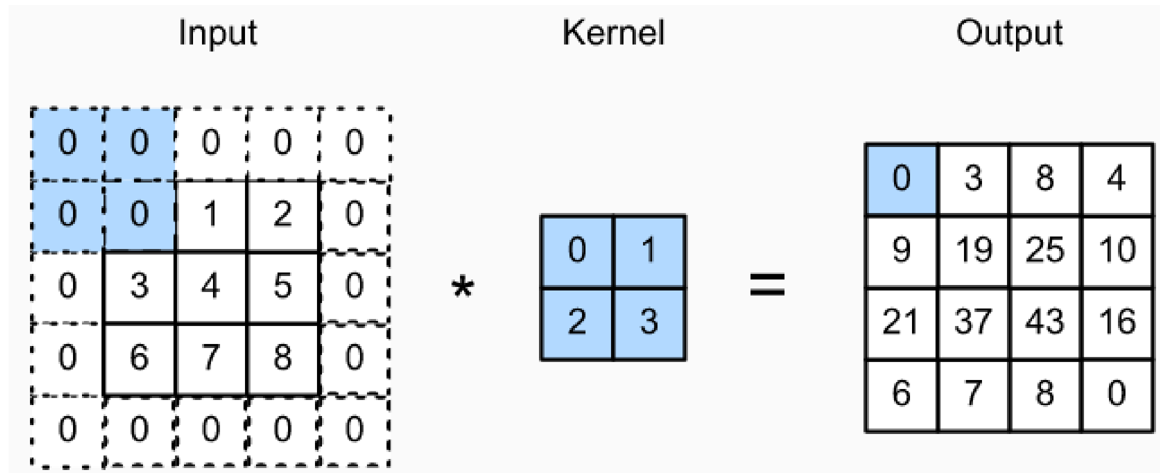
(Goodfellow et al., 2015)

Padding and stride

- Optional operations, sometimes useful to control the size of the output

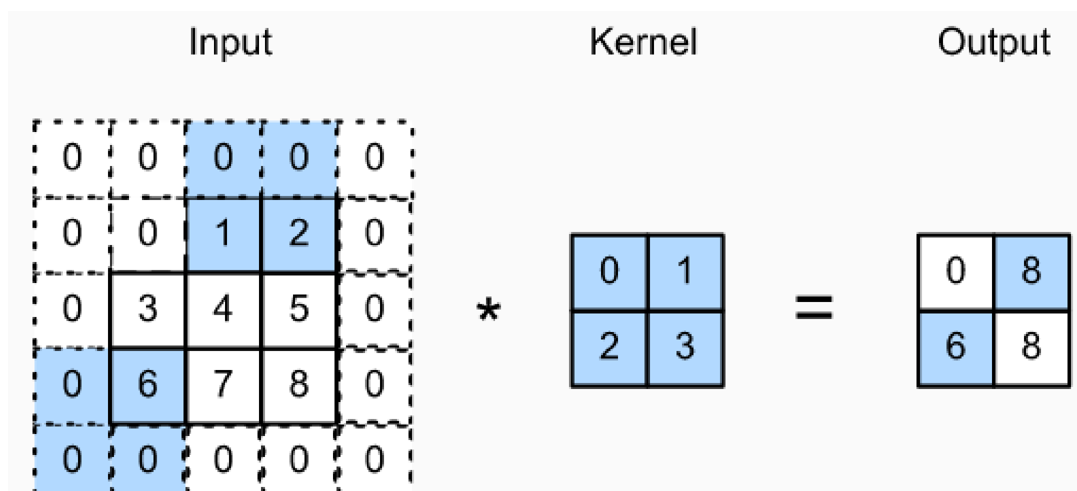
Padding – slows down image shrinking over layers

- zeros adding around the image



Stride – speeds up image size shrinking

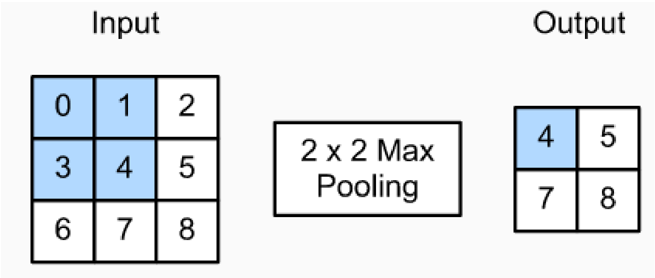
- determines the size of traversing steps over the image



(Zhang et al, 2020)

Pooling

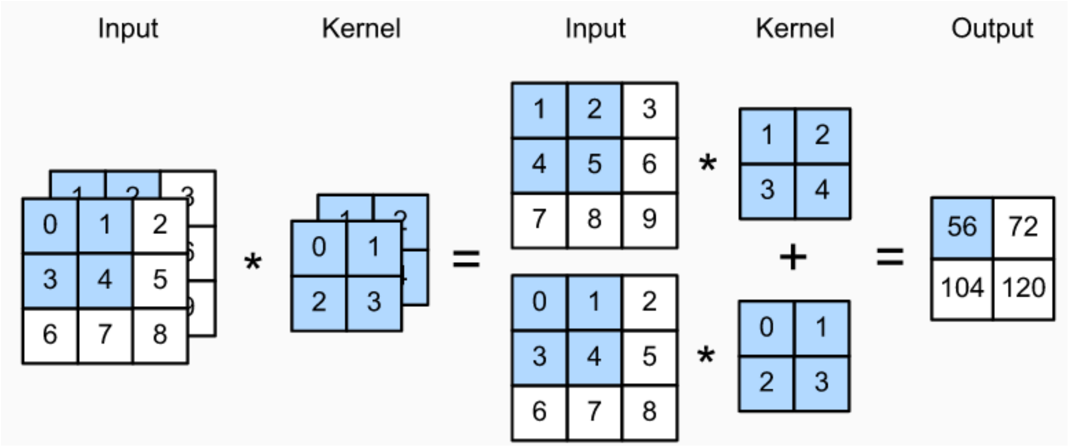
- used to gradually reduce the spatial resolution of hidden representations
- higher layers have larger receptive fields of each hidden node
- hence, we get gradual aggregation of information, yielding coarser and coarser maps, leading to global representation
- maximum pooling and average pooling
- pooling layers can also change the output shape...
- …since padding and stride can be applied



Multiple input and output channels

Multiple input channels

- a separate kernel for each
- example with 2 input ch's and one output channel

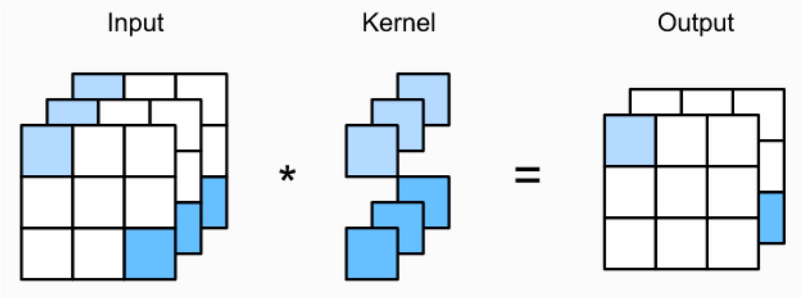


Multiple output channels

- are needed if we want to propagates channel across layers

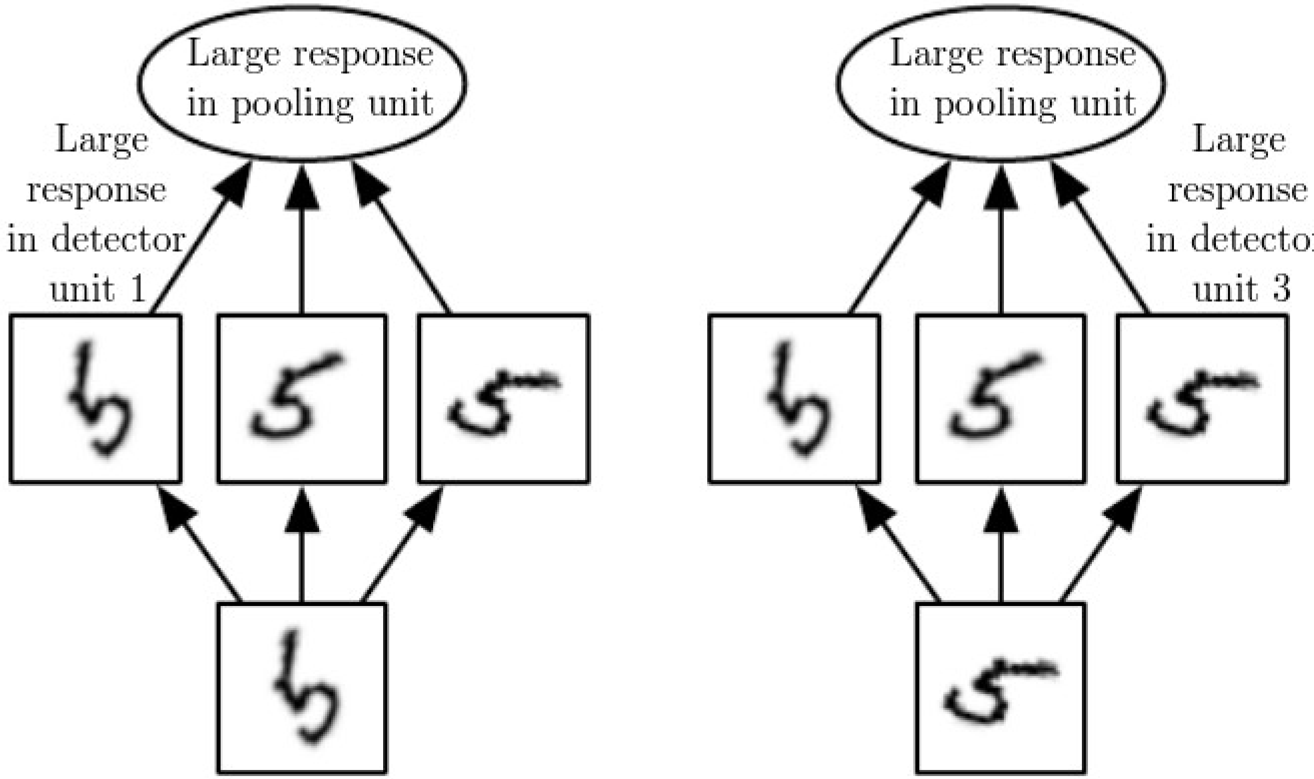
1×1 convolution kernel

- still makes sense, since the weights are tied (shared) across pixel location.



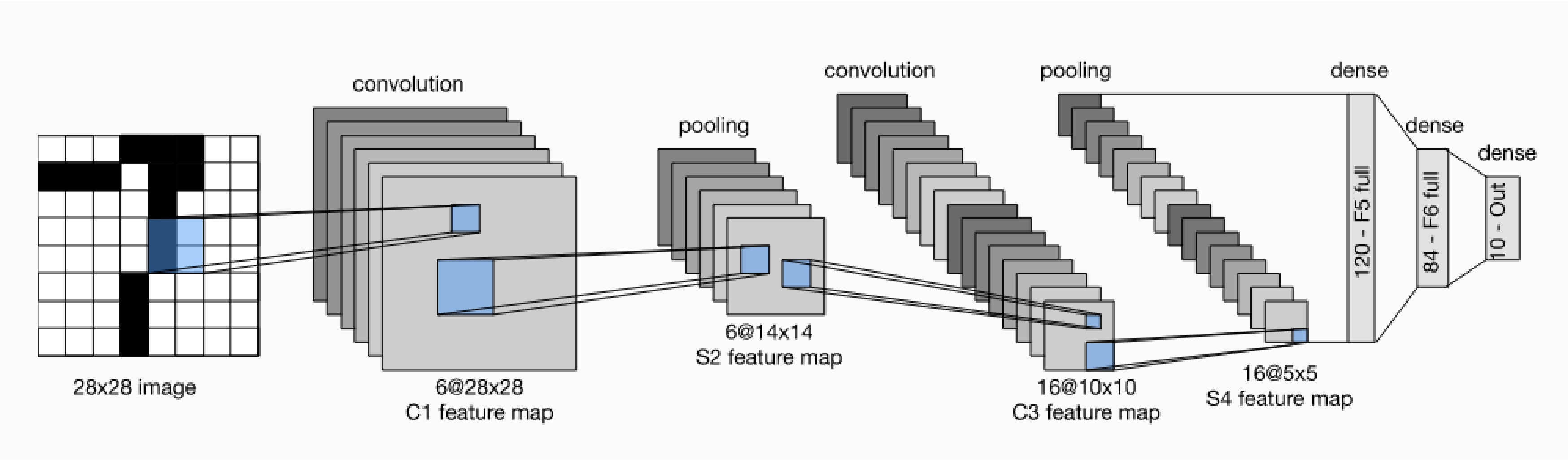
Examples of learned invariances

- A pooling unit spans over multiple features that are learned with separate parameters
- A pooling unit can learn to be invariant to transformations of the input (rotations)



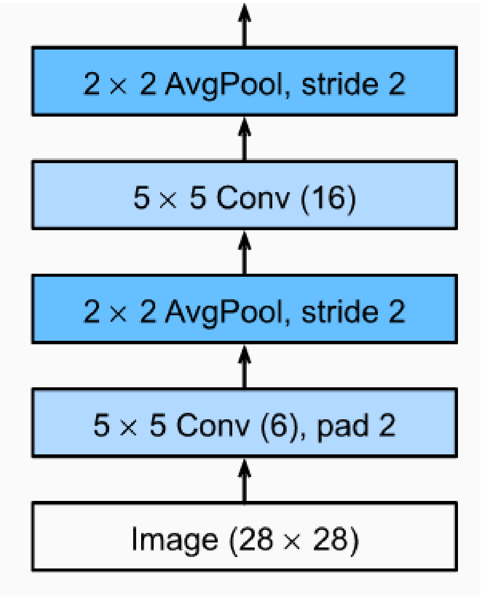
(Goodfellow et al., 2015)

LeNet – the 1st CNN



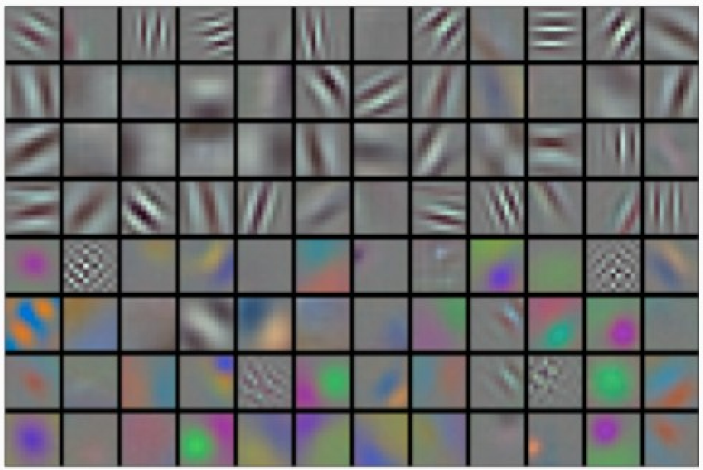
(LeCun et al., 1998)

- First CNN, used for computer vision tasks
  - still used for some ATMs
- 2 parts: convolutional layers + FC layers
- sigmoids used (ReLUs not known yet)

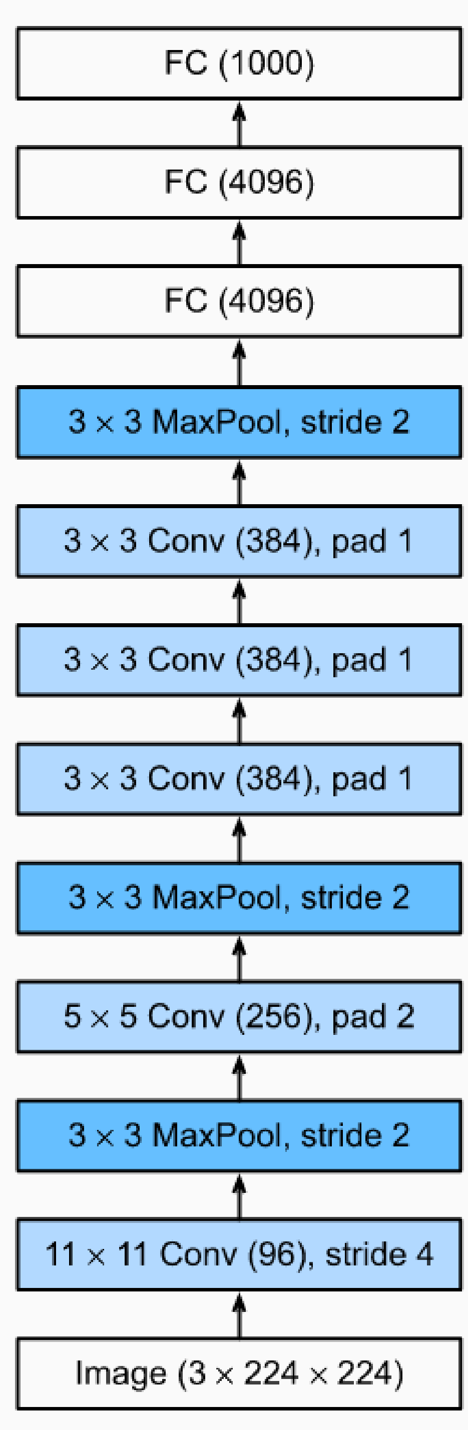


AlexNet

- winner of LSVRC-2012 competition, 1.3 mil. images (ImageNet), 1000 classes, ~62 mil. param.
- uses ReLU which yields 6x speed at the same accuracy, dropout, pooling to reduce network.
- Training: 90 epochs in 5 days, on two GTX 580 GPUs, SGD with LR 0.01 (decreased 3-times), momentum 0.9 and weight decay 0.0005.
- Original model used a dual data stream design (due to memory limitations)



learned features at the first hidden layer

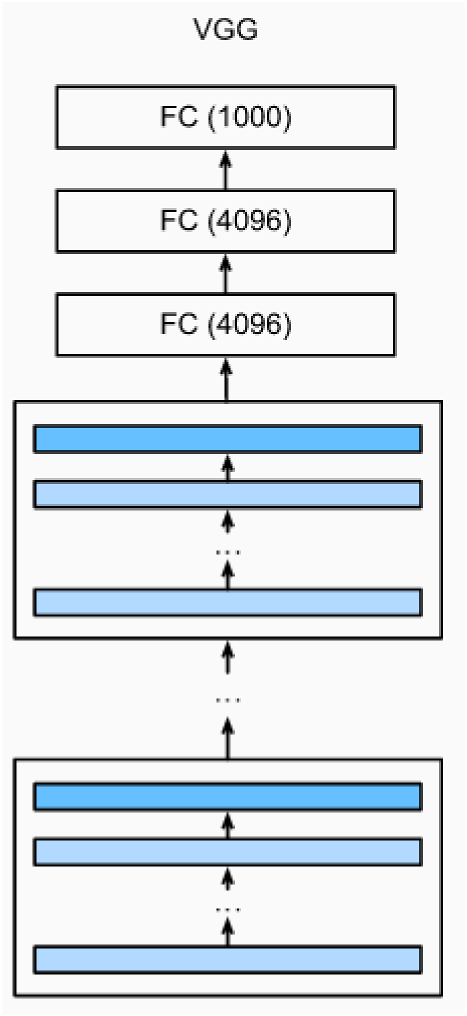


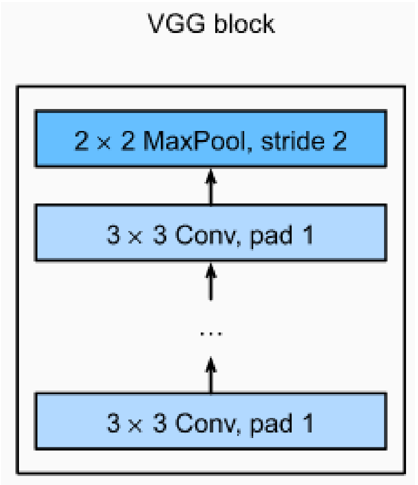
(Krizhevsky, Sutskever, Hinton, 2012)

Networks using blocks

- Proposed by Visual Geometry Group in UK
- VGG = (1) ConvL with padding to maintain the resolution, (2) nonlinearity, (3) pooling layer.
- Convolutional part = one or more VGG blocks
- VGG-11 = 8 ConvL + 3 FCL
- S&Z found that several layers of deep and narrow convolutions (i.e., 3×3) were more effective than fewer layers of wider convolutions.
- The use of blocks leads to very compact representations of the network definition. It allows for efficient design of complex networks.

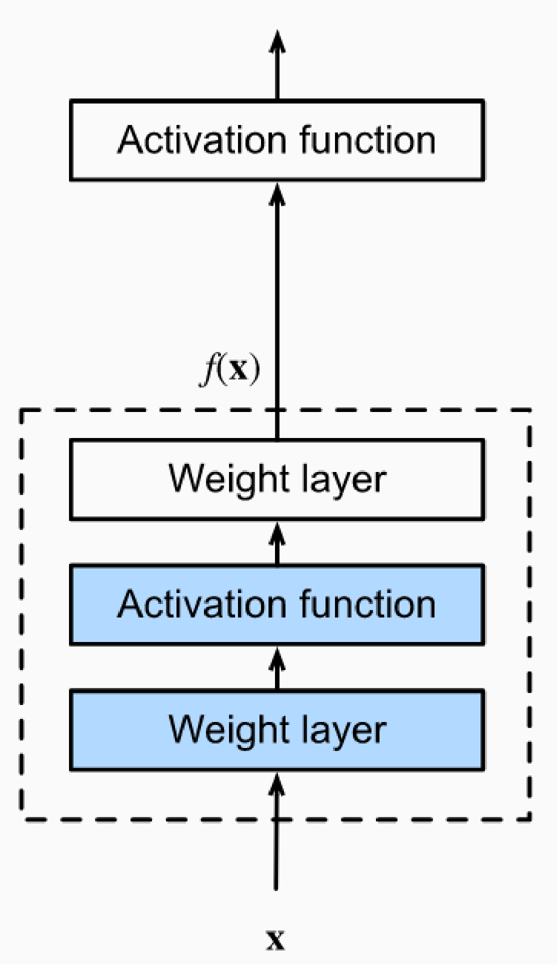
(Simonyan & Zisserman, 2015)



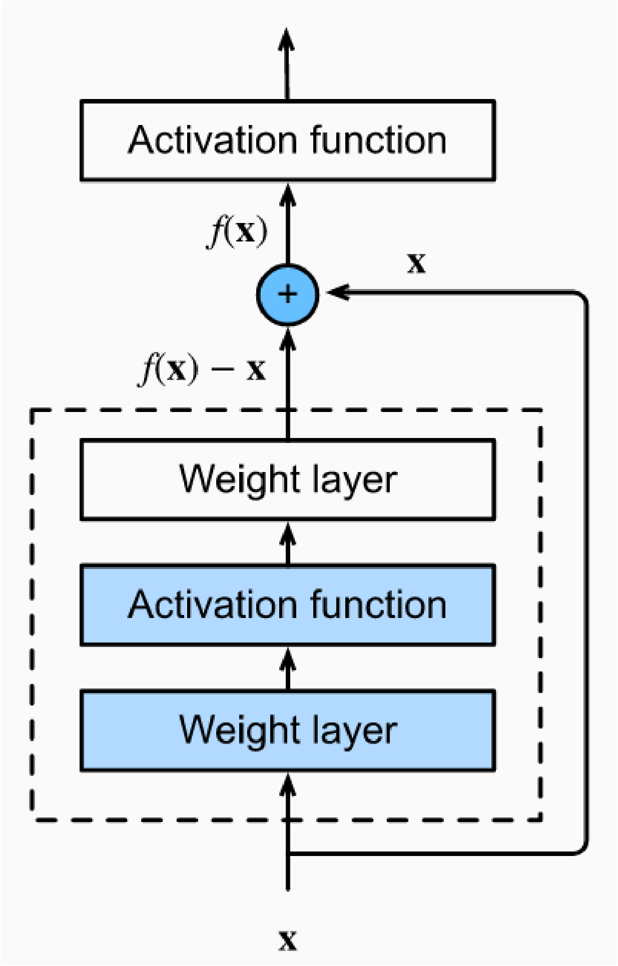


Residual networks

- Learns the residual mapping to $f(x) - x$
- ...using a shortcut
- is easier to learn, if target mapping is $f(x) = x$
- ResNet can be combined, e.g. with VGG
- ResNet won the 2015 ImageNet competition
- other popular models



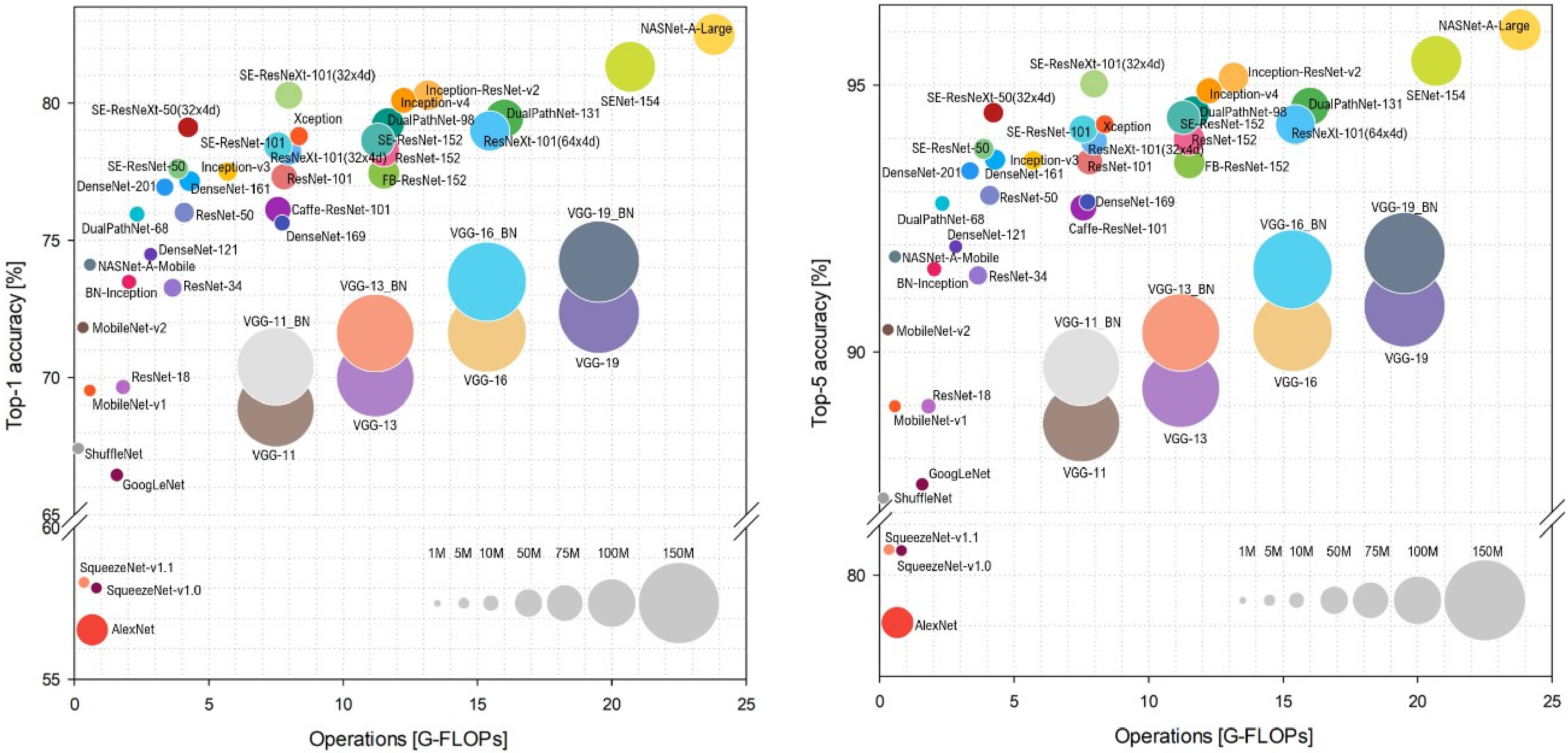
classical model



(He et al, 2015)

(Zhang et al, 2020)

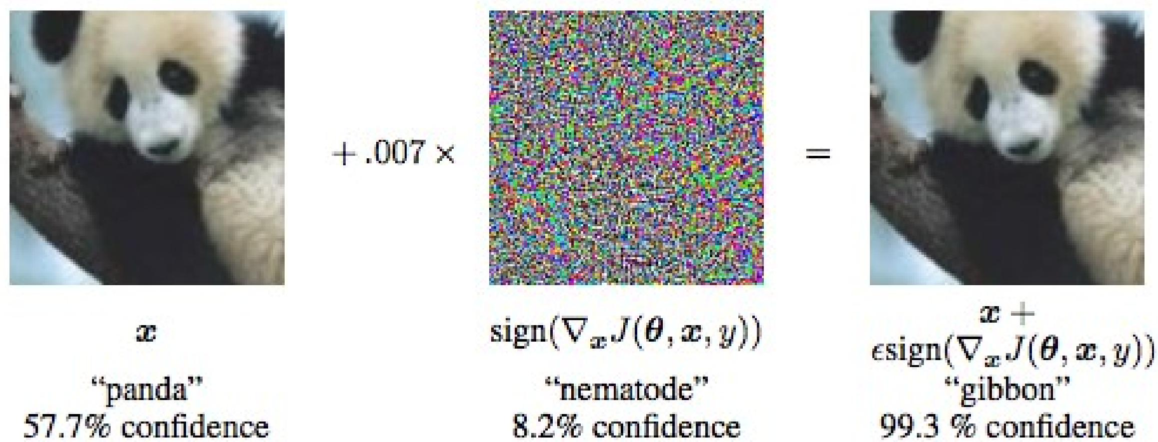
Performance of deep neural network models



Tested on ImageNet-1k

(Bianco et al., 2018)

Problems of deep networks



(Goodfellow et al, 2015)

- Deep networks lack robustness, because they are sensitive to adversarial examples (inputs with perturbations imperceivable by humans); applies to classifiers and RL models, too.
- There exist many adversarial attacks and respective defences
- AEs appear to be a feature, not a bug (Ilyas et al, 2019)
- Other problems: data greediness, limited generalization (contrary to humans)

Summary

- Deep learning very successful in concrete domain-specific applications
- end-to-end, i.e. no input preprocessing and/or feature extraction needed; shift towards engineering
- Various ways for improvement: proper weights initialization, activation functions, regularization,...
- Convolutional layers help implement various forms of invariance, and hence, increase accuracy.
- Convolution reduces number of trainable parameters
- Huge architectures reasonable and possible thank to high parallelization on GPUs and fast HW.
- Deep networks are very brittle (maybe shift to hybrid models).