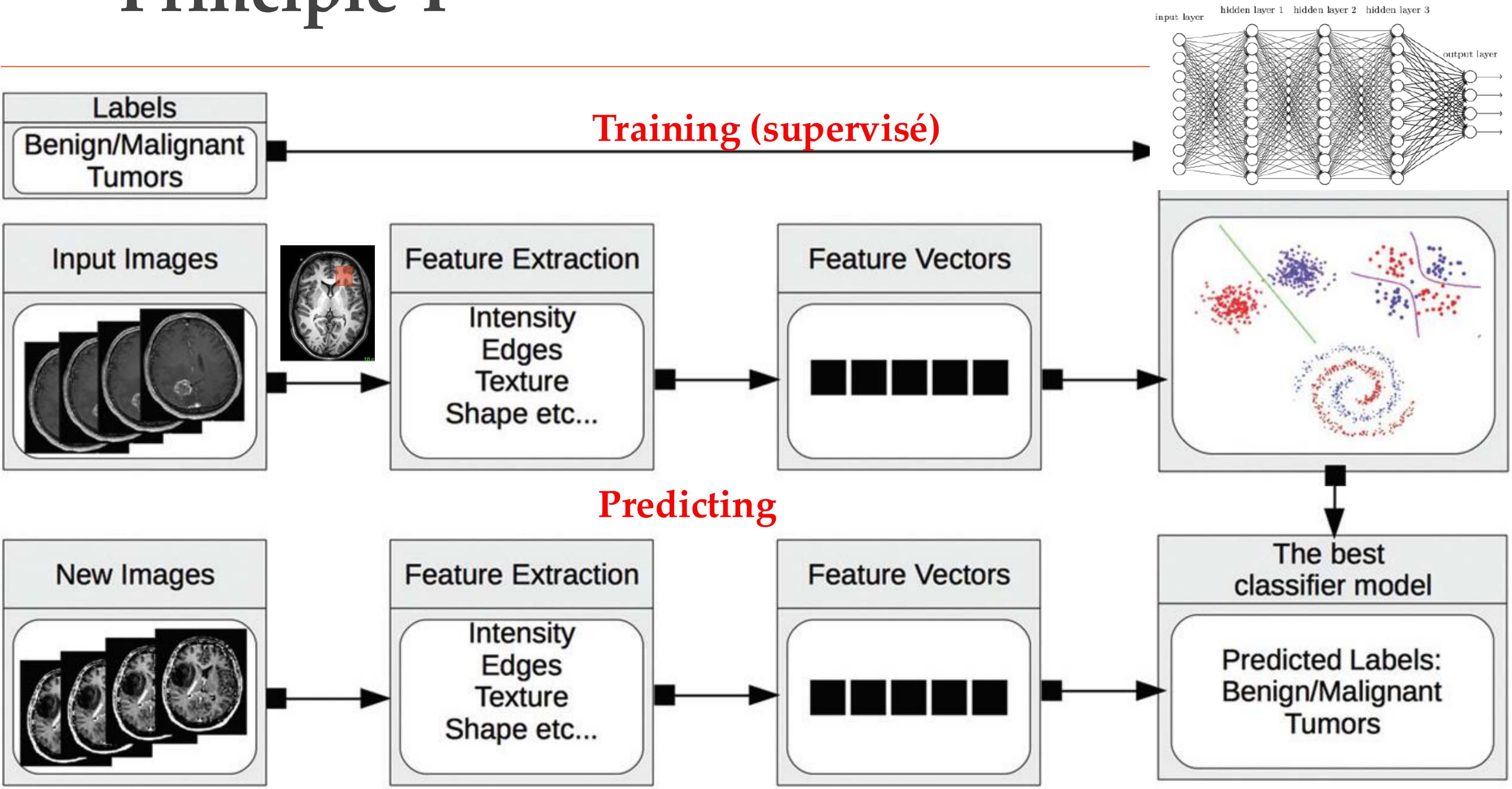
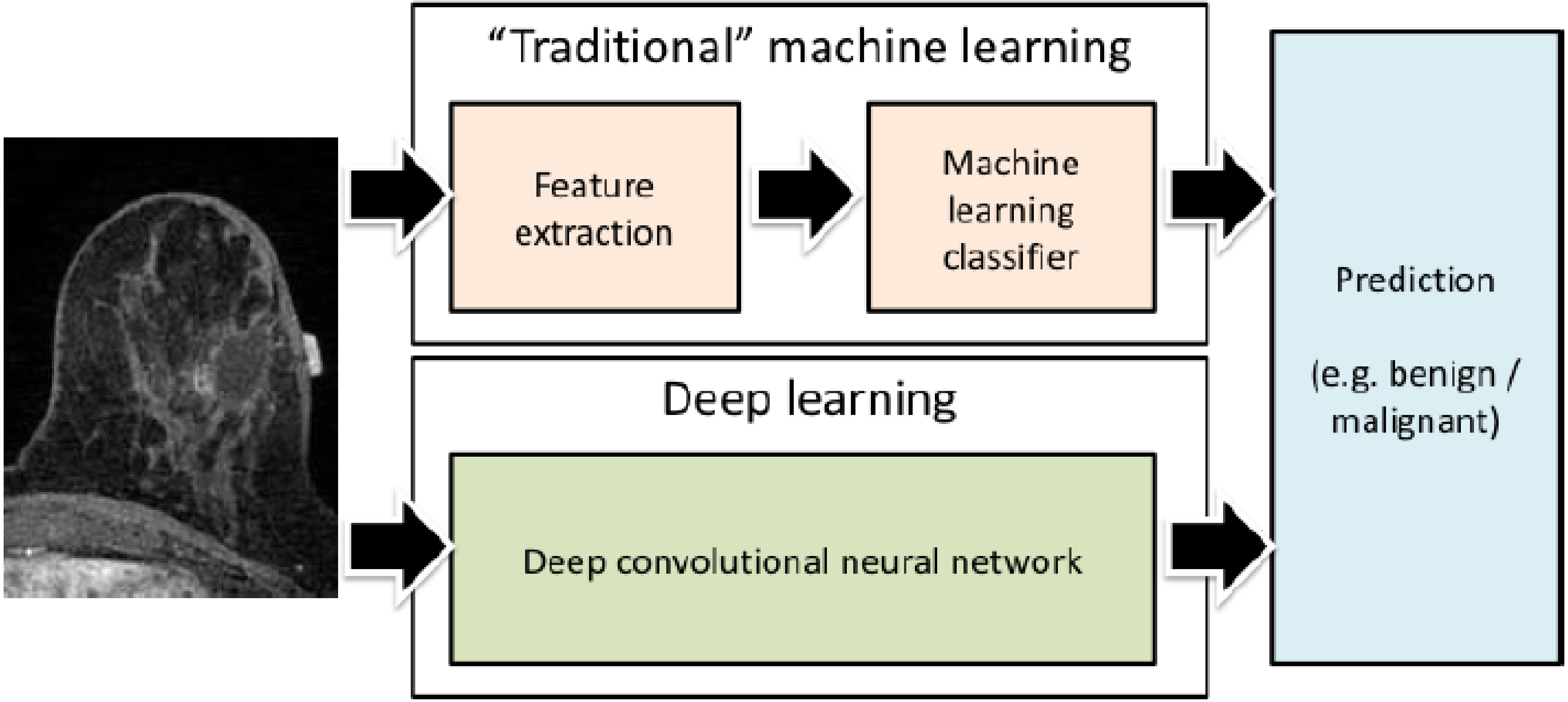
Principle-I



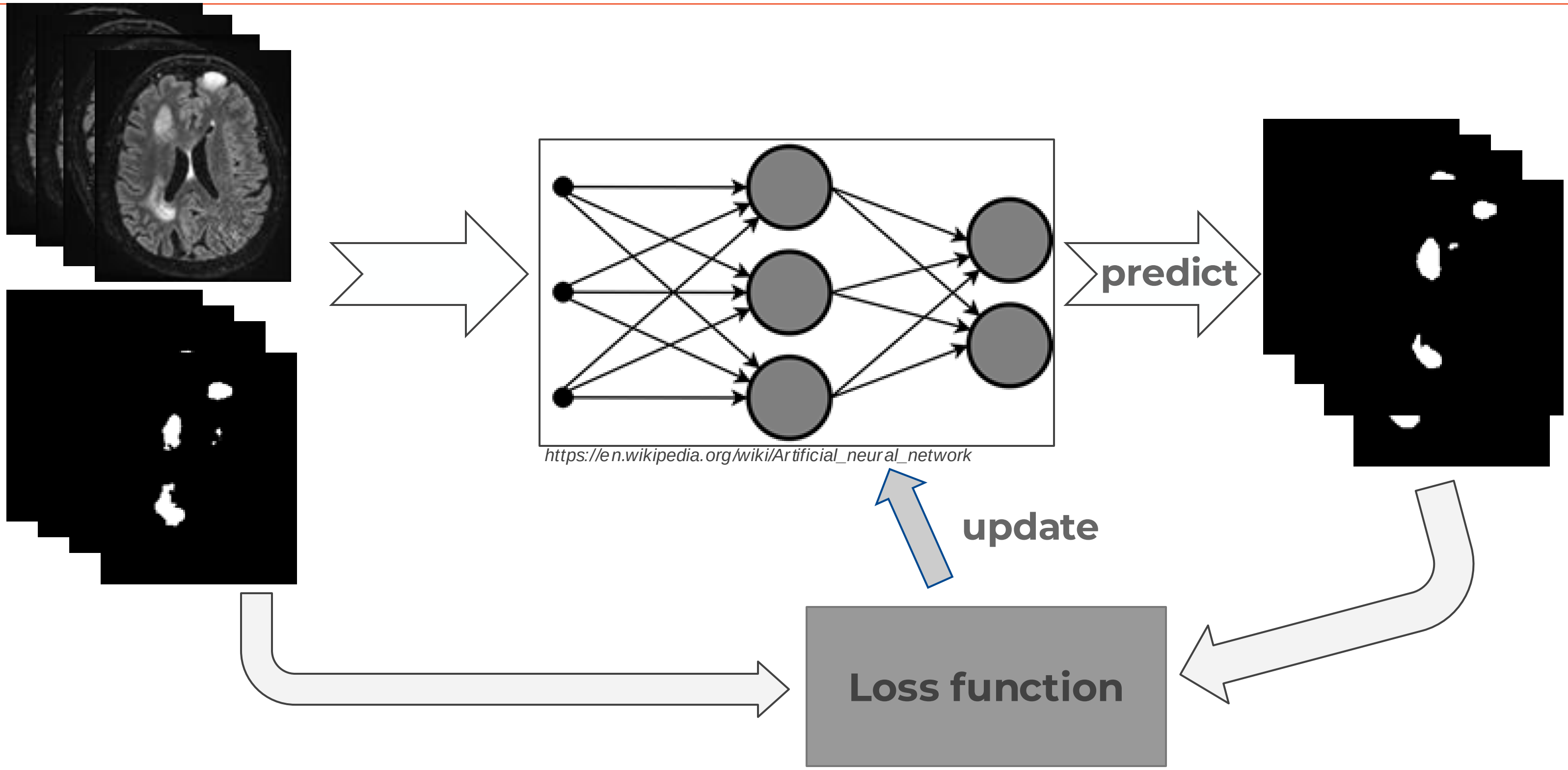
Labels
Benign/Malignant Tumors
Training (supervisé)
input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer
Input Images
Feature Extraction
Intensity
Edges
Texture
Shape etc...
Feature Vectors
Predicting
New Images
Feature Extraction
Intensity
Edges
Texture
Shape etc...
Feature Vectors
The best classifier model
Predicted Labels: Benign/Malignant Tumors

Principle - II

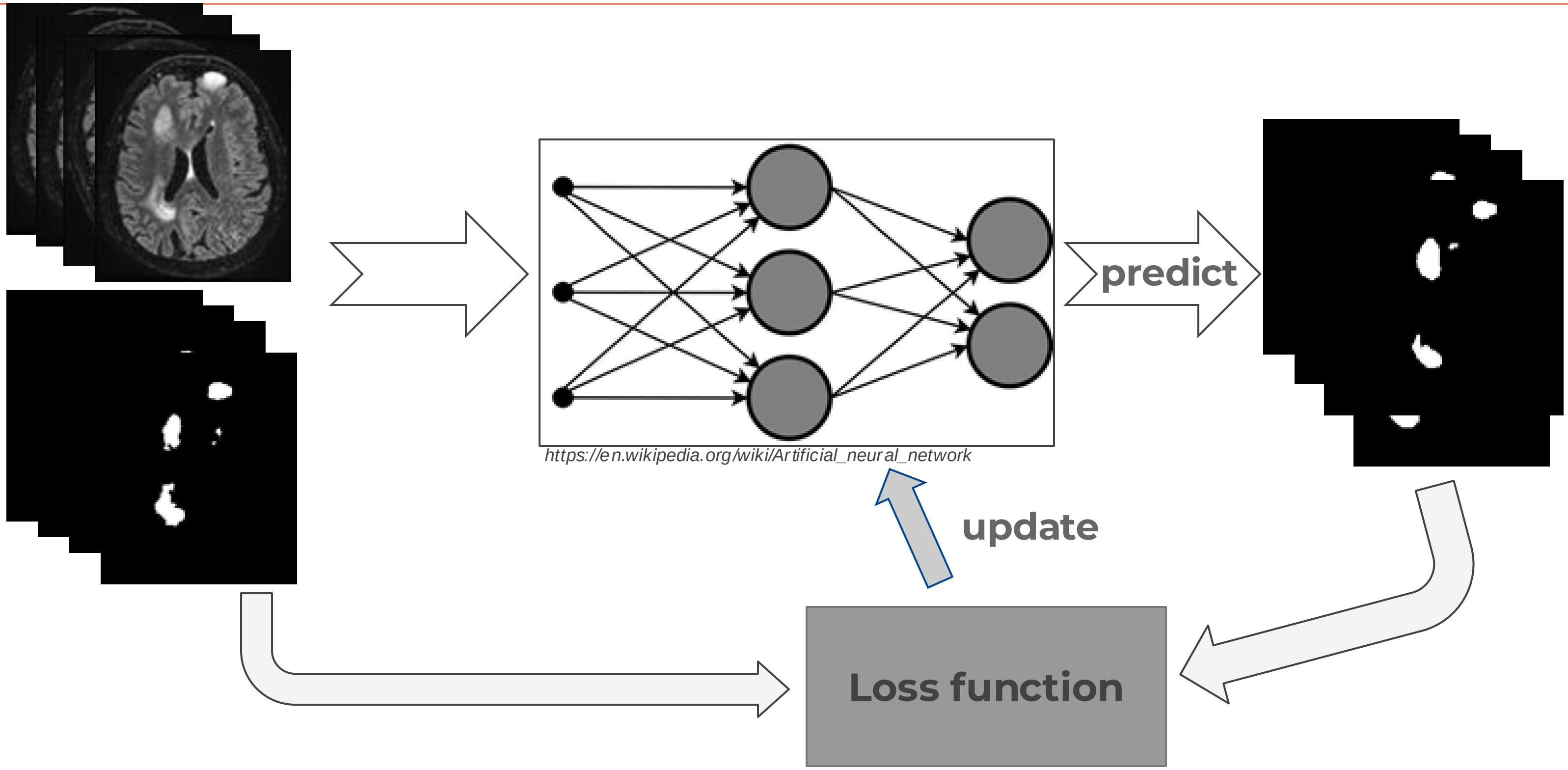


Mazurowski et al. JMRI 2018

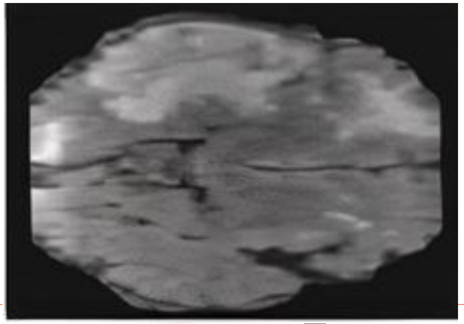
Principle III



Principle III

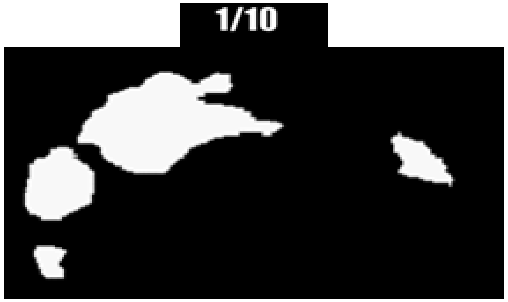


Flair image

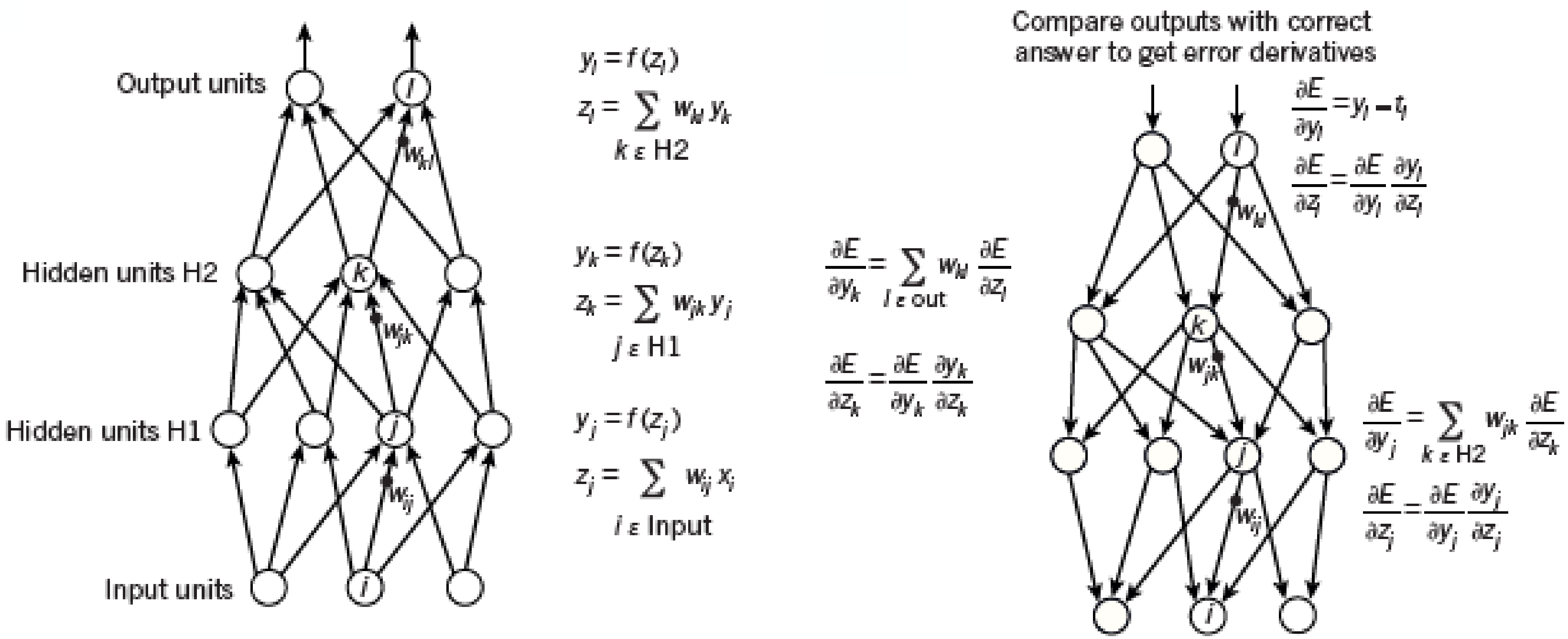


Ground Truth

Prediction



Neural Network



Le Cun et al. Nature 2015

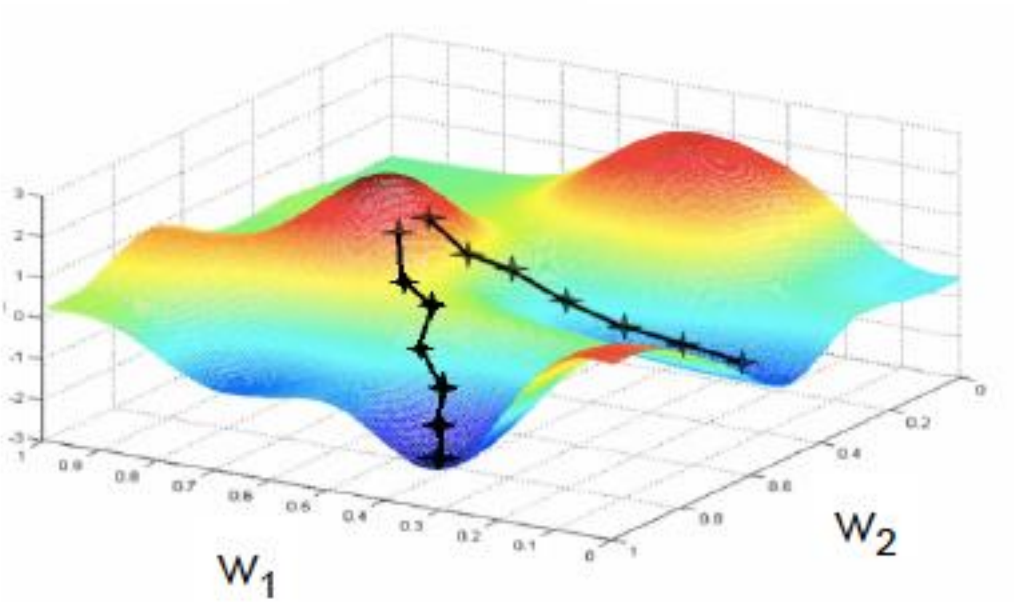
Training

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - f_{\mathbf{w}}(\mathbf{x}_i) \right)^2$$

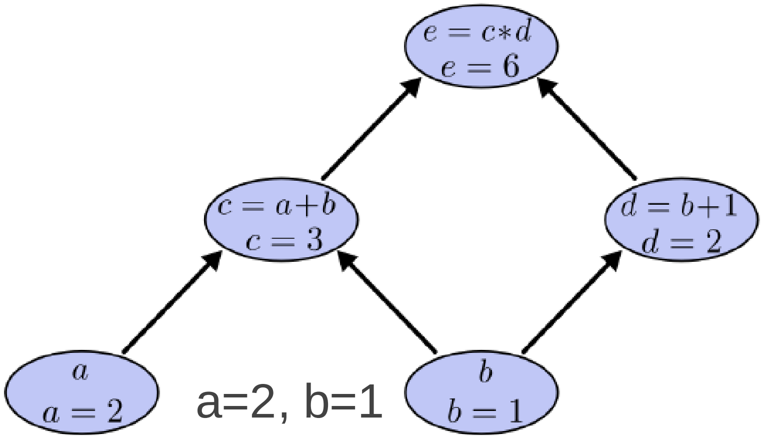
Weights updating by gradient descend

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

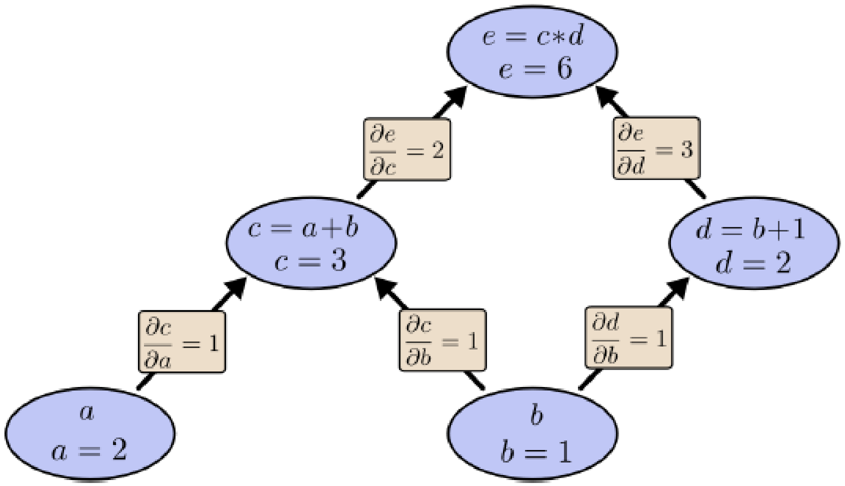
α = learning rate

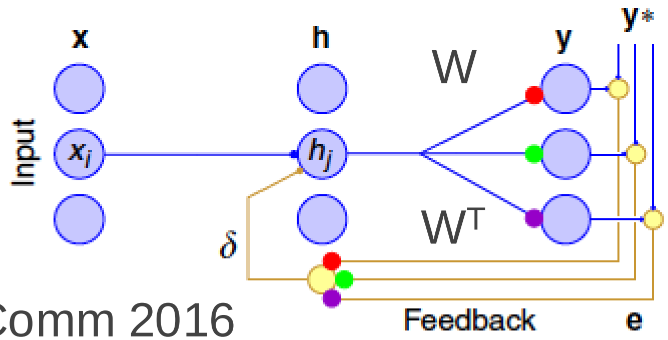


Simple mechanism



a=2, b=1





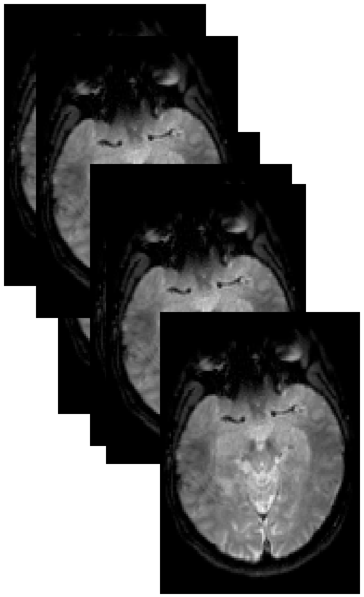
From Knoll Ismrm 2018

Lillicrap Nat Comm 2016

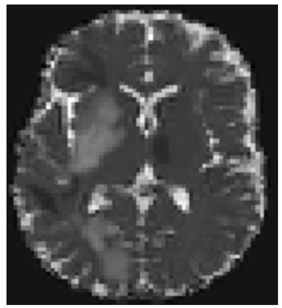
Multi-layer network

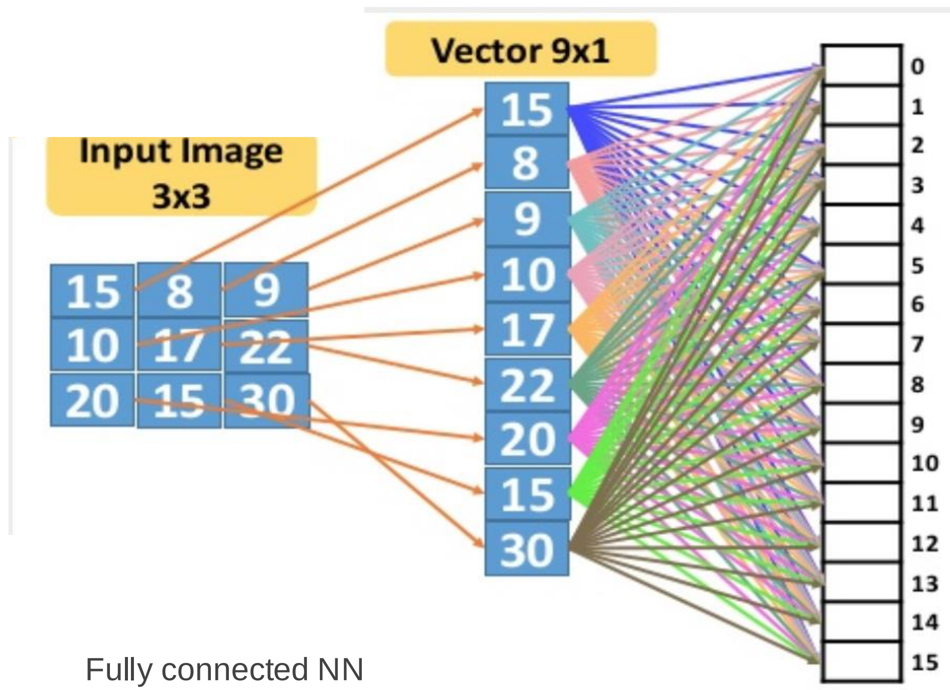
- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent:** compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

Image classification



VS

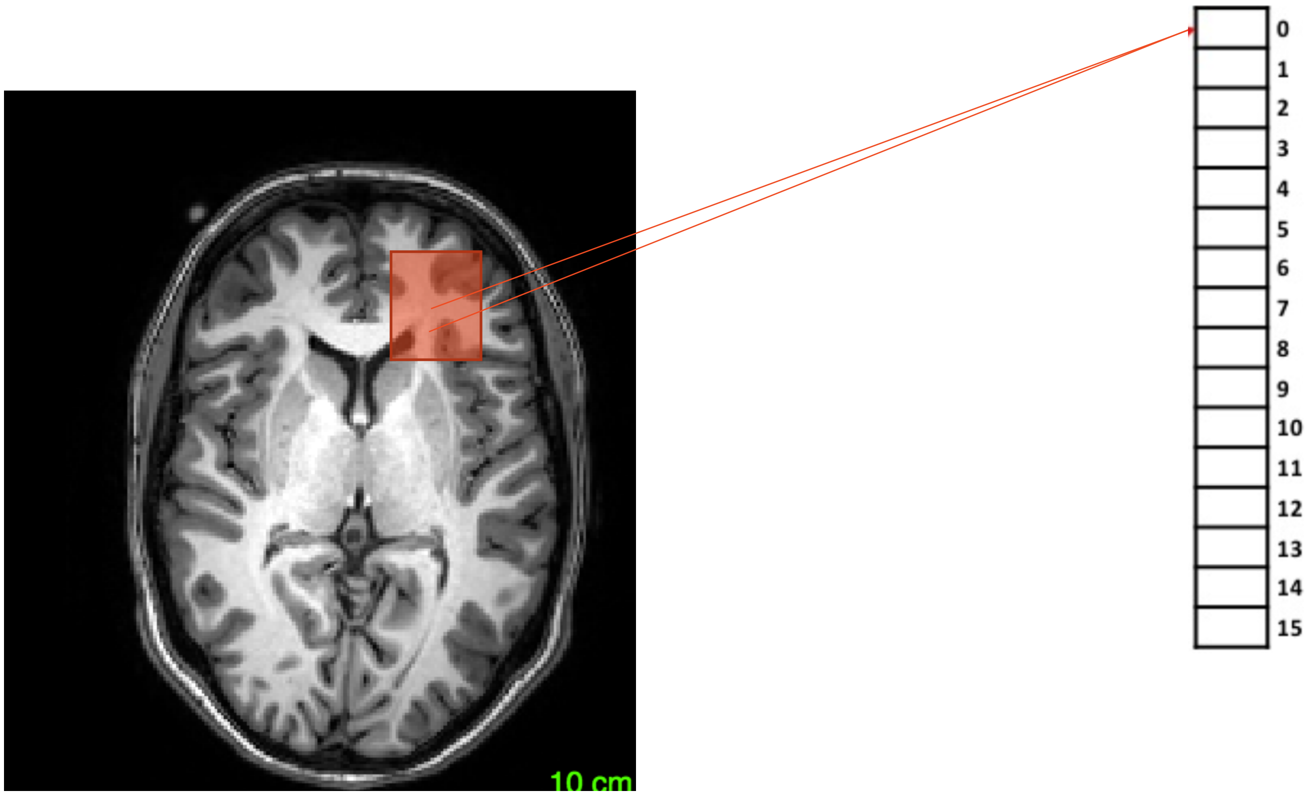




Fully connected NN

Adapted from Ahmed Gad

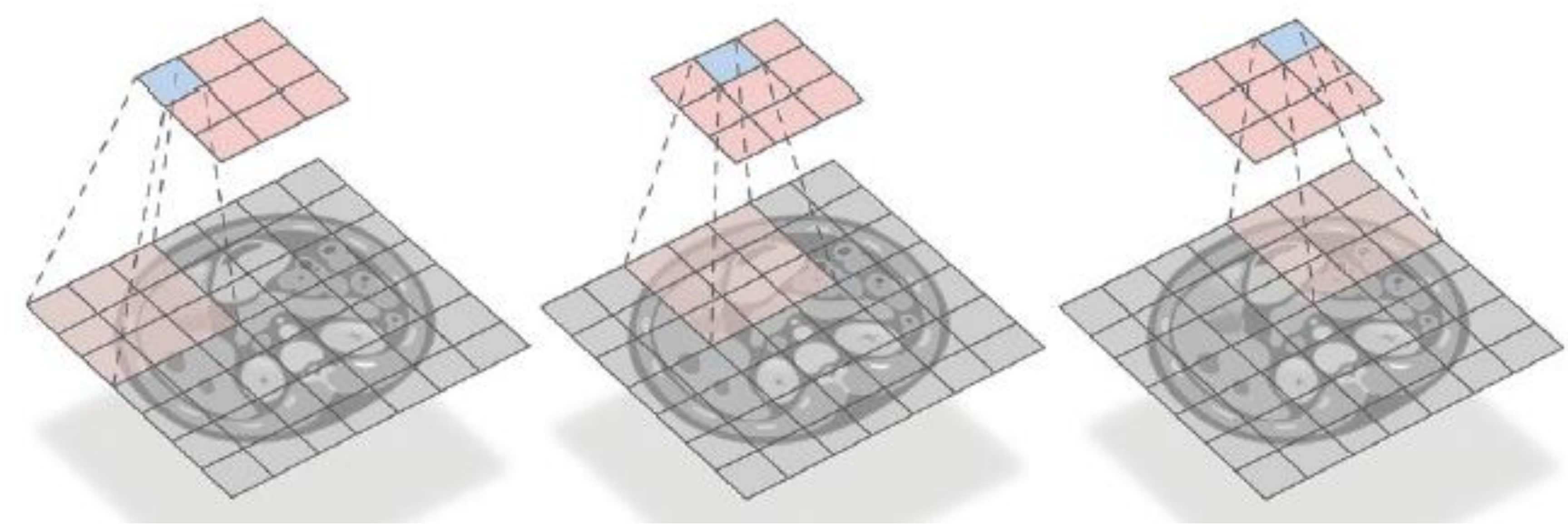
Convolutional neural network



256x256x16=65536x16=1048576 weights *3D !!

BUT: Each pixel is highly correlated to neighbours
Local statistics of images are generally invariant to location

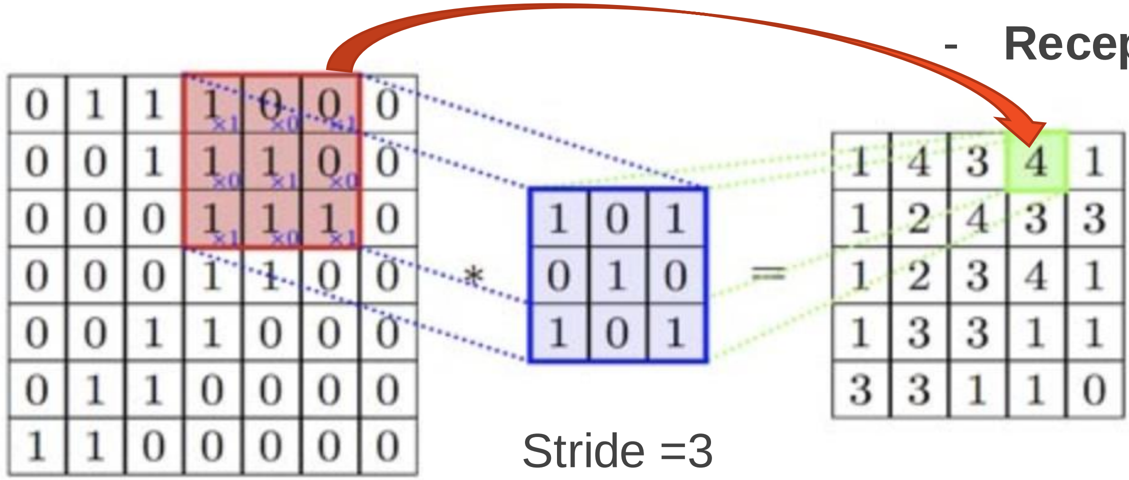
Convolution



Soffer et al Radiology 2019

Convolutional filter

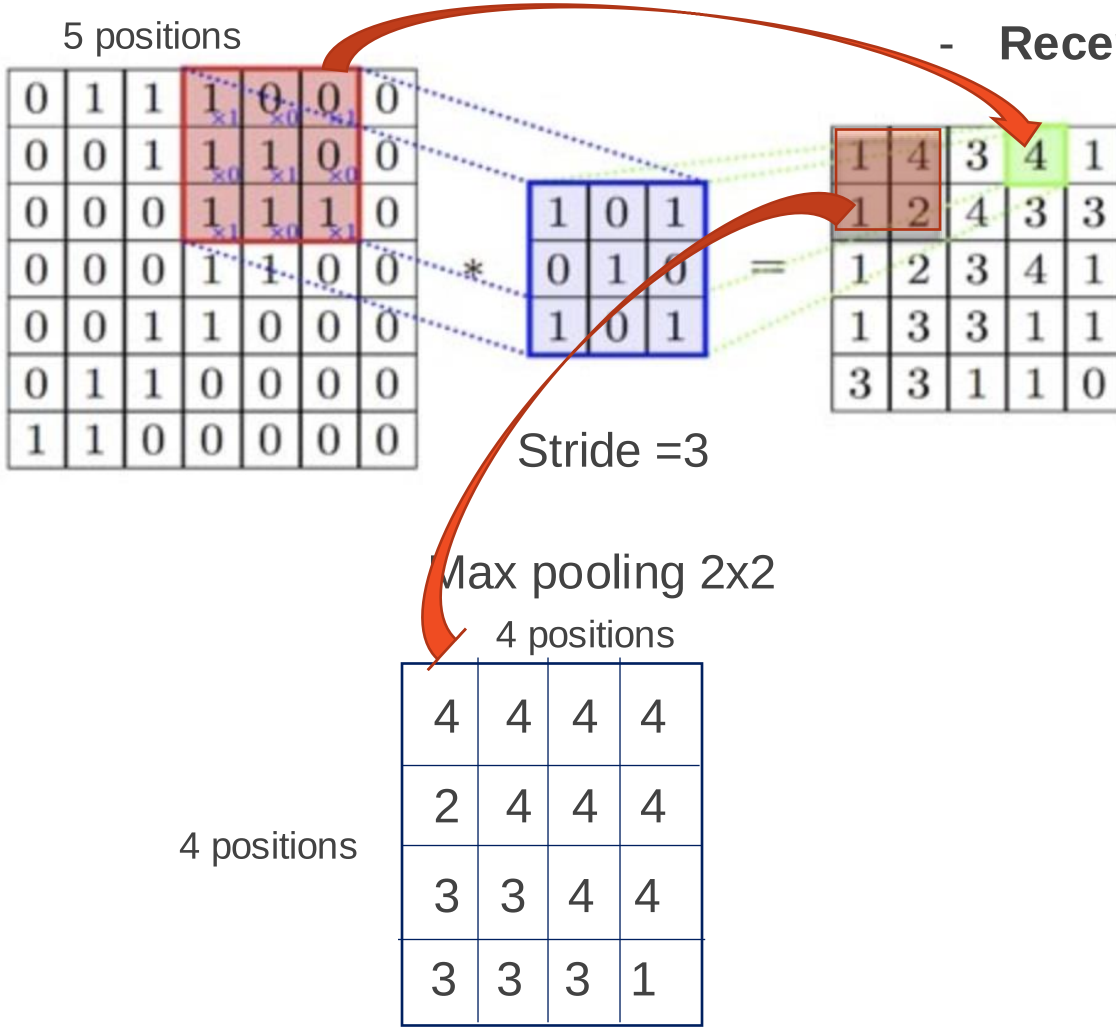
- Bio-inspired
- **Receptive field**



Convolutional filter

- Bio-inspired

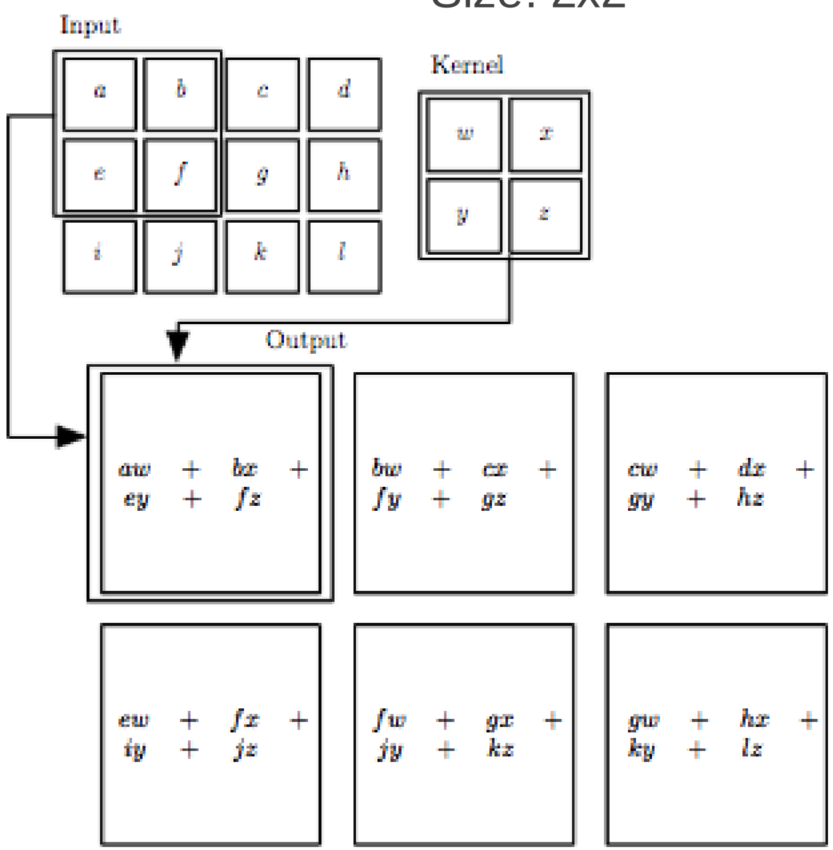
- **Receptive field**



Convolution

Size: 4x3

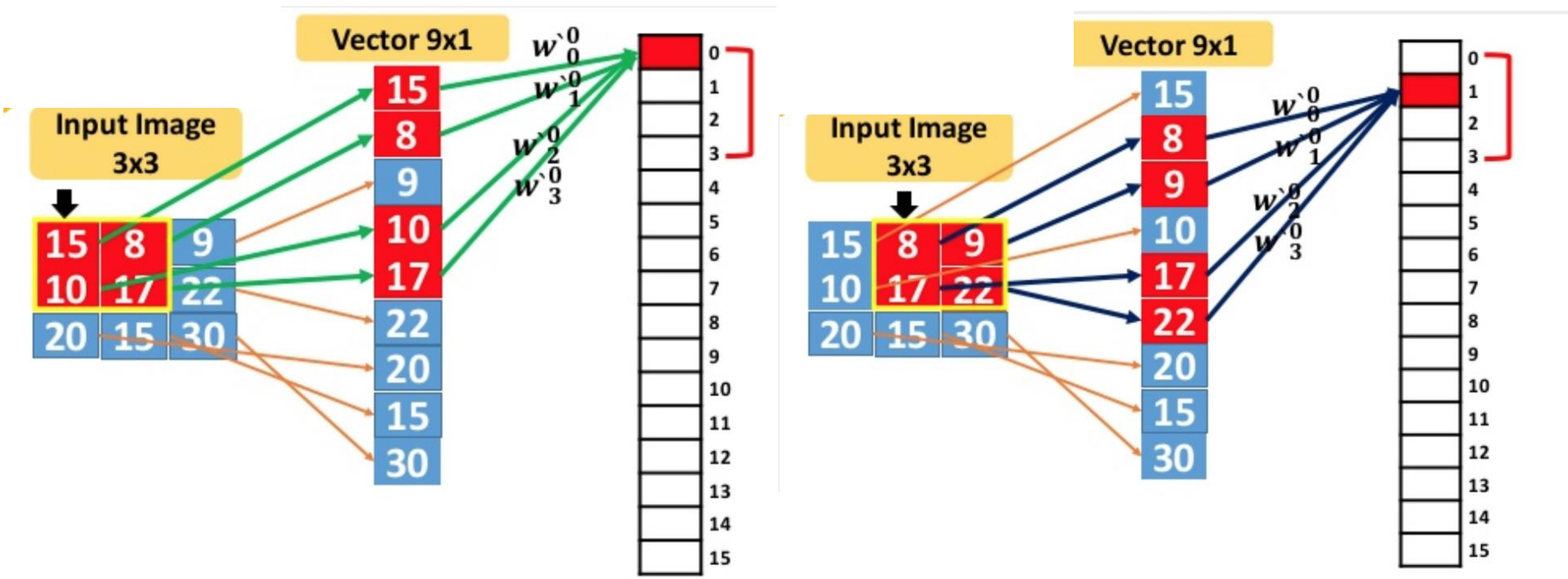
Size: 2x2



Size: 3x2

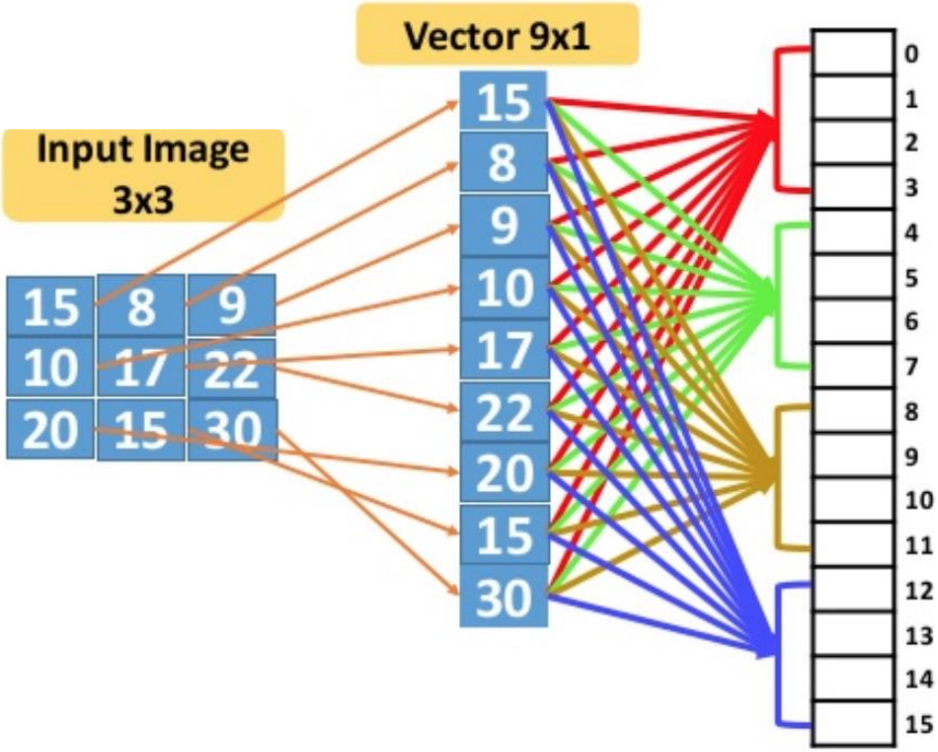
Convolutional neural network

For sparse connections



Adapted from Ahmed Gad

Convolutional neural network

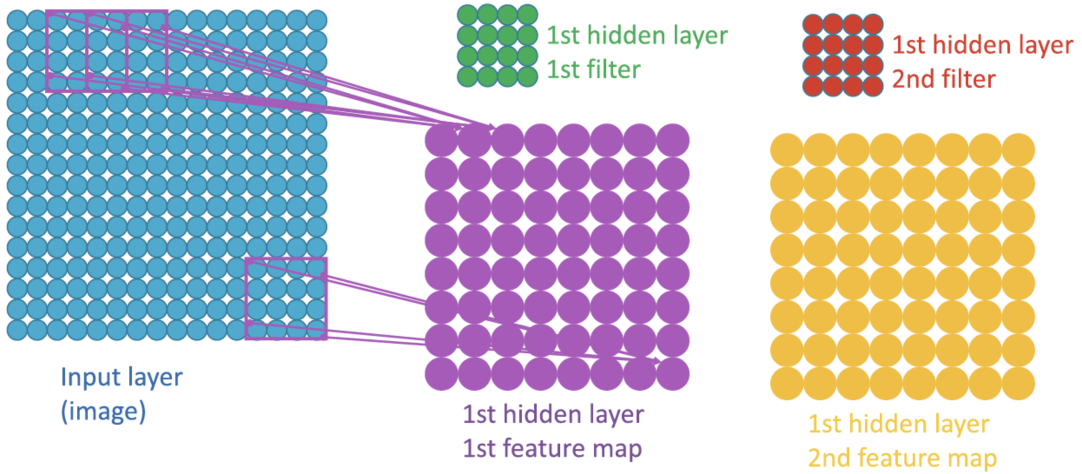


85x85x4=7225x4=28900 weights

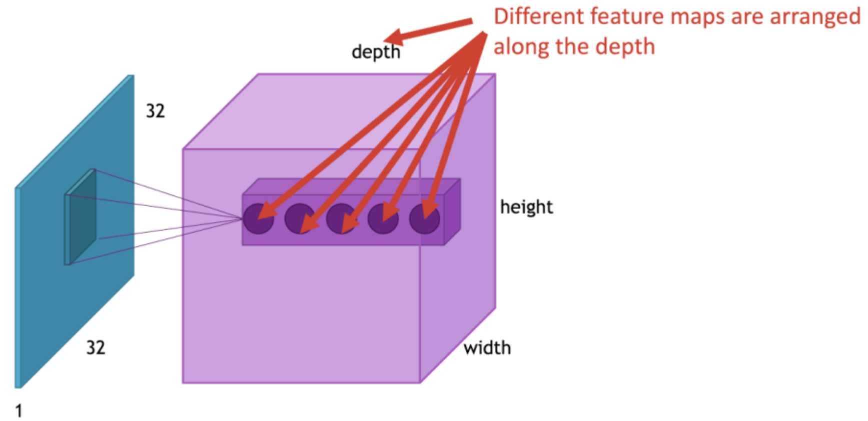
256x256x16=65536x16=1048576 weights

Adapted from Ahmed Gad

Many filters for features extraction

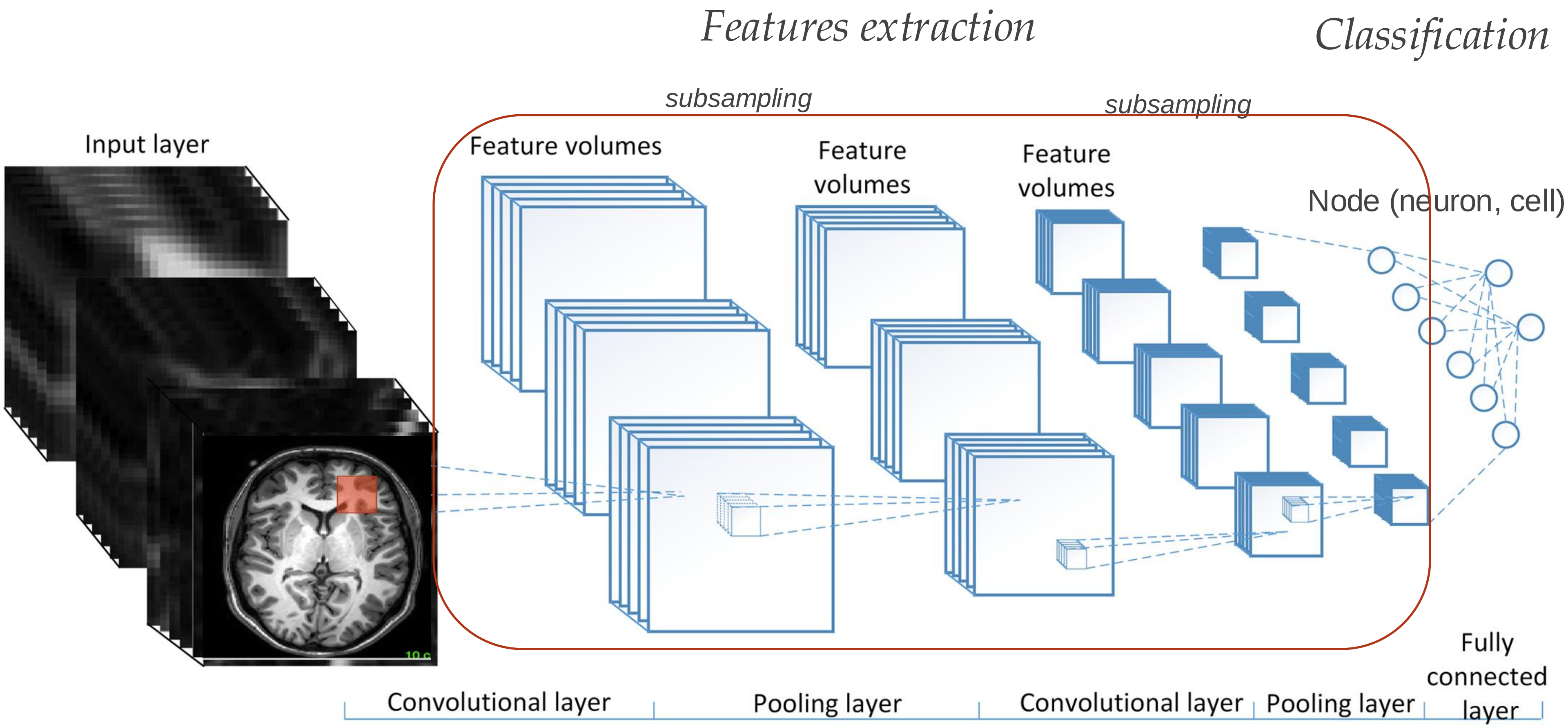


The feature maps are a 3D array for 2D input (and a 4D array for a 3D image)



Vakalopoulou et al. 2023

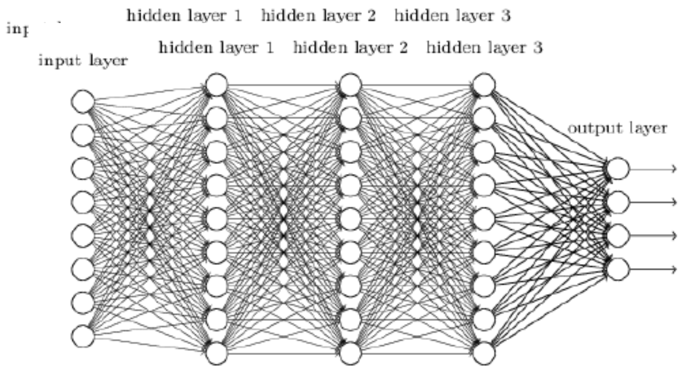
Example: Hierarchical construction



From Kang et al. PlosOne 2017

CNN Jargon

Node: Local part of a NN that involves two or more inputs, an activation function and produces an output. The activation function combines inputs to produce the output.

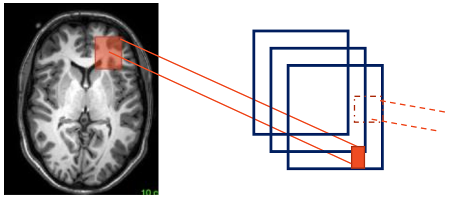


Layer: A set of nodes that are interconnected. Some layer may be hidden i.e. w.o. any connection with the external world

Weight: Each input is multiplied by some value (weighting operation). Each weight value is updated during the training/validation phases based on errors at its ouput to build the best model face to the data.

Batch: refers to number of examples considered at each training step (**epoch**).

CNN Jargon



Depth: Depth (or **feature** or **channel)** corresponds to the number of **filters** (**kernels**) we use for the convolution operation.

Filter: uses to extract information at each layer by convolution with the inputs. Defined by its size.

PAD: Put zero values to a zone around the filter defined by the PAD size.

Stride: Stride is the number of pixels by which we slide our filter matrix over the input matrix. Stride of n means that every n pixels will be mapped to 1 pixel in the next layer.

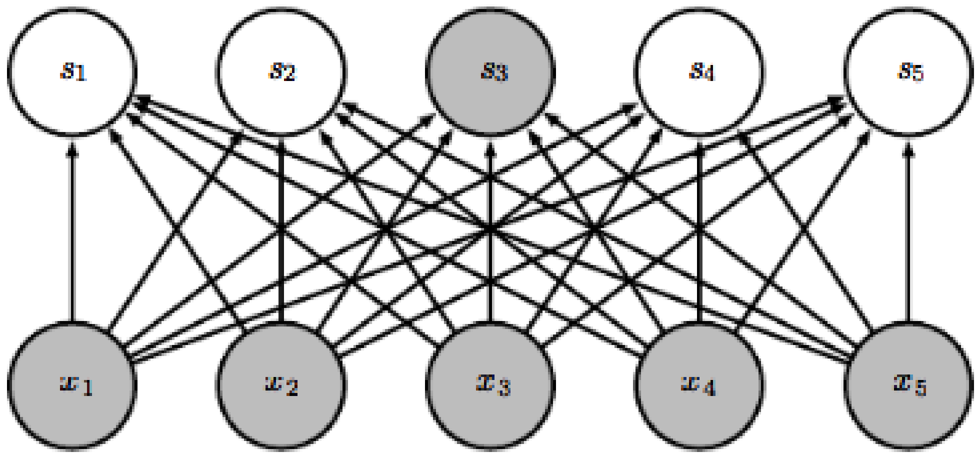
ReLU: is applied per pixel and replaces all negative pixel values in the feature map by zero to introduce non-linearity (Convolution is a linear operation).

Spatial Pooling: (subsampling or downsampling) reduces the dimensionality of each feature map. Map Pooling is generally used.

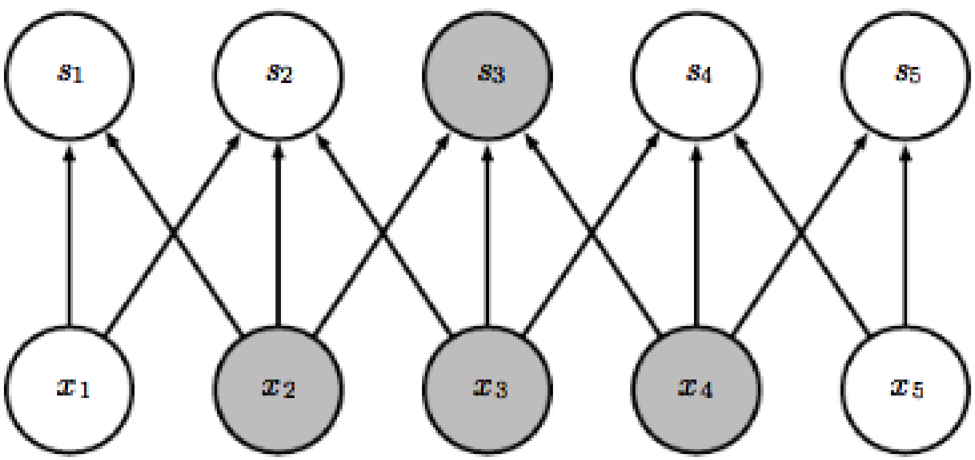
Dropout refers to ignoring units (i.e. neurons) chosen at random during the training phase.

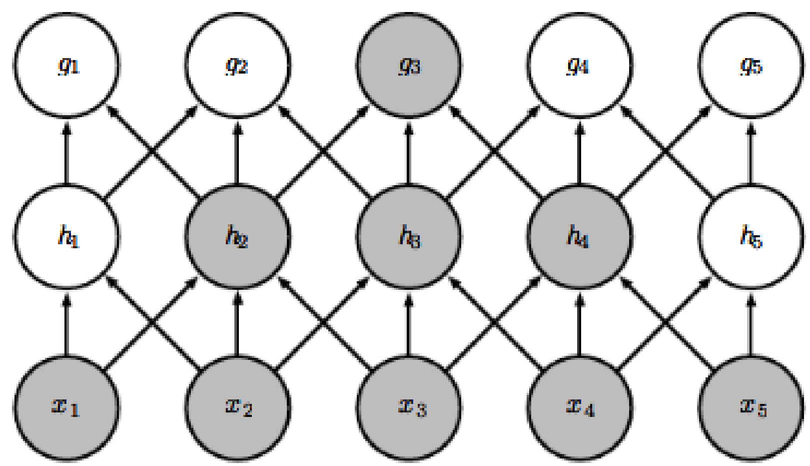
Sparse Connectivity

- Fully connected



- Sparse connections



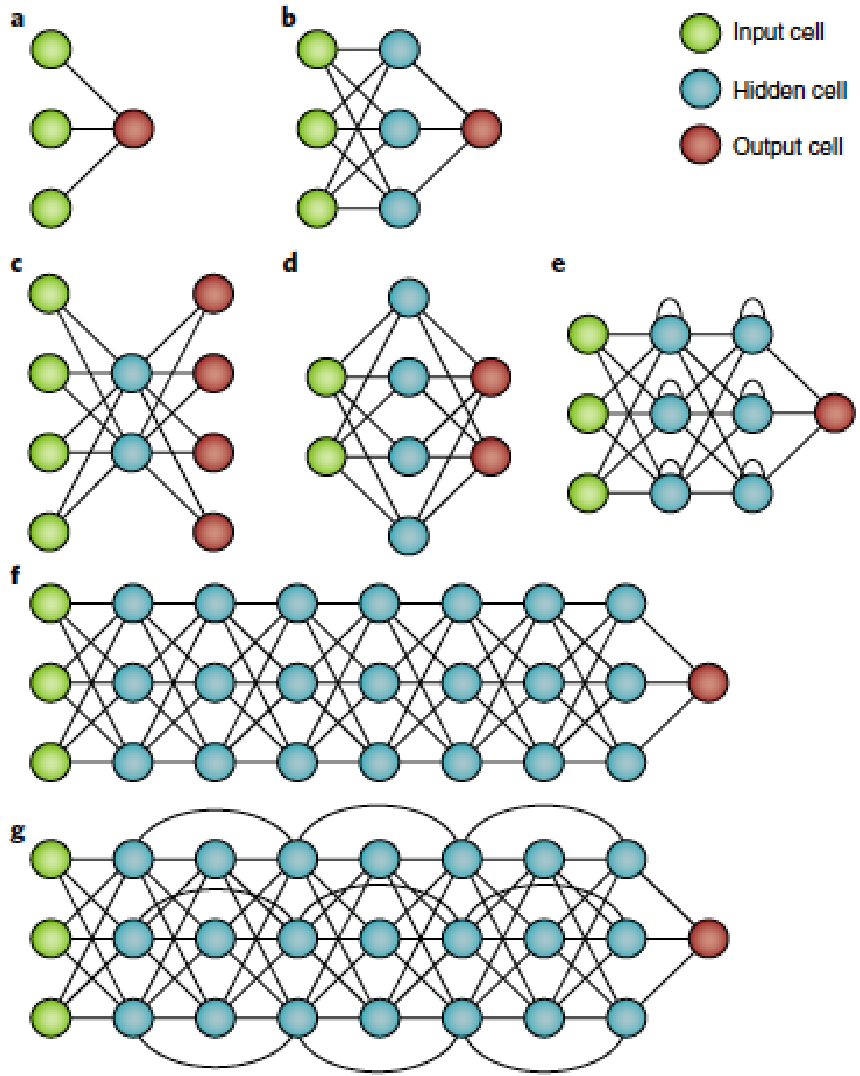


Nb param to estimate each l: nb neurons x nb connexions l+1

Different architectures

Complexity

autoencoders



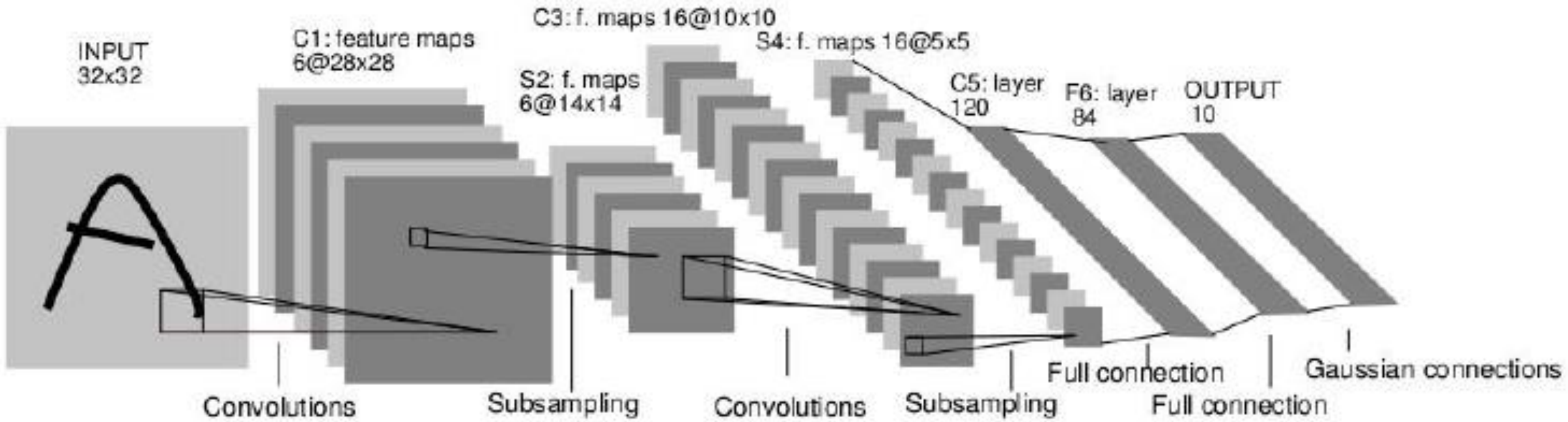
Recurent NN

Yu et Kohane Nat Biomedeng 2018]

Convolutional Neural Network

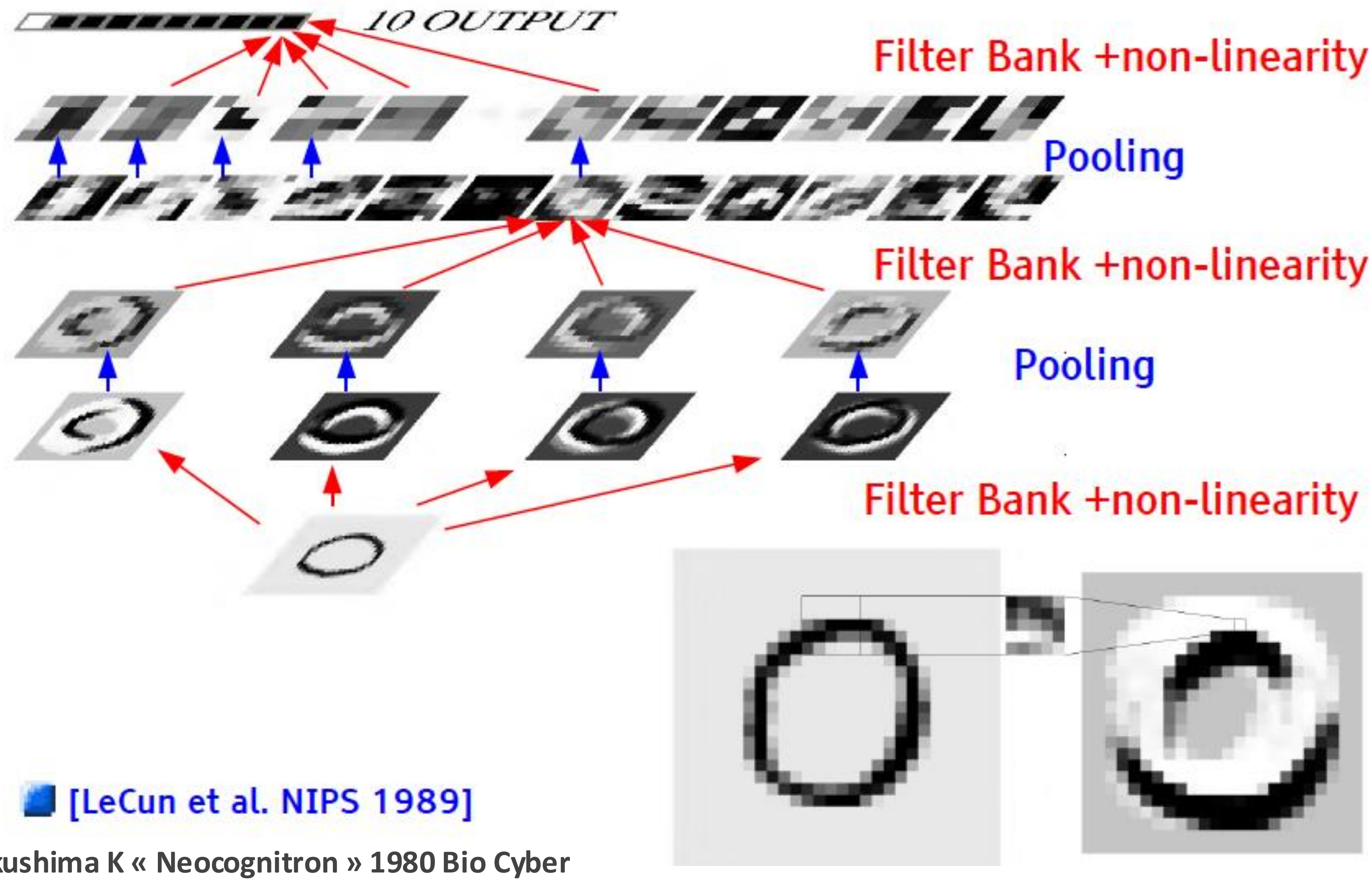
Emerge from Computer Vision research

LeNet 5



Le Cun et al Proc IEEE 1998

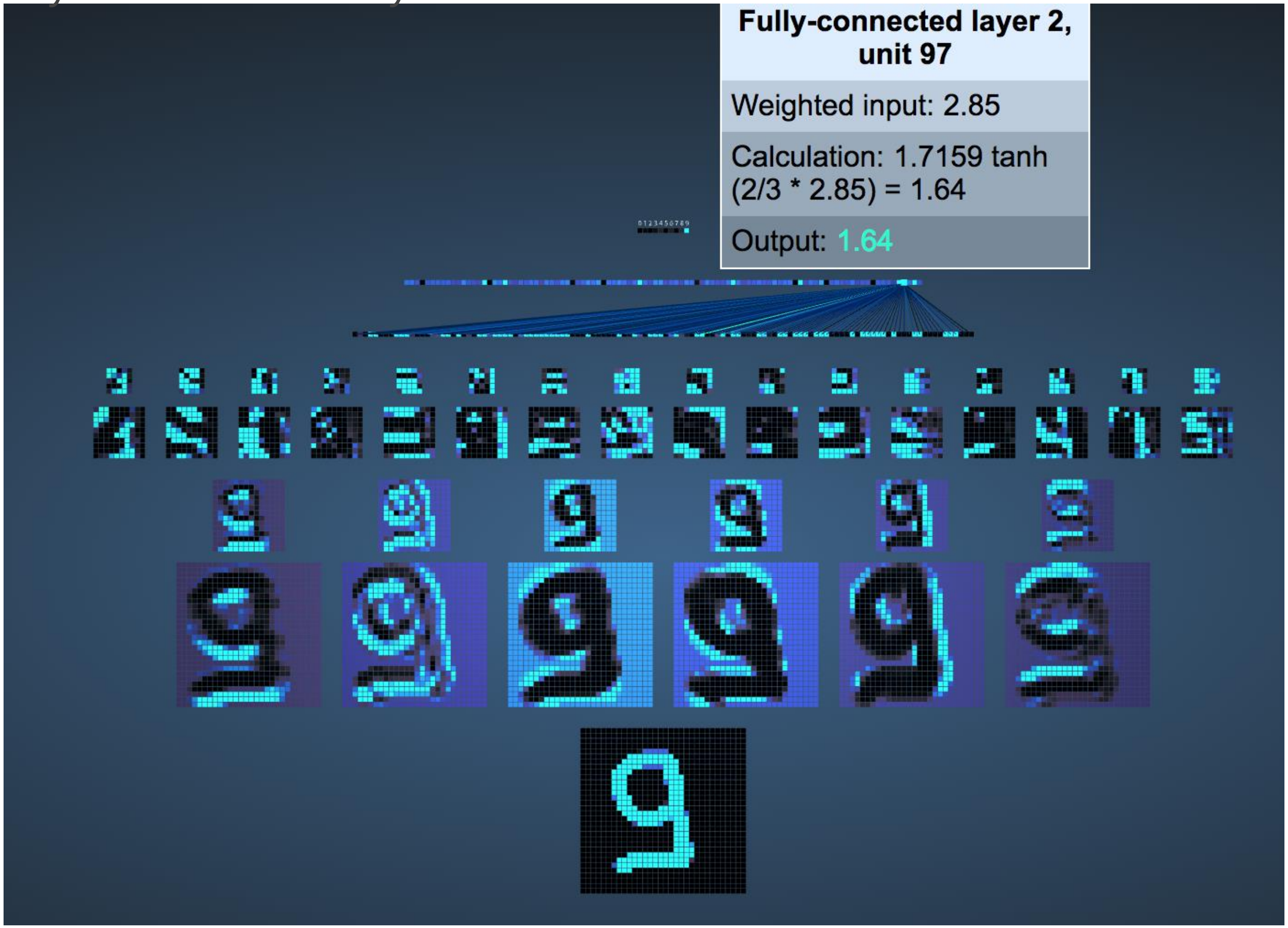
CNN



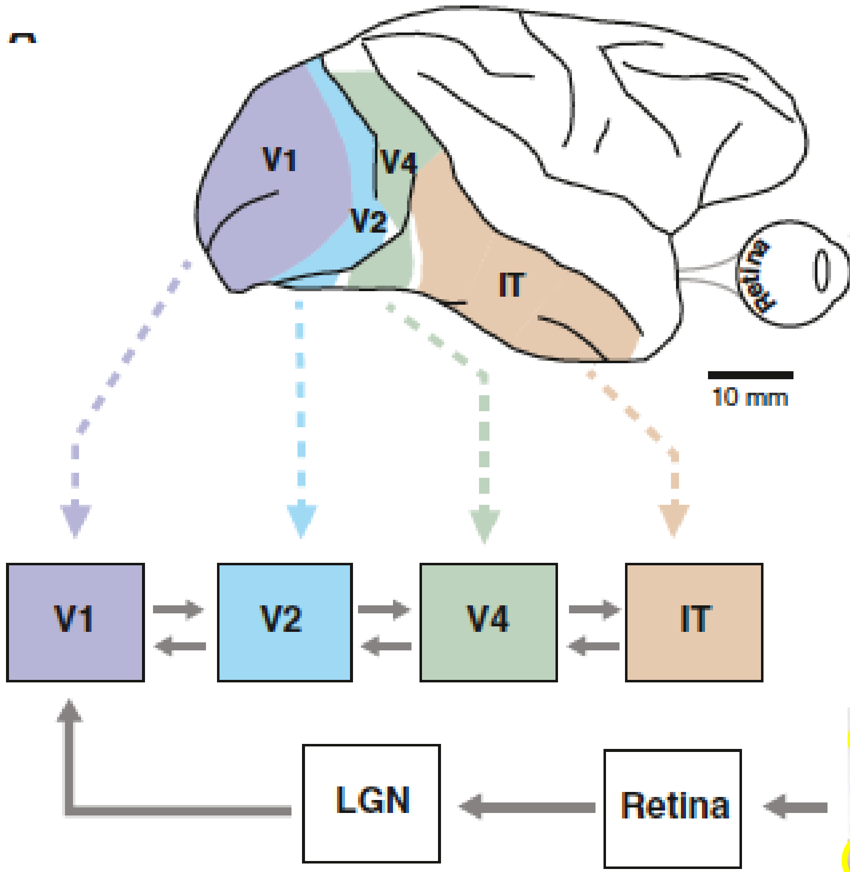
Fukushima K « Neocognitron » 1980 Bio Cyber

CNN

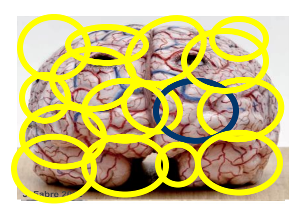
http://scs.ryerson.ca/~aharley/vis/conv/flat.html

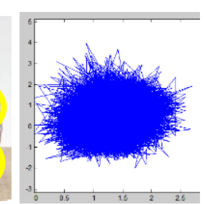


Mammalian visual system



Simultaneous and parallel computing

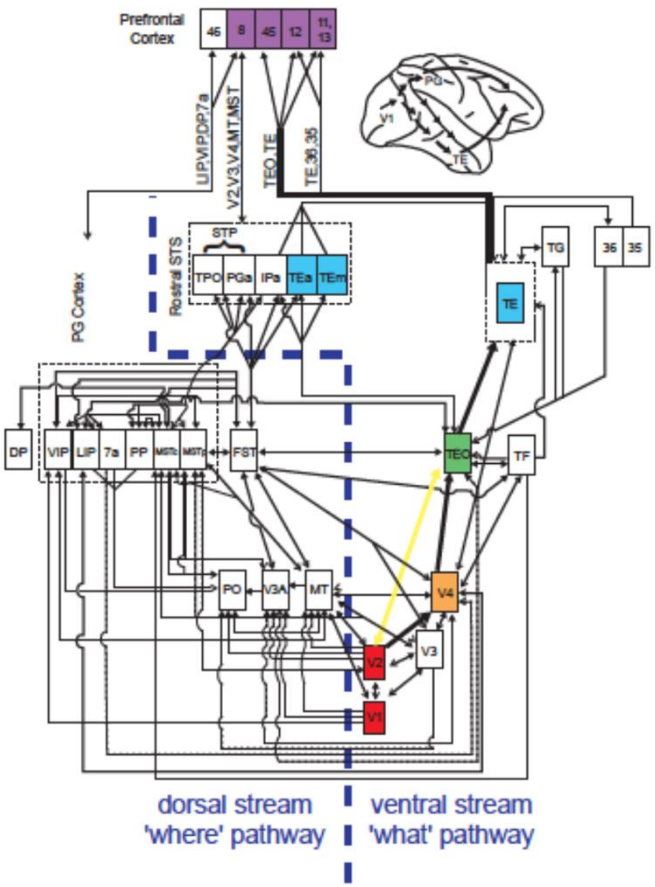




Image

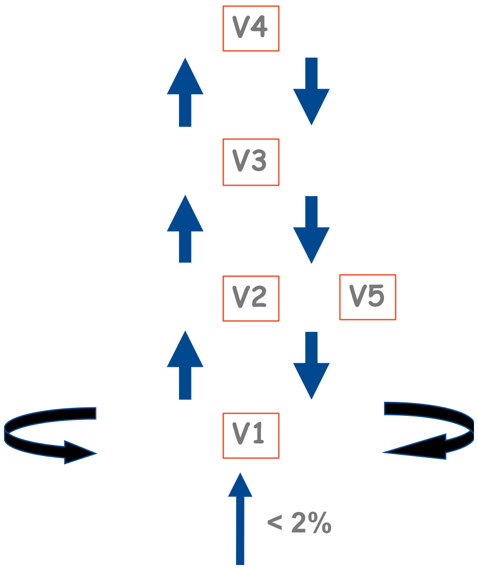
[adapted from Cox & Dean curr bio 2014]

Brain connectivity



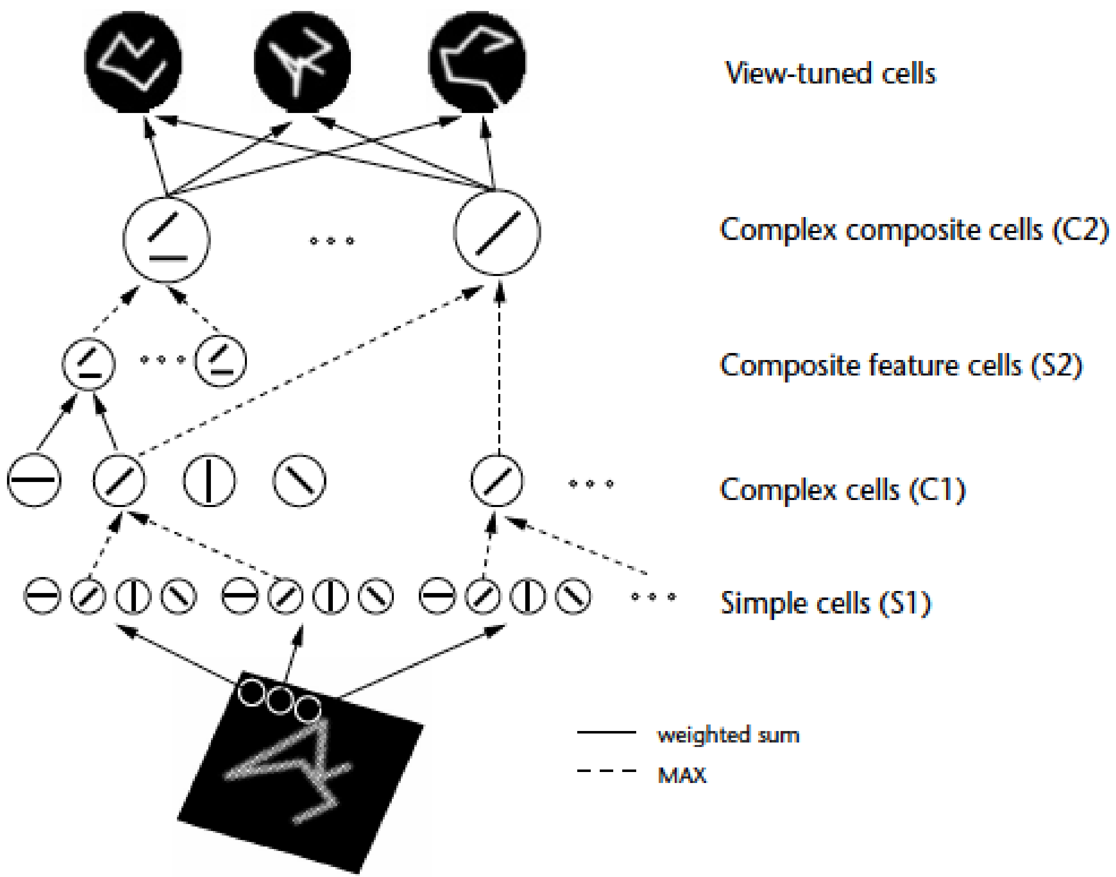
[Serre et al 2005 Tech Report]

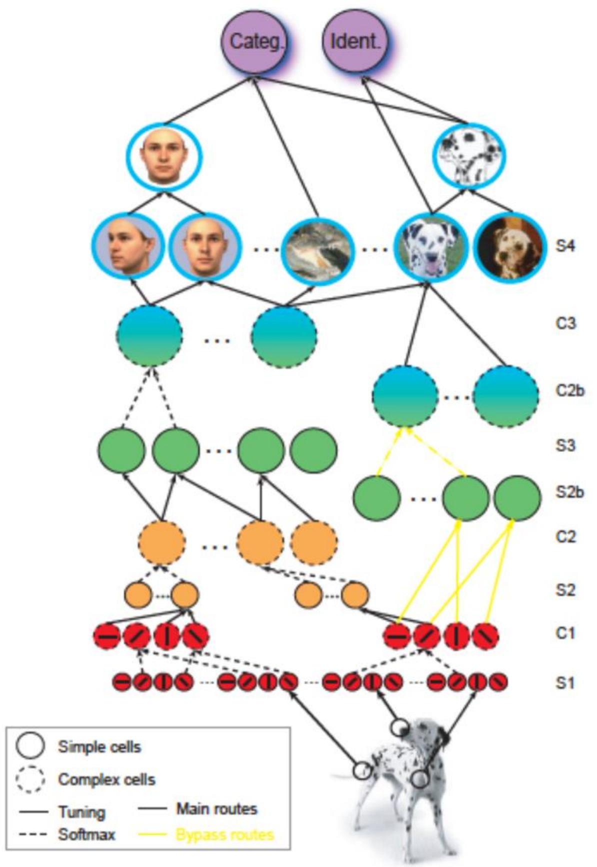
- Hierarchical model
- Several areas interconnected (layers)
- Receptive field sizes increasing
  - Local features
  - Global grouping



MAX pooling

MAX pooling: invariance to scale and translation; key mechanism for object recognition



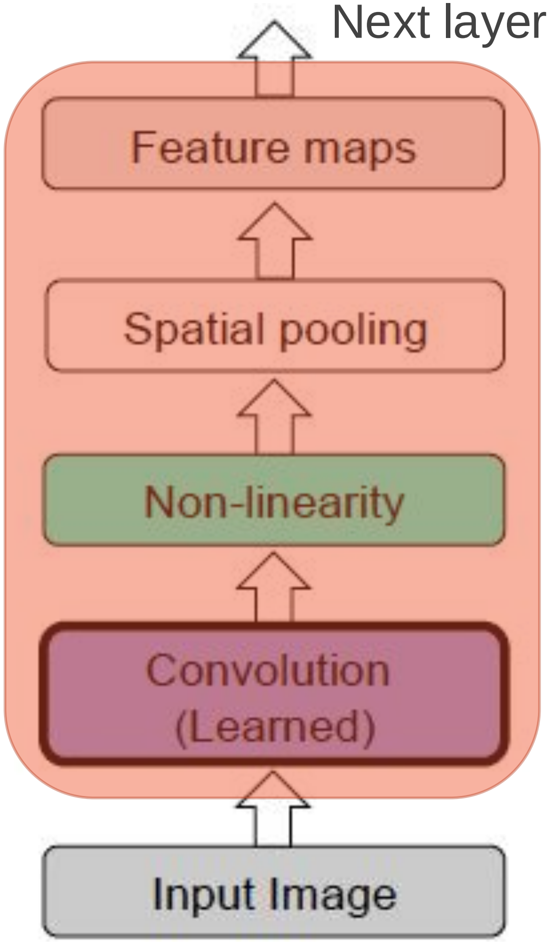


[Riesenhuber & Poggio Nature 1999]

[Serre et al 2005 Tech Report]

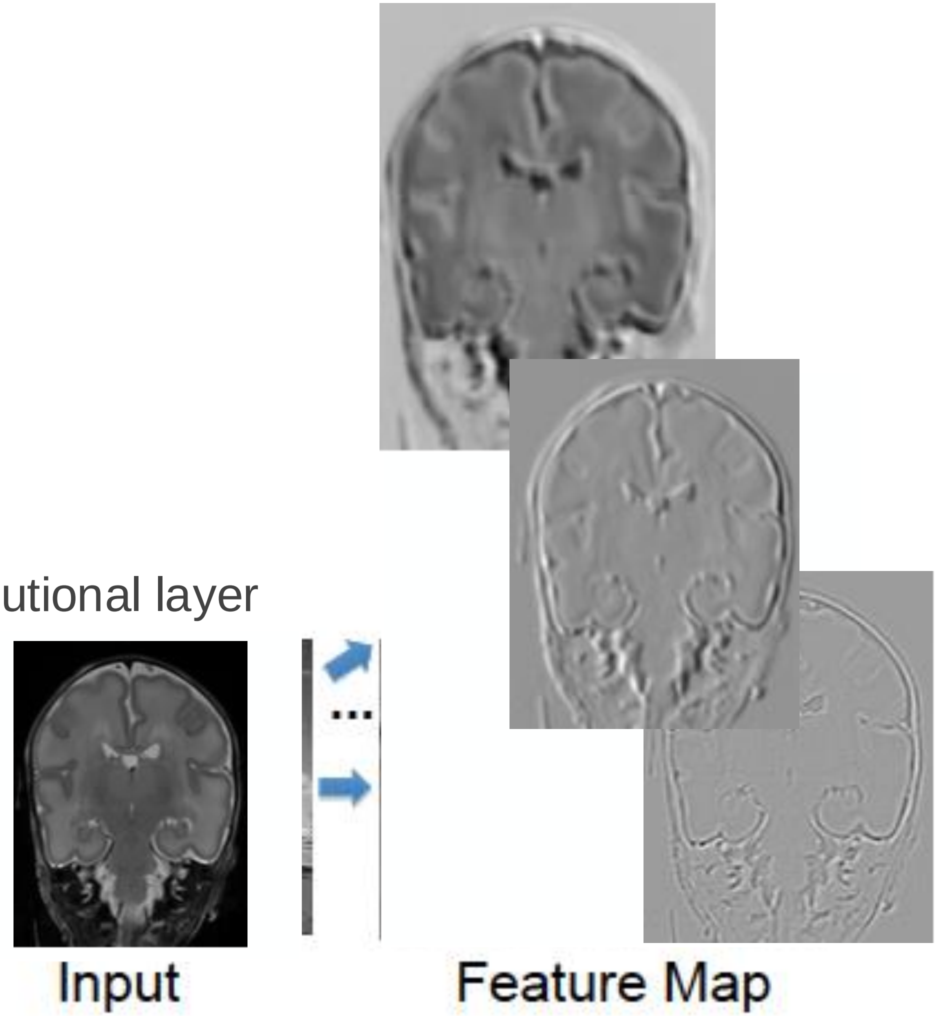
CNN

Next layer



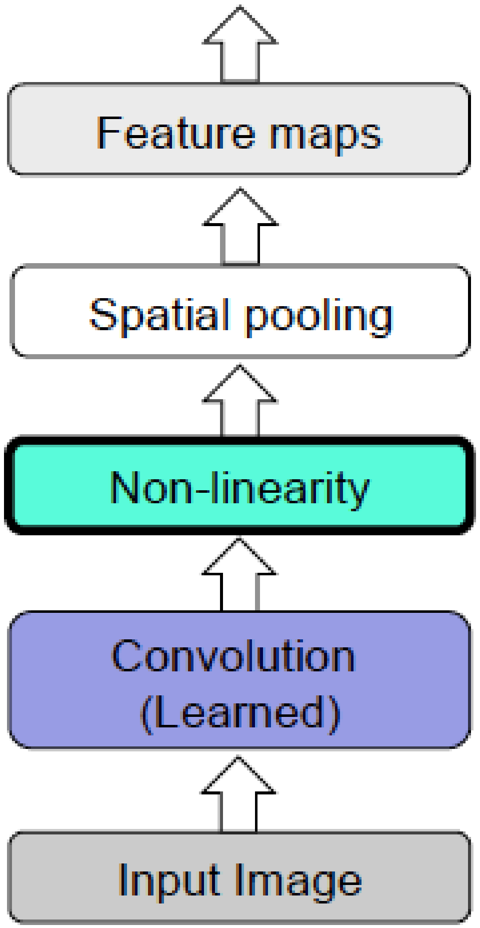
ReLU

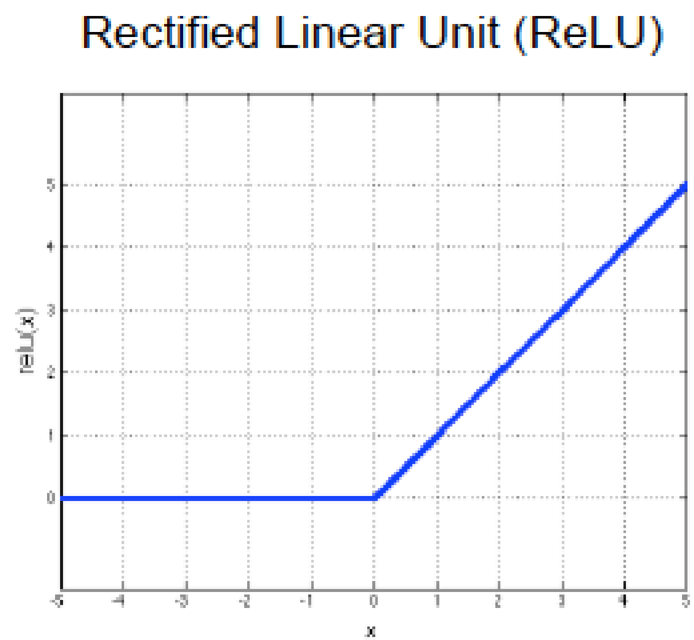
Convolutional layer



Source: R. Fergus, Y. LeCun

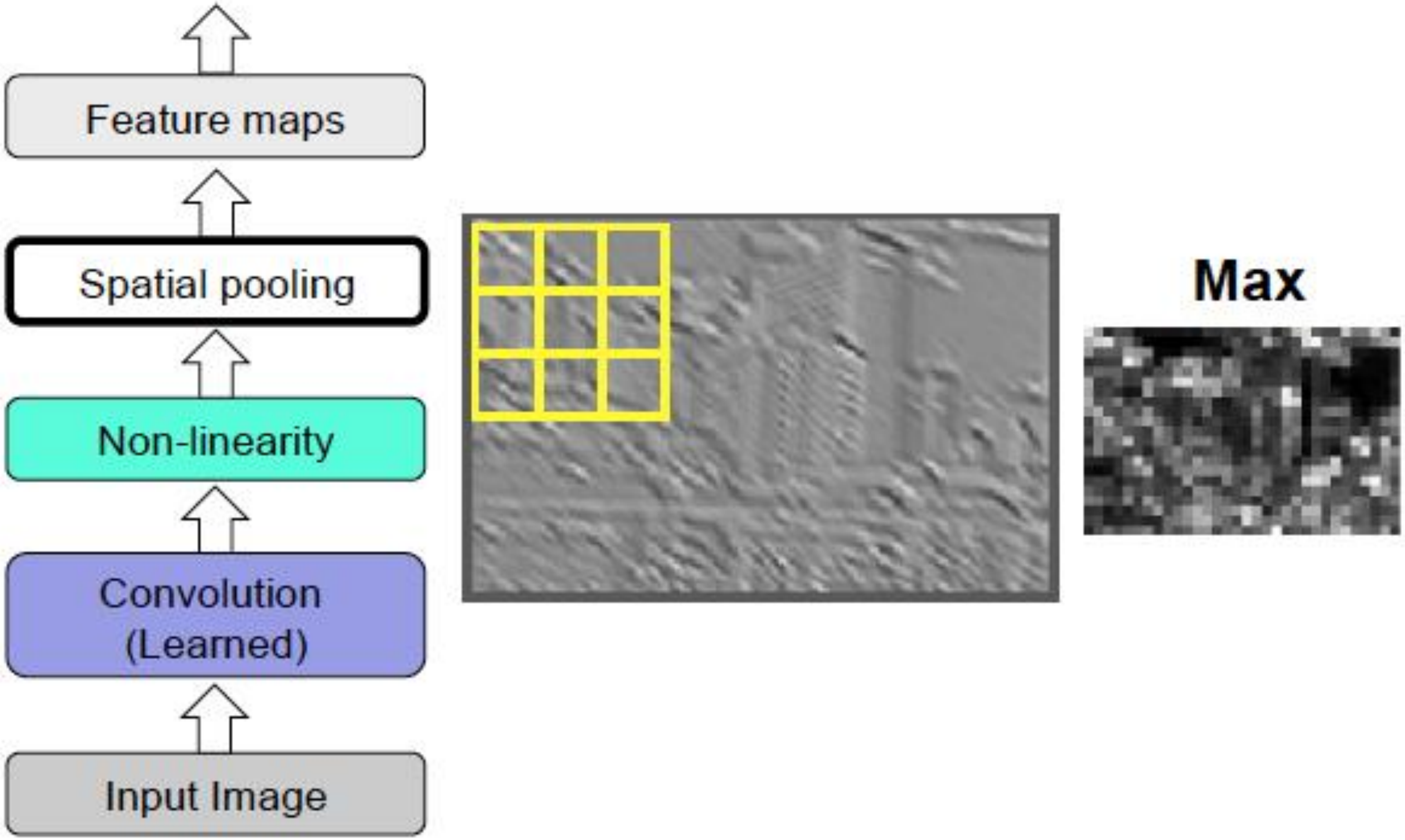
CNN





Source: R. Fergus, Y. LeCun

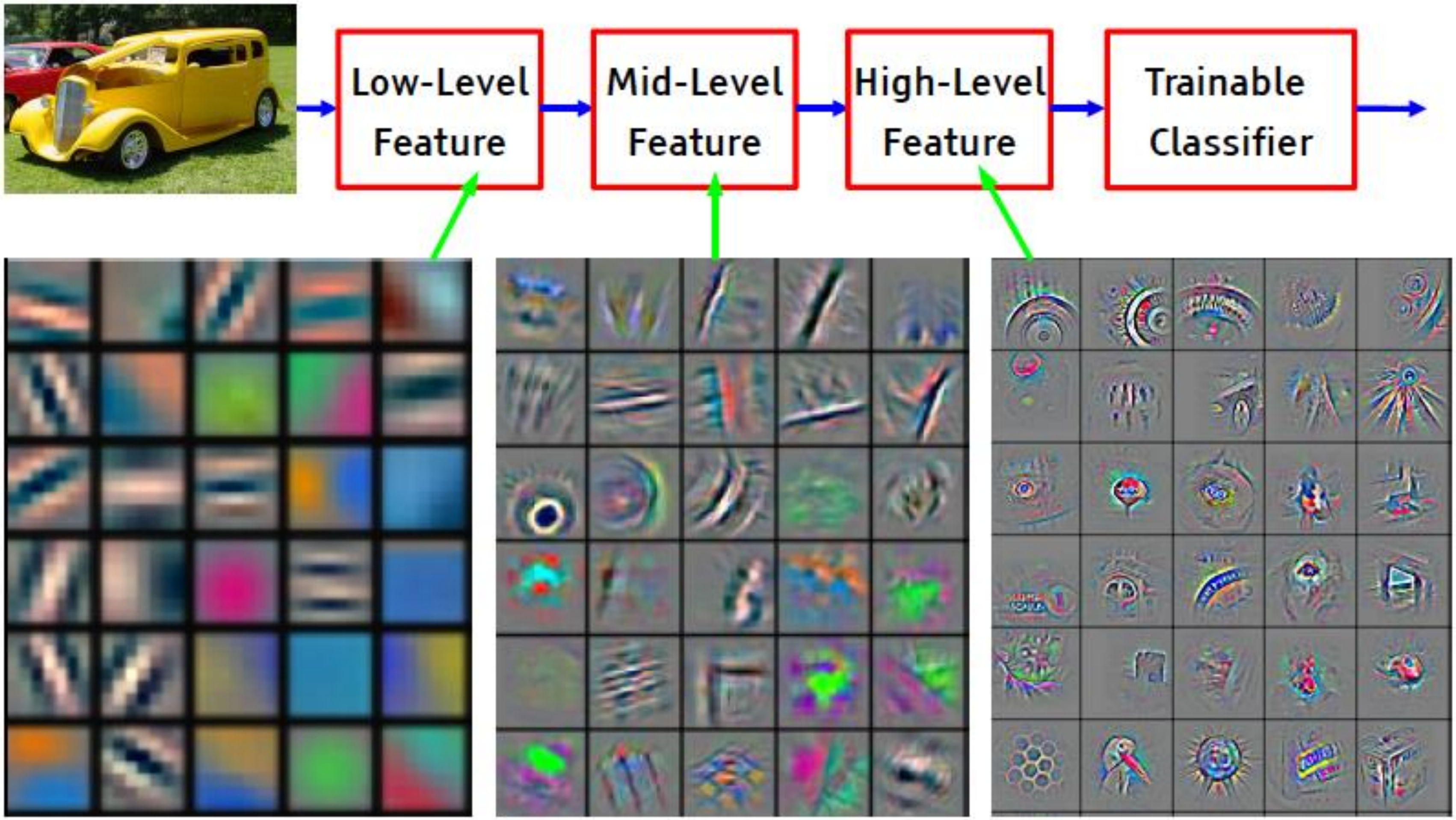
CNN



Source: R. Fergus, Y. LeCun

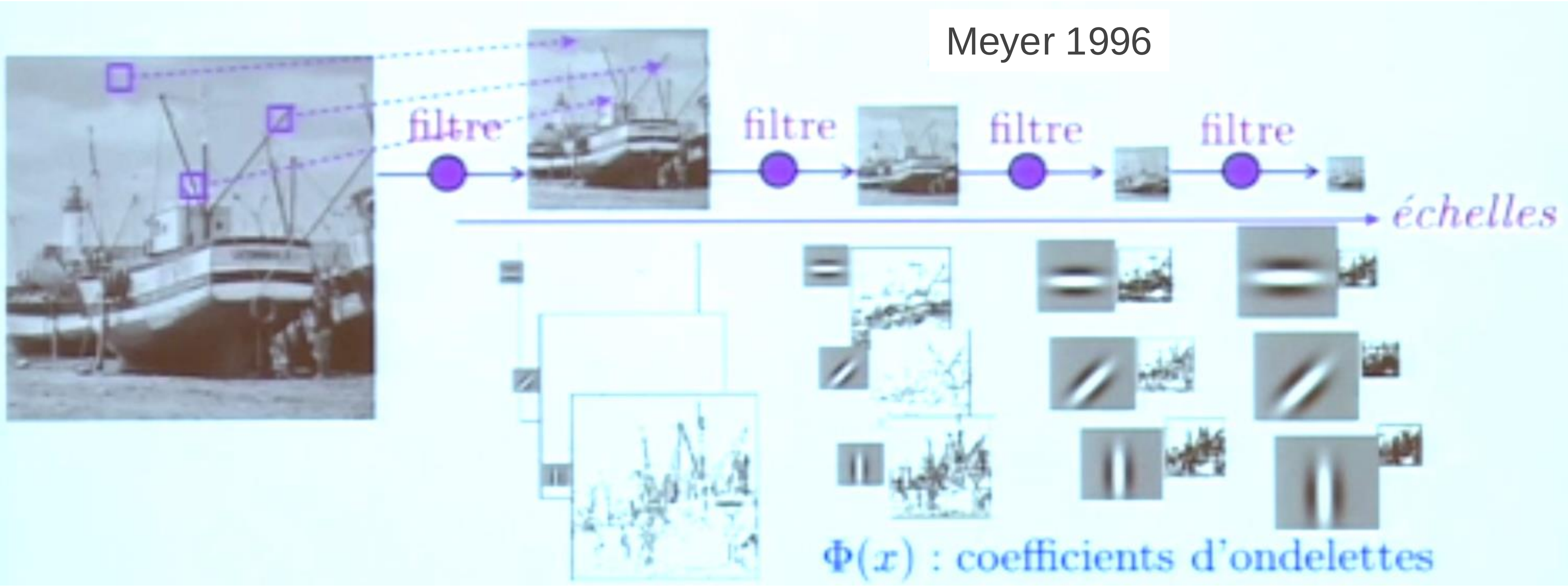
- **Step1:** We initialize all filters and parameters / weights with random values
- **Step2:** A training image as input, the forward propagation step
    - result [0.2, 0.4, 0.1, 0.3]
- **Step3:** Compute the total error at the output layer (summation over all 4 classes)
    - **Total Error = ∑ ½ (target probability – output probability)**
- **Step4:** Backpropagation to calculate the *gradients* of the error and use *gradient descent* to update all filter values / weights and parameter values to minimize the output error.
    - weights are adjusted in proportion to their contribution to the total error.
- **Step5:** Repeat steps 2-4 with all images in the training set. Training set is divided in n epochs of m examples (batches).

Hierarchical representation



From Y LeCun 2015 Open course CdF

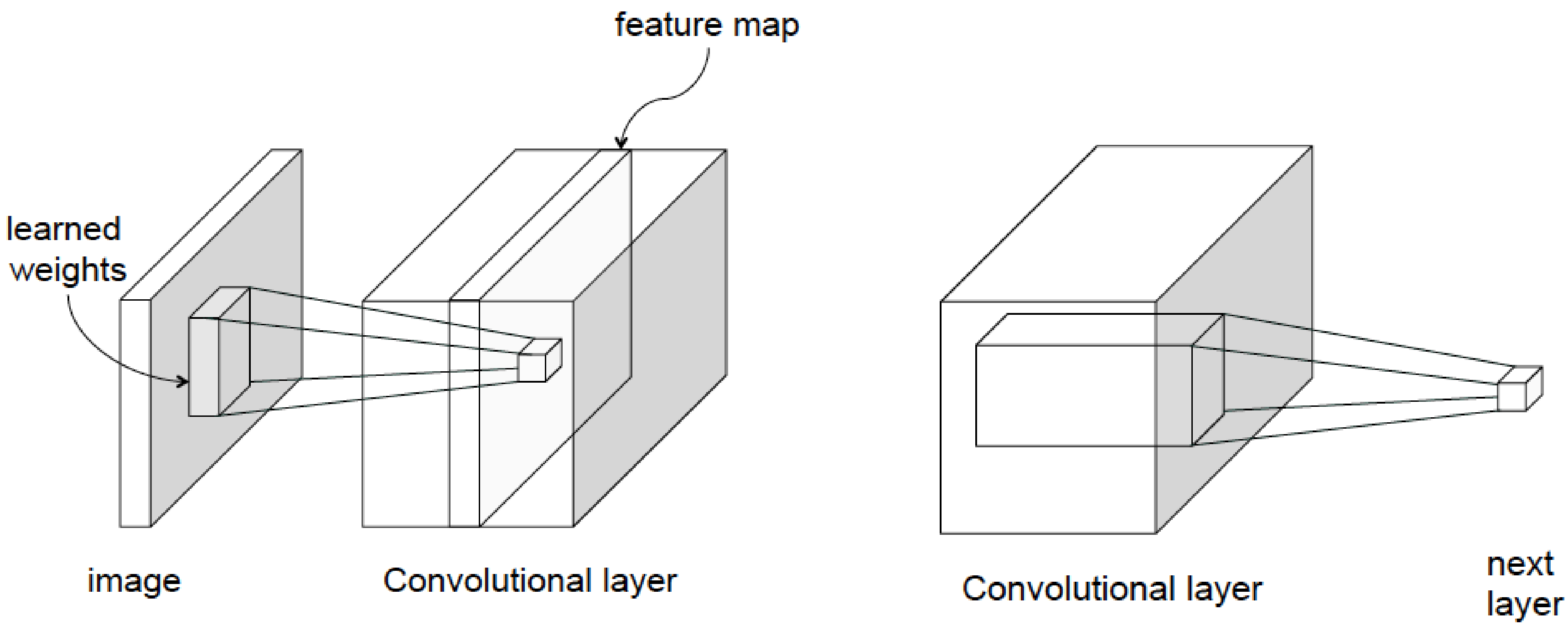
Multi-scale representation



From A. Maillard Open course CdF

Deep Learning

Lots of training data + Parallel Computation + Scalable, smart algorithms



AlexNet, 8 layers
(ILSVRC 2012)

15% errors on ImageNet

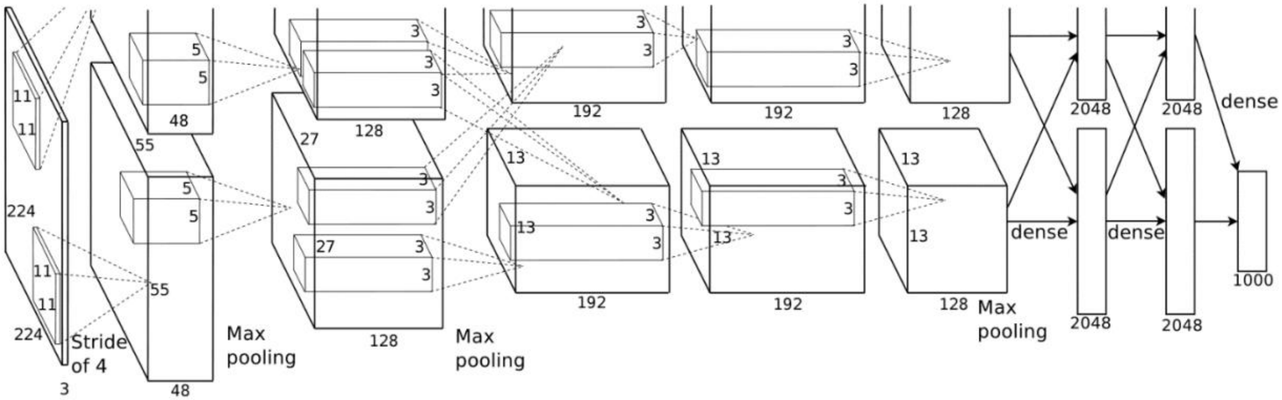
VGG, 19 layers
(ILSVRC 2014)

7% errors on ImageNet

ResNet, 152 layers
(ILSVRC 2015)

5% errors on ImageNet

Dedicated architectures



AlexNet

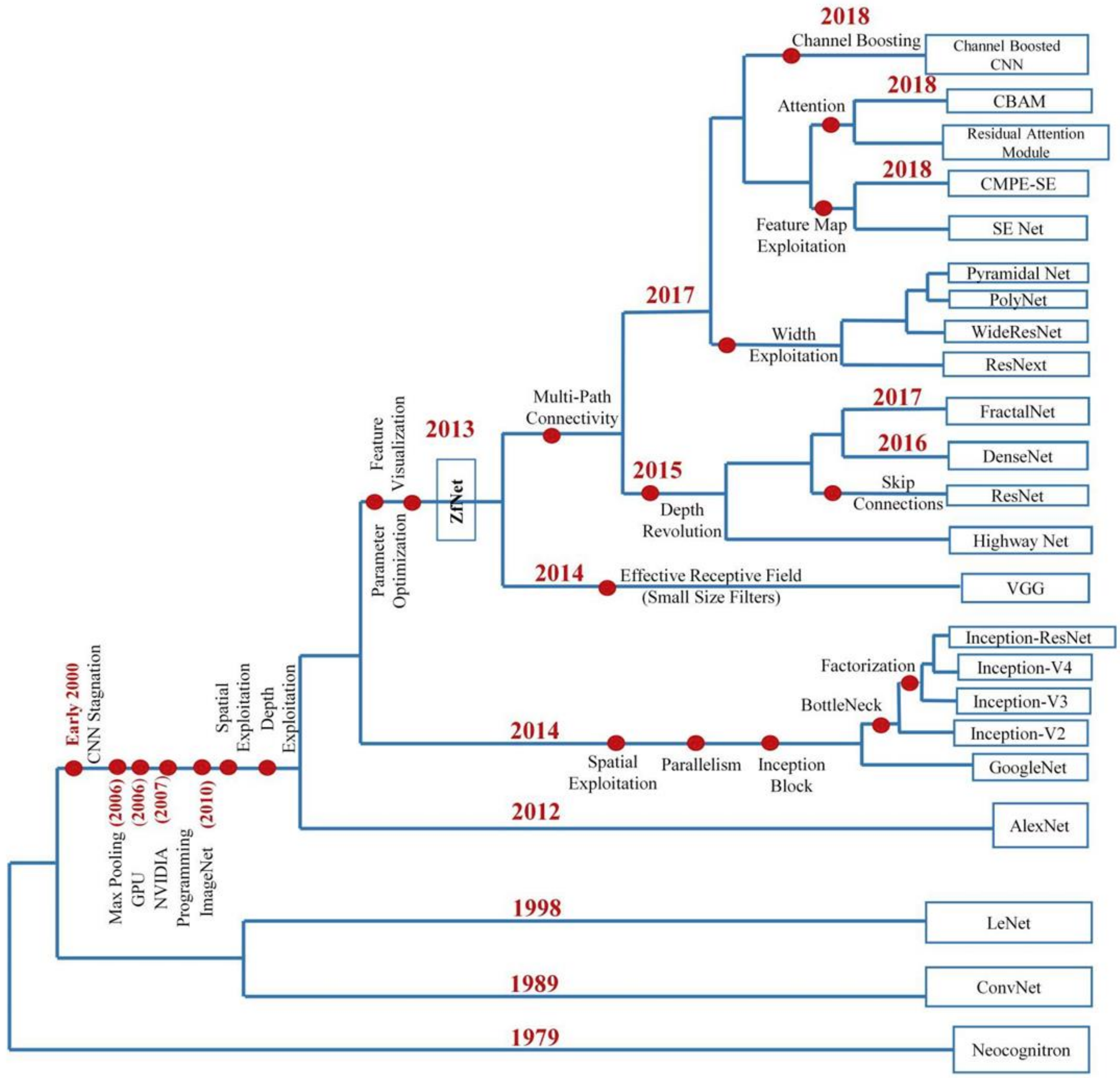
Network: AlexNet	Batch Size	Titan X (FP32)	Xeon E5-2698 v3 (FP32)
Inference Performance	1	405 img/sec	76 img/sec
Power		164.0 W	111.7 W
Performance/Watt		2.5 img/sec/W	0.7 img/sec/W
Inference Performance	128 (Titan X) 48 (Xeon E5)	3216 img/sec	476 img/sec
Power		227.0 W	149.0 W
Performance/Watt		14.2 img/sec/W	3.2 img/sec/W

Table 2 Inference performance, power, and energy efficiency on Titan X and Xeon E5-2698 v3.

White paper Nvidia 2015

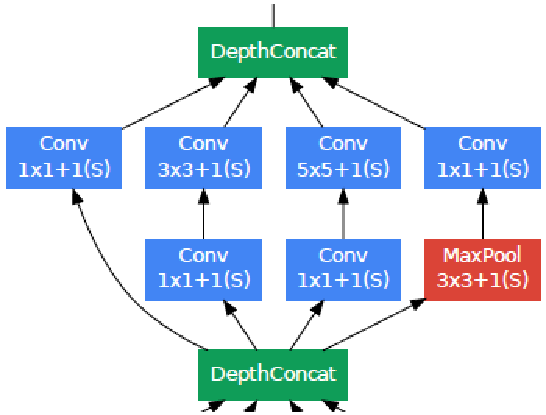
Evolution

Khan et al AT Rev 2020



Dedicated architectures

Inception module



GoogleNet

Network: GoogLeNet	Batch Size	Titan X (FP32)	Tegra X1 (FP32)	Tegra X1 (FP16)
Inference Performance	1	138 img/sec	33 img/sec	33 img/sec
Power		119.0 W	5.0 W	4.0 W
Performance/Watt		1.2 img/sec/W	6.5 img/sec/W	8.3 img/sec/W
Inference Performance	128 (Titan X) 64 (Tegra X1)	863 img/sec	52 img/sec	75 img/sec
Power		225.0 W	5.9 W	5.8 W
Performance/Watt		3.8 img/sec/W	8.8 img/sec/W	12.8 img/sec/W

Table 3 GoogLeNet inference results on Tegra X1 and Titan X. Tegra X1's total memory capacity is not sufficient to run batch size 128 inference.

Googlenet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

8 inception modules, 22 layers.

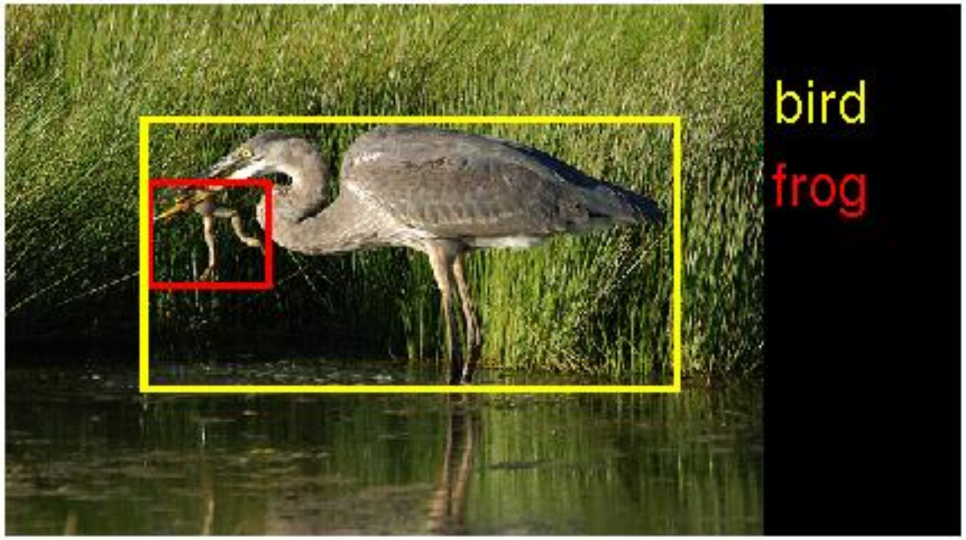
Impressive results

(a) Siberian husky

(b) Eskimo

Progress due to:

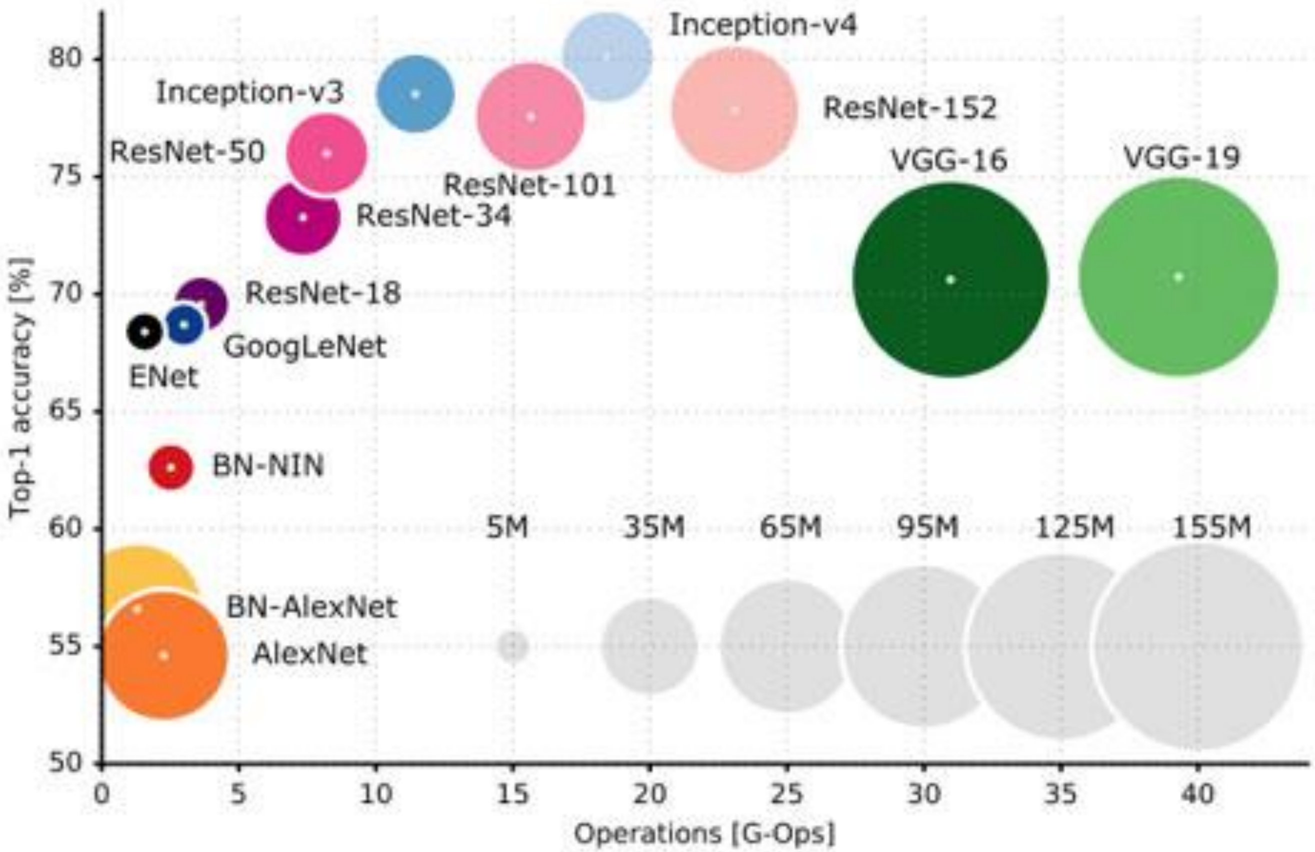
-Availability of large training sets (ImageNet Chall 1000 categories, 1.2 M images for training, 150000 for validation and for testing)



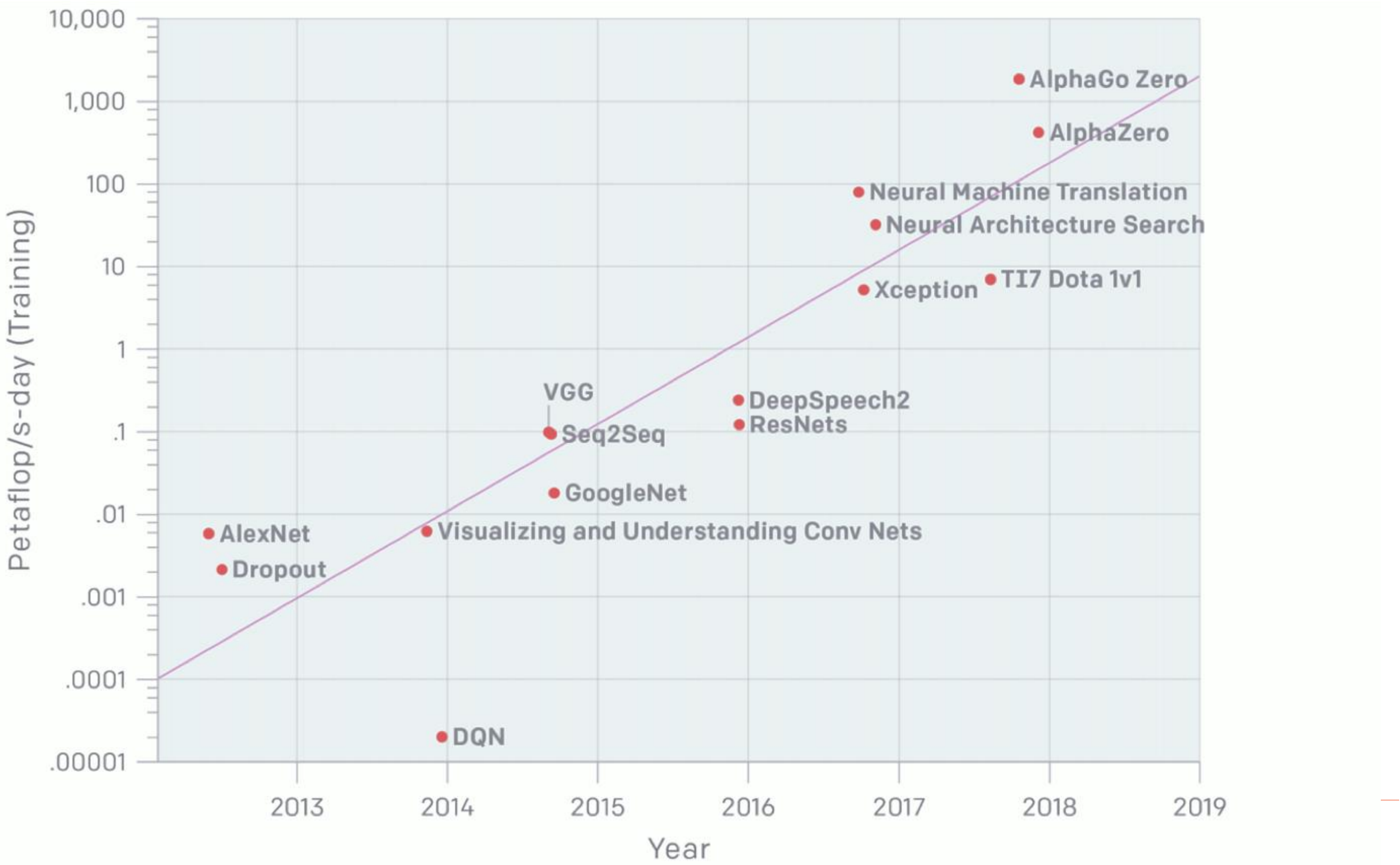
AND …

Szegedy et al. 2017

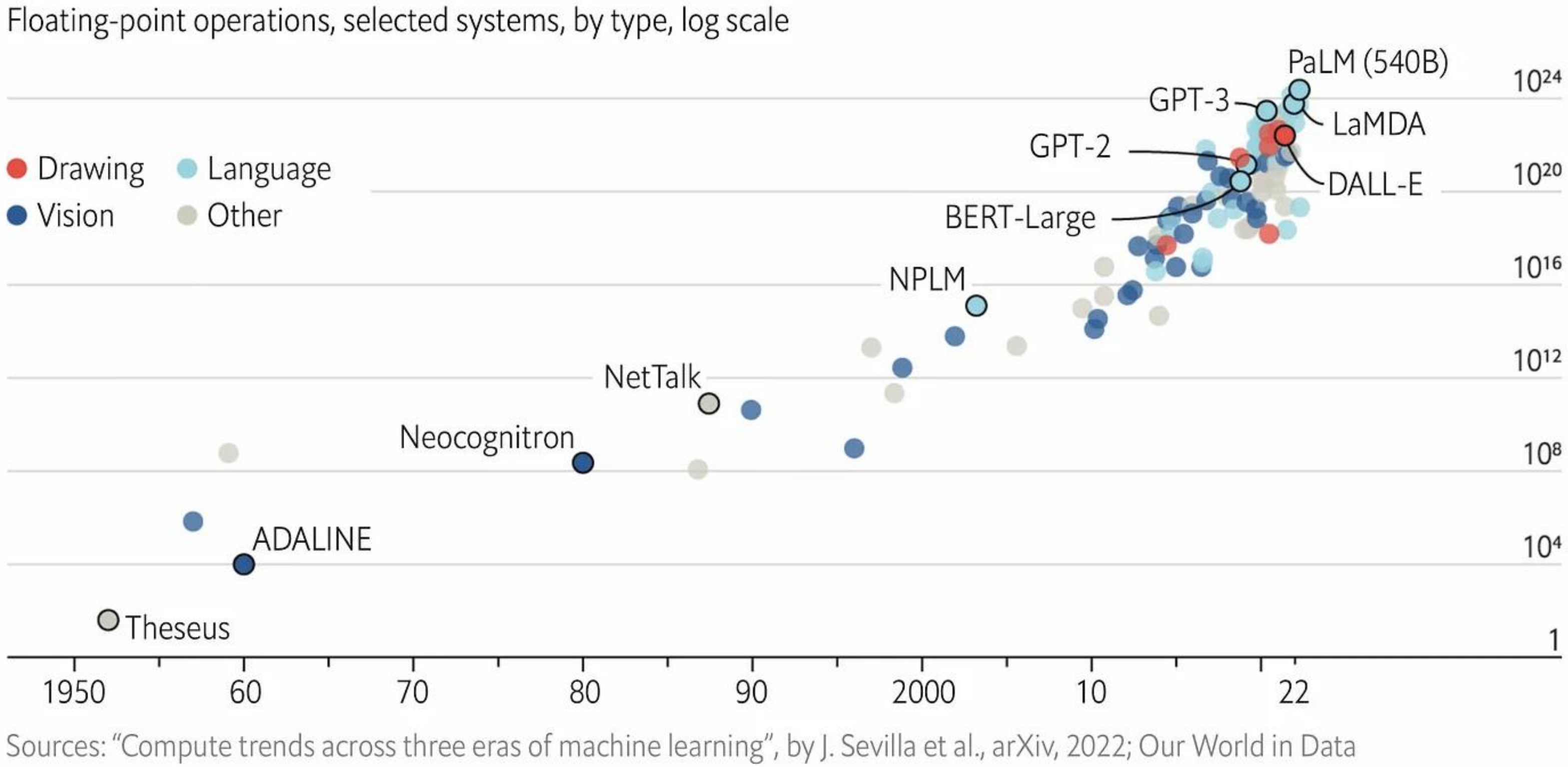
Dedicated Hardware



Computational burden-I

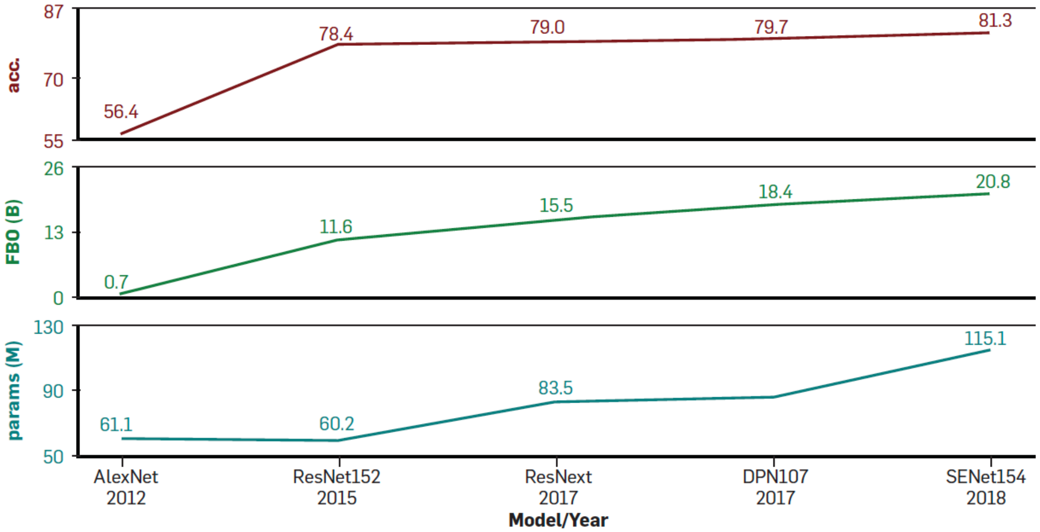


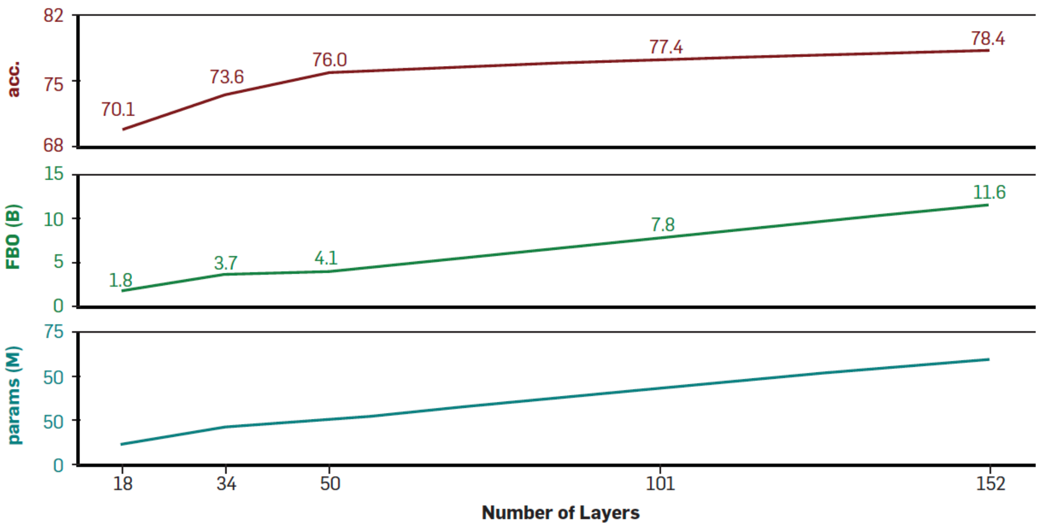
Computational burden-II



https://arxiv.org/abs/2202.05924

Mismatch Flops vs Accuracy



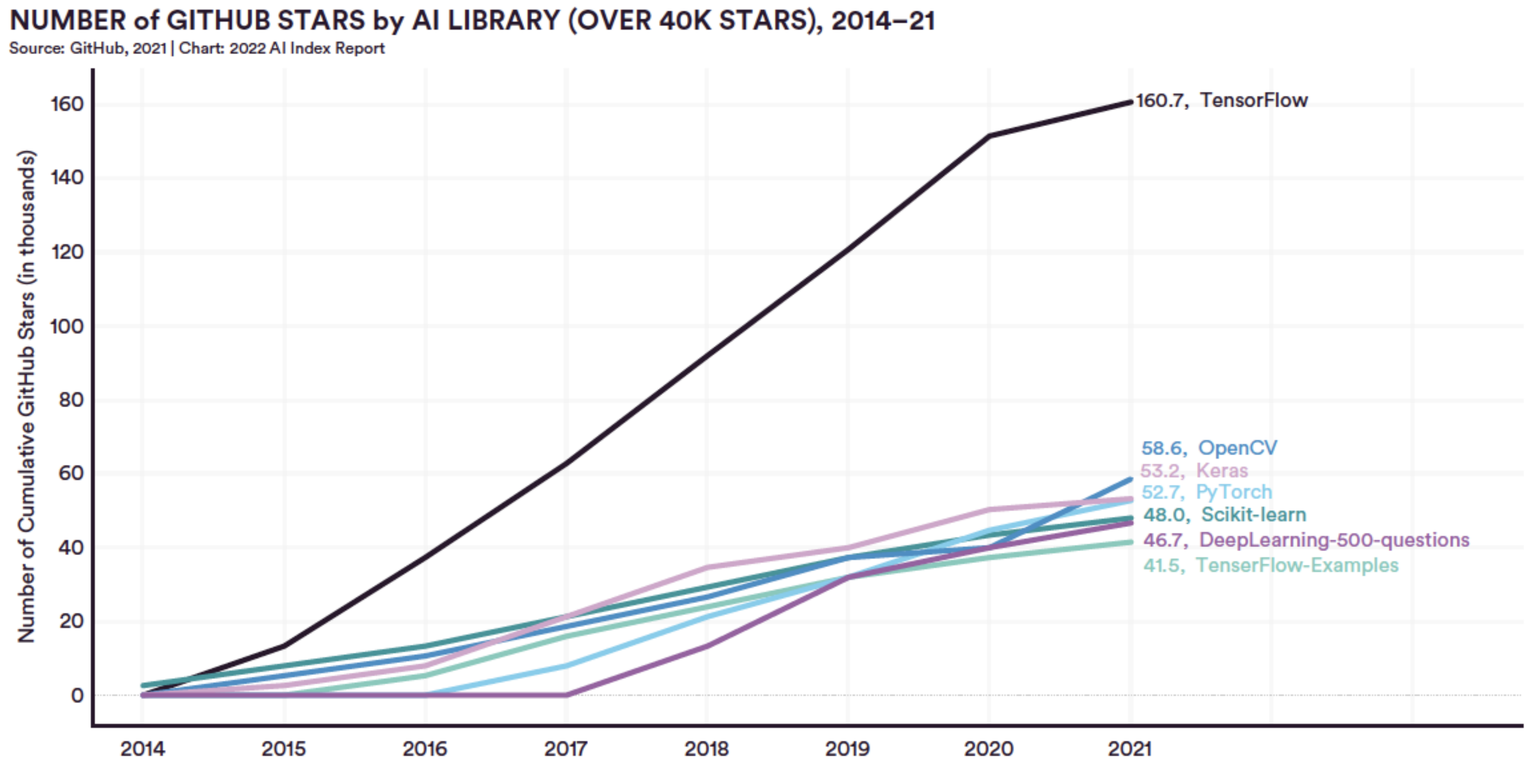


Schwartz et al. Acm 2020

Languages & Frameworks

Library	Rank	Overall	Github	Stack Overflow	Google Results
tensorflow	1	10.87	4.25	4.37	2.24
keras	2	1.93	0.61	0.83	0.48
caffe	3	1.86	1.00	0.30	0.55
theano	4	0.76	-0.16	0.36	0.55
pytorch	5	0.48	-0.20	-0.30	0.98
sonnet	6	0.43	-0.33	-0.36	1.12
mxnet	7	0.10	0.12	-0.31	0.28
torch	8	0.01	-0.15	-0.01	0.17
cntk	9	-0.02	0.10	-0.28	0.17
dlib	10	-0.60	-0.40	-0.22	0.02

From Knoll Ismrm 2018



AI index report 2022 Stanford Univ

Dedicated tools and languages

Torch7
http://torch.ch

```
for t = 1,trainData:size(),batchSize do
  inputs,outputs = getNextBatch()
  local feval = function(x)
    parameters:copy(x)
    gradParameters:zero()
    local f = 0
    for i = 1,#inputs do
      local output = model:forward(inputs[i])
      local err = criterion:forward(output,targets[i])
      f = f + err
      local df_do = criterion:backward(output,targets[i])
      model:backward(inputs[i], df_do)
    end
    gradParameters:div(#inputs)
    f = f/#inputs
    return f,gradParameters
  end  -- of feval
  optim.sgd(feval,parameters,optimState)
end
```

- one epoch over training set
- Get next batch of samples
- Create a "closure" feval(x) that takes the parameter vector as argument and returns the loss and its gradient on the batch.
- Run model on batch
- backprop
- Normalize by size of batch
- Return loss and gradient
- call the stochastic gradient optimizer

From Y LeCun 2015 cdfr

Dedicated tools and languages

Matlab

```
%% NN with 3 HIDDEN LAYER NEURONS
nElements = 100;
nLayers = 3;
inputLayer=imageInputLayer([nFeatures,1,1]);
f1=fullyConnectedLayer(nElements);
f2=fullyConnectedLayer(nElements);
f3=fullyConnectedLayer(nElements);
f4=fullyConnectedLayer(nClasses);
s1=softmaxLayer();
outputLayer=classificationLayer();

architecture = [inputLayer; f1; f2; f3; f4; s1; outputLayer];
disp(architecture);
```

```
epochs = 250;
miniBatchSize = 1024;
InitialLearnRate = 0.001;

% Training options: Note that we set the validation patience stopping
% criterion to the number of epochs. This is a stupid thing to do, but we
% want force the training to go to the defined number of epochs so that it
% is consistent with Tensorflow and Pytorch
options = trainingOptions('adam','MaxEpochs',epochs,'InitialLearnRate',InitialLearnRate,...
    'MiniBatchSize',miniBatchSize,'ExecutionEnvironment','cpu','Plots','training-progress',...
    'ValidationData',{x_val,y_val},'ValidationPatience',epochs);
```

```
%% Train
tic
[net,op] = trainNetwork(x_train,y_train,architecture,options);
toc
```

From Knoll Ismrm 2018

But …

“It is hard to give a definitive guidance to the most effective single way to train these networks” Szegedy et al 2015, CVPR

“Finding optimal parameters for each task, can be challenging”

Kamnitsas et al Media 2017

A rough rule of thumb is that the number of training samples for backpropagation should be 10 times the number of network parameters. Given that the number of parameters in a modern deep network far exceeds 100,000, the need for millions of training samples becomes evident, at least for current parameter learning strategies

Yamins and DiCarlo Nat Neurosc 2016

Caveats

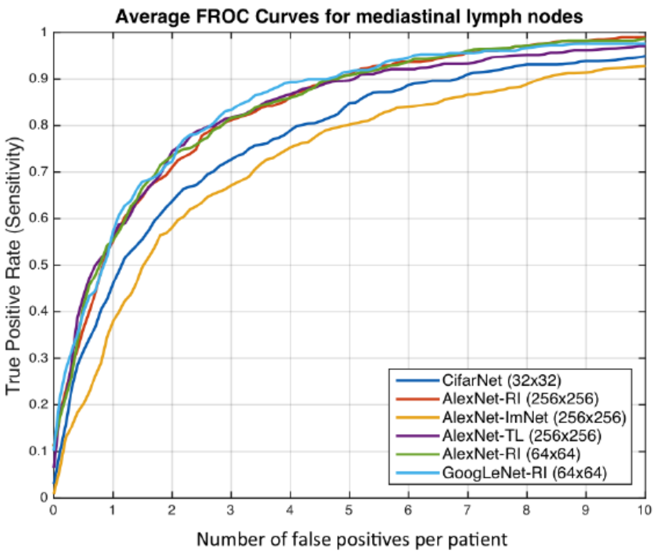
Deep Learning
Ian Goodfellow
Yoshua Bengio
Aaron Courville
MIT Press 2016

Hyperparameter	Increases capacity when...	Reason	Caveats
Number of hidden units	increased	Increasing the number of hidden units increases the representational capacity of the model.	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.
Learning rate	tuned optimally	An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure.	
Convolution kernel width	increased	Increasing the kernel width increases the number of parameters in the model.	A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect. Wider kernels require more memory for parameter storage and increase runtime, but a narrower output reduces memory cost.
Implicit zero padding	increased	Adding implicit zeros before convolution keeps the representation size large.	Increases time and memory cost of most operations.
Weight decay coefficient	decreased	Decreasing the weight decay coefficient frees the model parameters to become larger.	
Dropout rate	decreased	Dropping units less often gives the units more opportunities to "conspire" with each other to fit the training set.	

Key points

- Importance of the architecture :
  - 8-22 layers better than 2-3
- Trade-off between using better models and using more training data.
- Well annotated datasets is at least as crucial as developing new algorithms.

Shin et al TMI 2016



AI for Medical Imaging

- **Pros**:
  - Excellent performances
  - Automatic feature learning
  - Knowledge emergence
  - On the shelves tools
  - Discharge Expert
  - Automatic Quantification

- **Cons**:
  - Importance of Image Quality
  - Annotation
  - Data hungry
  - Computational cost
  - Black box / trustability
  - Specific to one problem
  - Adversarial attack
  - Catastrophic forgetting
  - Ethic, social and law
  - Needs for specific tools &infra

Not everything is provable

ARTICLES
https://doi.org/10.1038/s42256-018-0002-3

nature machine intelligence
Jv 2019

Corrected: Author Correction

Learnability can be undecidable

Shai Ben-David[1], Pavel Hrubeš[2], Shay Moran[3], Amir Shpilka[4] and Amir Yehudayoff[5]*

The mathematical foundations of machine learning play a key role in the development of the field. They improve our understanding and provide tools for designing new learning paradigms. The advantages of mathematics, however, sometimes come with a cost. Gödel and Cohen showed, in a nutshell, that not everything is provable. Here we show that machine learning shares this fate. We describe simple scenarios where learnability cannot be proved nor refuted using the standard axioms of mathematics. Our proof is based on the fact the continuum hypothesis cannot be proved nor refuted. We show that, in some cases, a solution to the 'estimating the maximum' problem is equivalent to the continuum hypothesis. The main idea is to prove an equivalence between learnability and compression.