Multimedia Indexing and Retrieval

Deep Learning for multimedia indexing and retrieval

Georges Quénot

Multimedia Information Modeling and Retrieval Group

Laboratory of Informatics of Grenoble

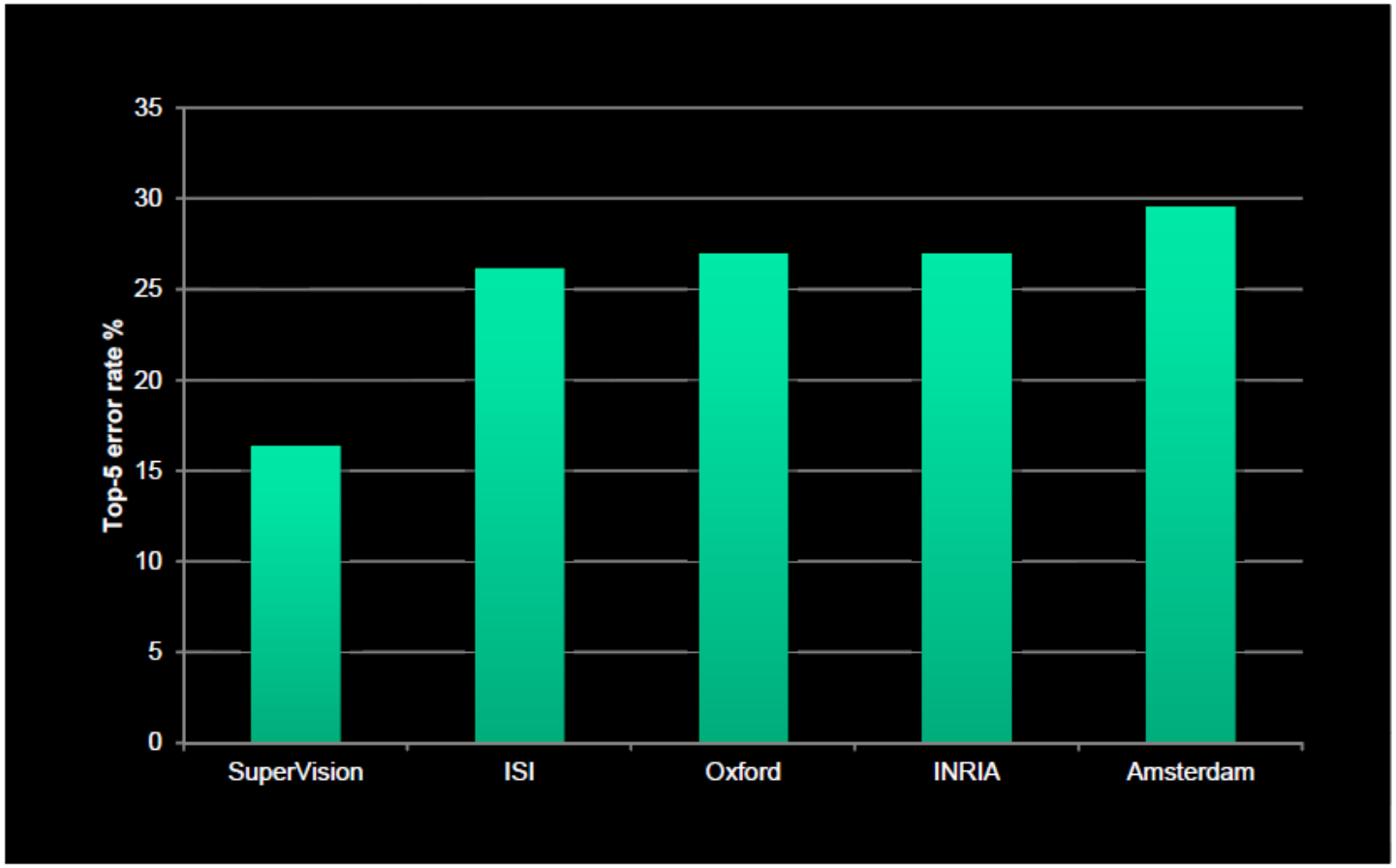
Outline

- Introduction
- Reminders about differential calculus
- Machine learning
- Loss function
- Formal neuron
- Single layer perceptron
- Multilayer perceptron
- Back-propagation
- Learning rate
- Mini-batches
- Convolutional layers
- Pooling, softmax …

2012

ImageNet Classification 2012 Results

Krizhevsky et al. – **16.4% error** (top-5)
Next best (Pyr. FV on dense SIFT) – **26.2% error**



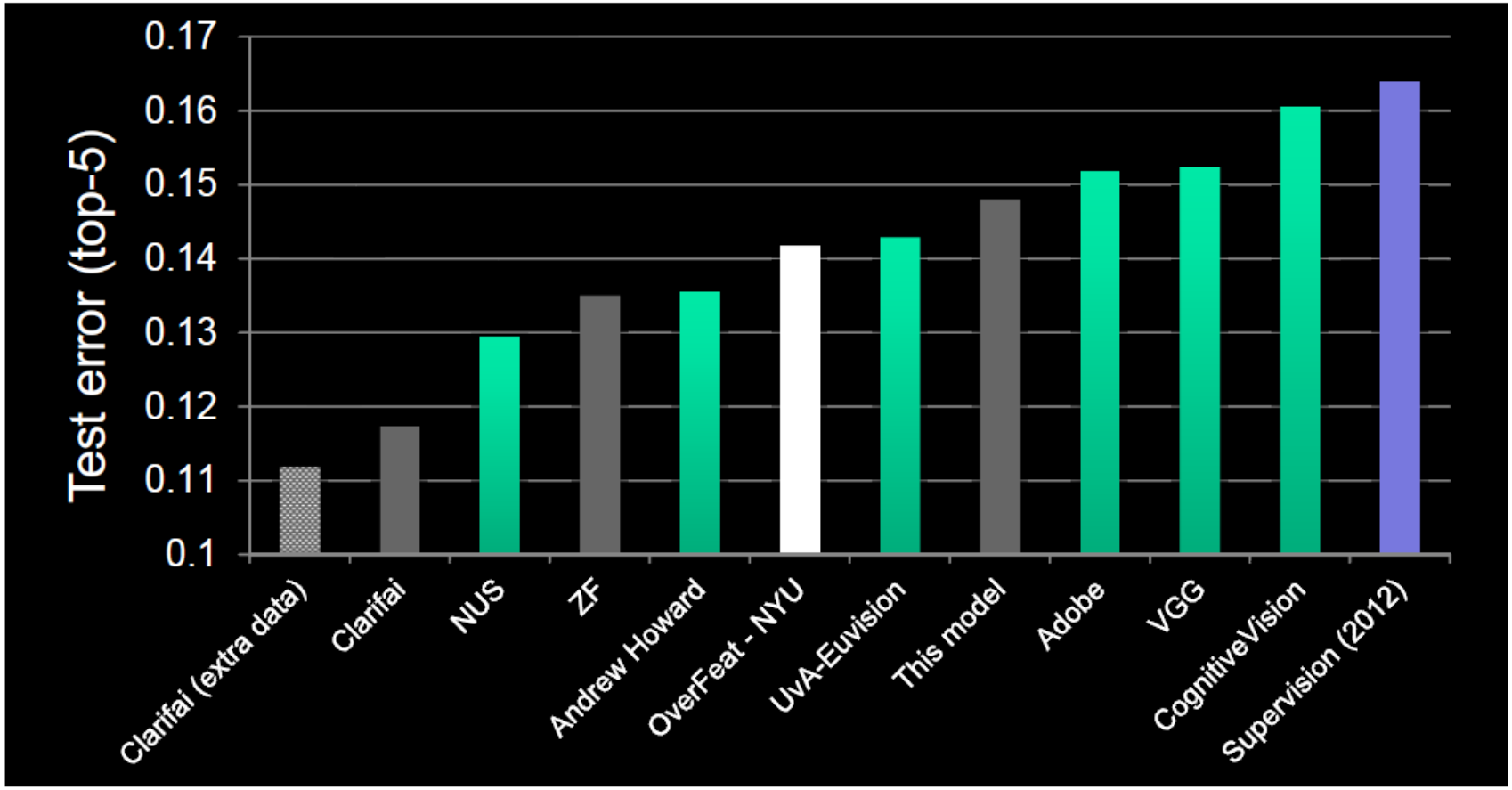
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

- 1000 visual “fine grain” categories / labels (exclusive)
- 150,000 test images (hidden “ground truth”)
- 50,000 validation images
- 1,200,000 training images
- Each training, validation or test image falls within exactly one of the 1000 categories
- Task: for each image in the test set, rank the categories from most probable to least probable
- Metric: top-5 error rate: percentage of images for which the actual category is not in the five first ranked categories
- Held from 2010 to 2015, frozen since 2012

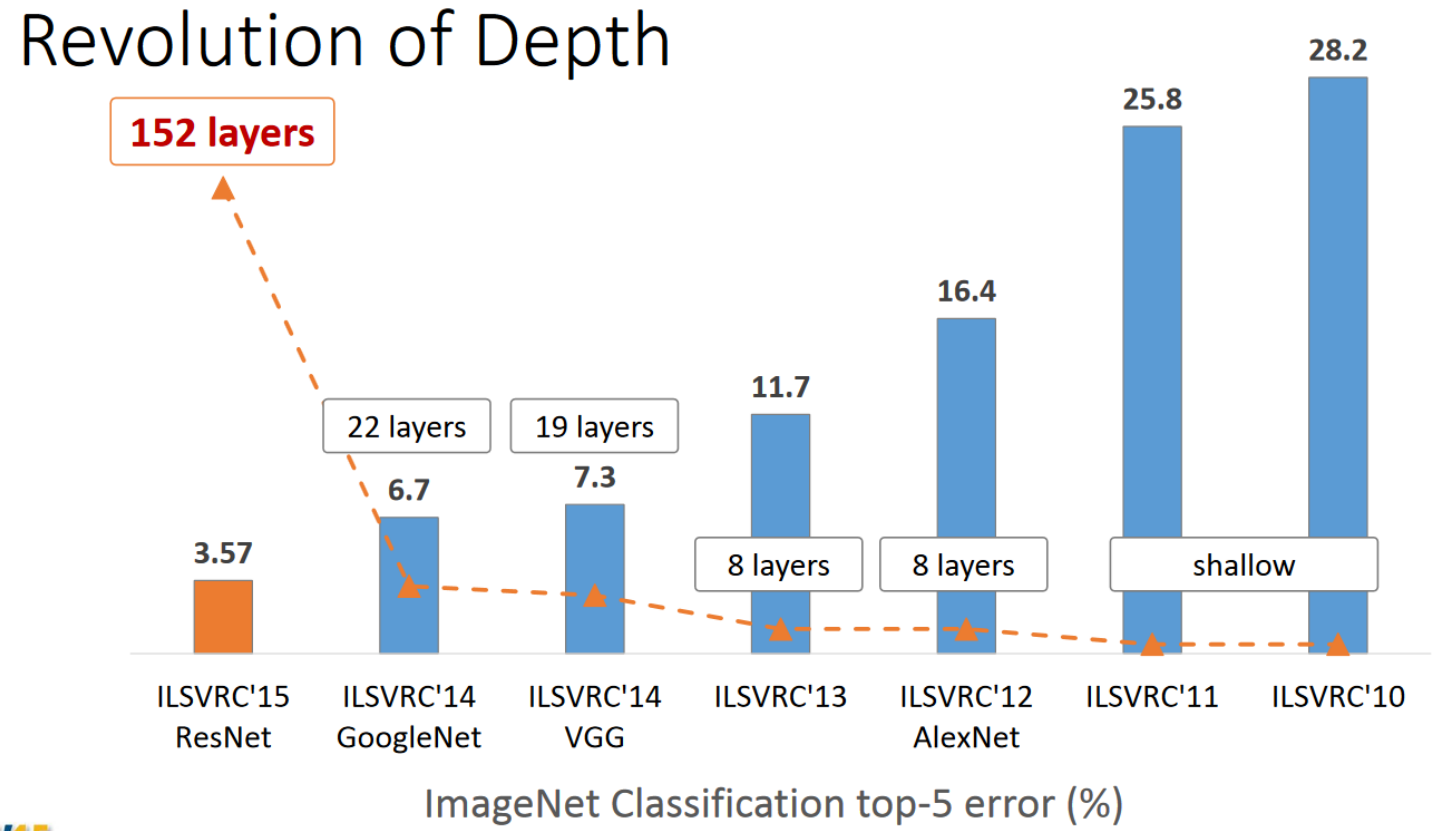
ImageNet Classification 2013 Results

http://www.image-net.org/challenges/LSVRC/2013/results.php

Demo: http://www.clarifai.com/



Going deeper and deeper



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

For comparison, human performance is 5.1% (Russakovsky et al.)

Deep Convolutional Neural Networks

- Decades of algorithmic improvements in neural networks (Stochastic Gradient Descent, initialization, momentum …)
- Very large amounts of properly annotated data (ImageNet)
- Huge computing power (Teraflops × weeks): GPU!
- Convolutional networks
- Deep networks (>> 3 layers)
- ReLU (Rectified Linear Unit) activation functions
- Batch normalization
- Drop Out
- …

Differential of a function scalar input and scalar output

- $f : \mathbb{R} \rightarrow \mathbb{R} : x \rightarrow f(x)$ $\qquad$ f is differentiable
- $y = f(x)$
- $f(x) - f(x+h) = f'(x)h + o(h)$ $\qquad$ $(\lim_{h \rightarrow 0} \frac{o(h)}{h} = 0)$
- $dy = f'(x)dx$
- $\frac{dy}{dx} \equiv f'(x)$ $\qquad$ (notation)
- $dy = \frac{dy}{dx} dx$
- All values are scalar

Differential of a composed function scalar input and scalar output

- $f : \mathbb{R} \rightarrow \mathbb{R} : x \rightarrow f(x)$ $\qquad$ f is differentiable
- $y = f(x)$
- $g : \mathbb{R} \rightarrow \mathbb{R} : y \rightarrow g(y)$ $\qquad$ g is differentiable
- $z = g(y)$
- $(g \; o \; f)'(x) = (g' o \; f)(x).f'(x) = g'(y).f'(x)$
- $dy = \frac{dy}{dx} dx$ $\qquad$ $dz = \frac{dz}{dy} dy$
- $dz = \frac{dz}{dy}.\frac{dy}{dx} dx$ $\qquad$ $\frac{dz}{dx} = \frac{dz}{dy}.\frac{dy}{dx}$

Differential of a function of a vector vector input and scalar output

- $f : \mathbb{R}^N \to \mathbb{R} : x \to f(x)$ $\qquad$ f is differentiable
- $y = f(x)$ $\qquad$ $x = (x_i)_{(1 \le i \le N)}$
- $f(x) - f(x+h) = \text{grad}\, f\,(x).h + o(\|h\|)$
- $dy = \text{grad}\, f\,(x).dx = \sum_{i=1}^{i=n} \frac{\partial f}{\partial x_i}(x).dx_i = \sum_{i=1}^{i=n} \frac{\partial y}{\partial x_i}.dx_i = \frac{\partial y}{\partial x}.dx$
- $\frac{\partial y}{\partial x} \equiv \frac{\partial f}{\partial x}(x) = \text{grad}\, f\,(x)$ $\qquad$ $\frac{\partial y}{\partial x_i} \equiv \frac{\partial f}{\partial x_i}(x)$ $\qquad$ (notations)

- y, dy and $f(x)$ are scalars;
- x, dx and h are “regular” (column) vectors;
- $\frac{\partial y}{\partial x}$ is a transpose (row) vector.

Differential of a vector function of a vector vector input and vector output

- $f : \mathbb{R}^N \to \mathbb{R}^P : x \to f(x)$ $\qquad$ f is differentiable
- $y = f(x)$ $\qquad$ $x = (x_i)_{(1 \le i \le N)}$ $\quad$ $y = (y_j)_{(1 \le j \le P)}$ $\quad$ $f = (f_j)_{(1 \le j \le P)}$
- $f(x) - f(x+h) = \mathrm{grad}\, f\,(x).h + o(\|h\|)$
- $dy = \mathrm{grad}\, f\,(x).dx = \frac{\partial f}{\partial x}(x).dx = \frac{\partial y}{\partial x}.dx$
- $dy_j = \sum_{i=1}^{i=n} \frac{\partial f_j}{\partial x_i}(x).dx_i = \sum_{i=1}^{i=n} \frac{\partial y_j}{\partial x_i}.dx_i$

- x, dx, y, dy, $f(x)$ and h are all “regular” vectors;
- $\frac{\partial y}{\partial x}$ is a matrix (Jacobian of f: $J_{ij} = \left(\frac{\partial y}{\partial x}\right)_{ij} = \frac{\partial y_j}{\partial x_i} = \frac{\partial f_j}{\partial x_i}(x)$).

Differential of a composed function vector inputs and vector outputs

- $f : \mathbb{R}^N \to \mathbb{R}^P : x \to y = f(x)$ $\quad$ f is differentiable
- $g : \mathbb{R}^P \to \mathbb{R}^Q : y \to z = g(y)$ $\quad$ g is differentiable
- $x = (x_i)_{(1 \le i \le N)}$ $\quad$ $y = (y_j)_{(1 \le j \le P)}$ $\quad$ $z = (z_k)_{(1 \le k \le Q)}$
- $dy = \frac{\partial z}{\partial y} . \frac{\partial y}{\partial x} . dx$
- $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} . \frac{\partial y}{\partial x}$ $\quad$ (matrix multiplication: non commutative!)
- x, dx, y, dy, z, dz, $f(x)$ and $g(y)$ are all regular vectors;
- $\frac{\partial y}{\partial x}$, $\frac{\partial z}{\partial y}$ and $\frac{\partial z}{\partial x}$ are matrices (f, g and gof Jacobians).

Differential of a composed function vector inputs and scalar output

- $f : \mathbb{R}^N \to \mathbb{R}^P : x \to y = f(x)$ $\quad$ f is differentiable
- $g : \mathbb{R}^P \to \mathbb{R} \quad : y \to z = g(y)$ $\quad$ g is differentiable
- $x = (x_i)_{(1 \le i \le N)}$ $\quad$ $y = (y_j)_{(1 \le j \le P)}$ $\quad$ $z \in \mathbb{R}$
- $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} . \frac{\partial y}{\partial x}$ $\quad$ (left row vector × matrix mult. → row vector)
- z, dz and $g(y)$ are scalars;
- x, dx, y, dy, and $f(x)$ are regular vectors;
- $\frac{\partial z}{\partial y}$ and $\frac{\partial z}{\partial x}$ are transpose (row) vectors (f and gof gradients);
- $\frac{\partial y}{\partial x}$ is a matrix (f Jacobian).

Supervised learning

- Target function: $f : X \rightarrow Y$

$$x \rightarrow y = f(x)$$

  - x : input object (typically vector)
  - y : desired output (continuous value or class label)
  - X : set of valid input objects
  - Y : set of possible output values

- Training data: $S = (x_i, y_i)_{(1 \leq i \leq I)}$
  - I : number of training samples

- Learning algorithm: $L : (X \times Y)^* \rightarrow Y^X$

$$S \rightarrow f = L(S)$$

- Regression or classification system: $y = f(x) = [L(S)](x) = g(S,x)$

$$((X \times Y)^* = \cup_{n \in N} (X \times Y)^n)$$

Single-label loss function

- Quantifies the cost of classification error or the "empirical risk"
- Example (Mean Square Error): $E_S(f) = \sum_{i=1}^{i=I}(f(x_i) - y_i)^2$
- If f depends on a parameter vector θ (L learns θ):
  $E_S(\theta) = \frac{1}{2}\sum_{i=1}^{i=I}(f(\theta, x_i) - y_i)^2$
- For a linear SVM with soft margin, $\theta = (w, b)$:
  $E_S(\theta) = \frac{1}{2}\|w\|^2 + C.\sum_{i=1}^{i=I}\max(0,1 - y_i(w^T x_i + b))$
- The learning algorithm aims at minimizing the empirical risk: $\theta^* = \underset{\theta}{\operatorname{argmin}}\ E_S(\theta)$

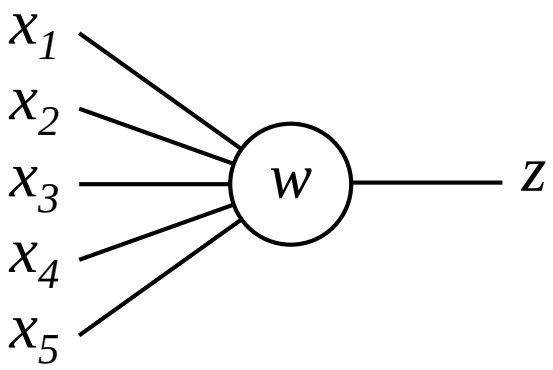
Multi-label loss function

- Predict P labels for each data sample x
- P decision functions : $f = (f_p)_{(1 \leq p \leq P)}$
- Example with f depending on a parameter vector:

  $E_S(\theta) = \frac{1}{2}\sum_{i=1}^{i=I}\sum_{p=1}^{p=P}(f_p(\theta, x_i) - y_{ip})^2 = \frac{1}{2}\sum_{i=1}^{i=I}(f(\theta, x_i) - y_i)^2$

  (same as single label case with Euclidean distance between vectors of predictions and vectors of labels)
- $\theta^* = \underset{\theta}{\operatorname{argmin}}\ E_S(\theta)$
- The f_p functions may take any real value

Formal neural or unit



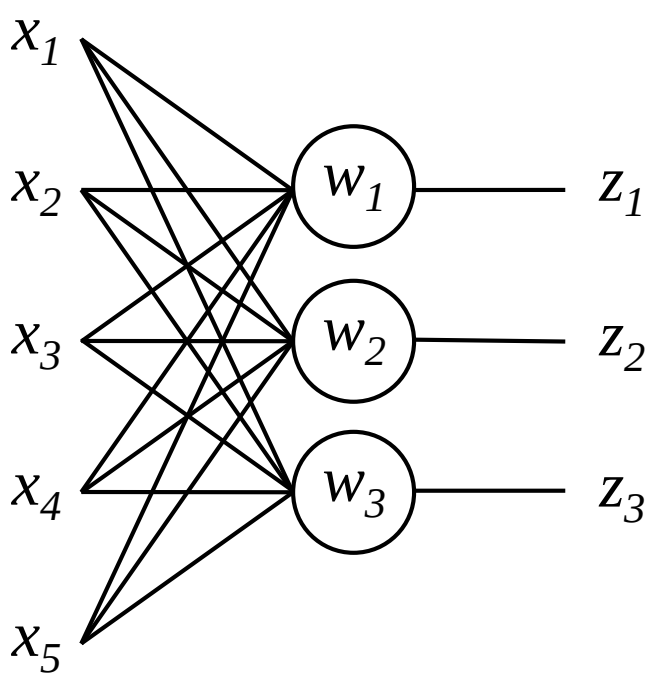
$$y = \sum_j w_j x_j$$

linear combination

$$z = \frac{1}{1 + e^y}$$

sigmoid function

Neural layer (all to all)



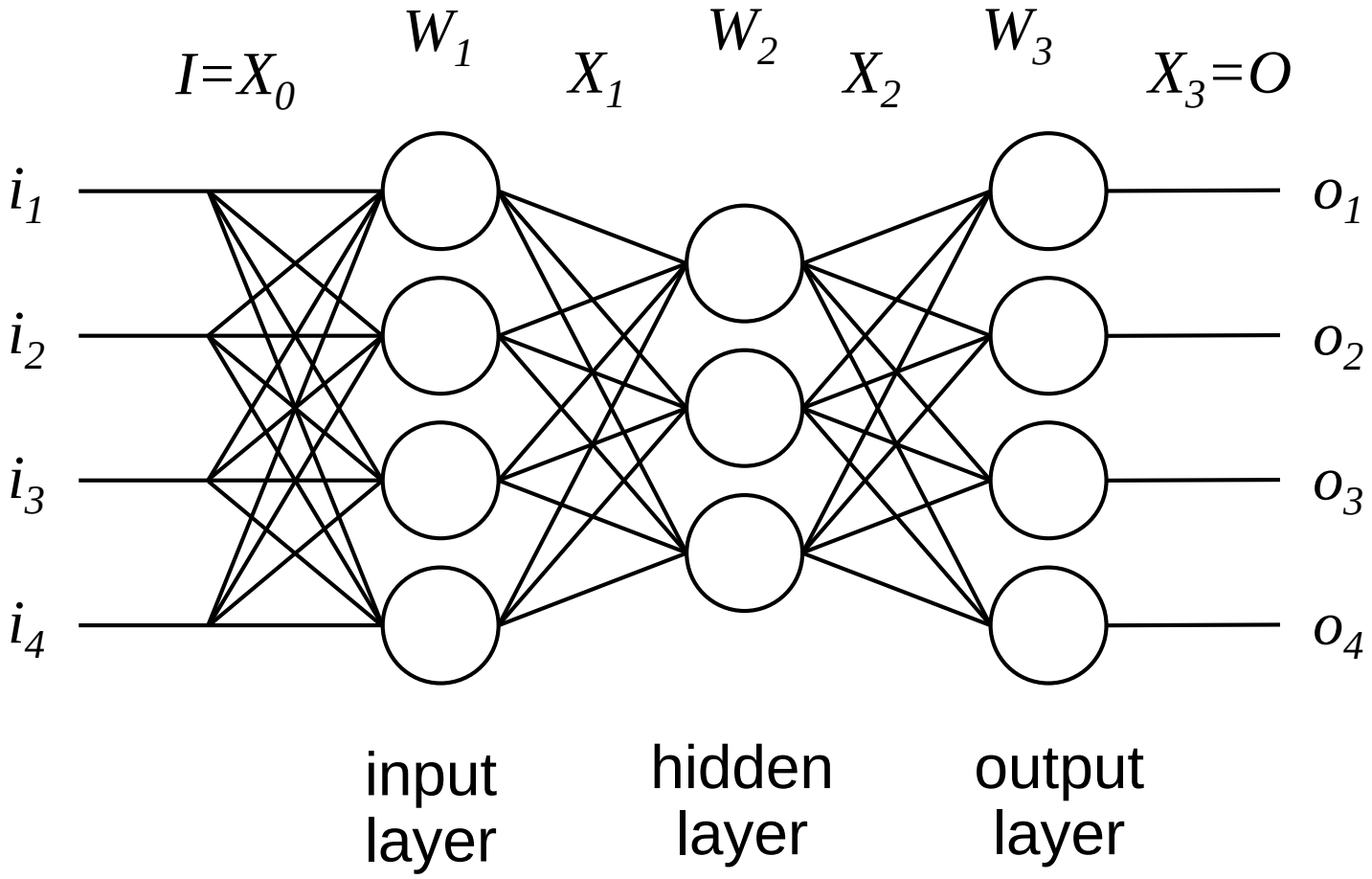
$$y_i = \sum_j w_{ij} x_j$$

matrix-vector multiplication

$$z_i = \frac{1}{1 + e^{y_i}}$$

per component operation

Multilayer perceptron (all to all)



Feed forward

- Global network definition: $O = F(W, I)$
  ($I \equiv x$ $O \equiv y$ $F \equiv f$ $W \equiv \theta$ relative to previous notations)

- Layer values: $(X_0, X_1 \ldots X_N)$
  with $X_0 = I$ and $X_N = O$ (X_n are vectors)

- Vector of all unit parameters:
  $W = (W_1, W_2 \ \ldots \ W_N)$
  (weights by layer concatenated, W_n are matrices)

- Feed forward: $X_{n+1} = F_{n+1}(W_{n+1}, X_n)$

Error back-propagation

- Training set: $S = (I_i, O_i)_{(1 \leq i \leq I)}$ input-output samples
- $X_{i,0} = I_i$ and $X_{i,n+1} = F_{n+1}\left(W_{n+1}, X_{i,n}\right)$
- Note: regarding this notation the vector-matrix multiplication counts as one layer and the element-wise non-linearity counts as another one (not mandatory but greatly simplifies the layer modules' implementation)
- Error (empirical risk) on the training set:
  $E(W) = \sum_i (F(W, I_i) - O_i)^2 = \sum_i \left(X_{i,N} - O_i\right)^2$
- Minimization of $E(W)$ by gradient descent

Error back-propagation

- Minimization of $E_S(W)$ by gradient descent:
  - The gradient indicate an ascending direction: move in the opposite
  - Randomly initialize $W(0)$
  - Iterate $W(t+1) = W(t) - \eta \frac{\partial E}{\partial W}\big(W(t)\big)$ $\quad \eta = f(t)$ or $\left(\frac{\partial^2 E}{\partial W^2}\big(W(t)\big)\right)^{-1}$
  - $\frac{\partial E}{\partial W} = \left(\frac{\partial E}{\partial W_1}, \frac{\partial E}{\partial W_2} \ \dots \ \frac{\partial E}{\partial W_N}\right)$
  - Back-propagation: $\frac{\partial E}{\partial W_n}$ is computed by backward recurrence from $\frac{\partial F_n}{\partial W_n}$ and $\frac{\partial F_n}{\partial X_{n-1}}$ applying iteratively $(g \ o \ f)' = (g' o \ f).f'$
  - Two derivatives, relative to weight and to data to be considered

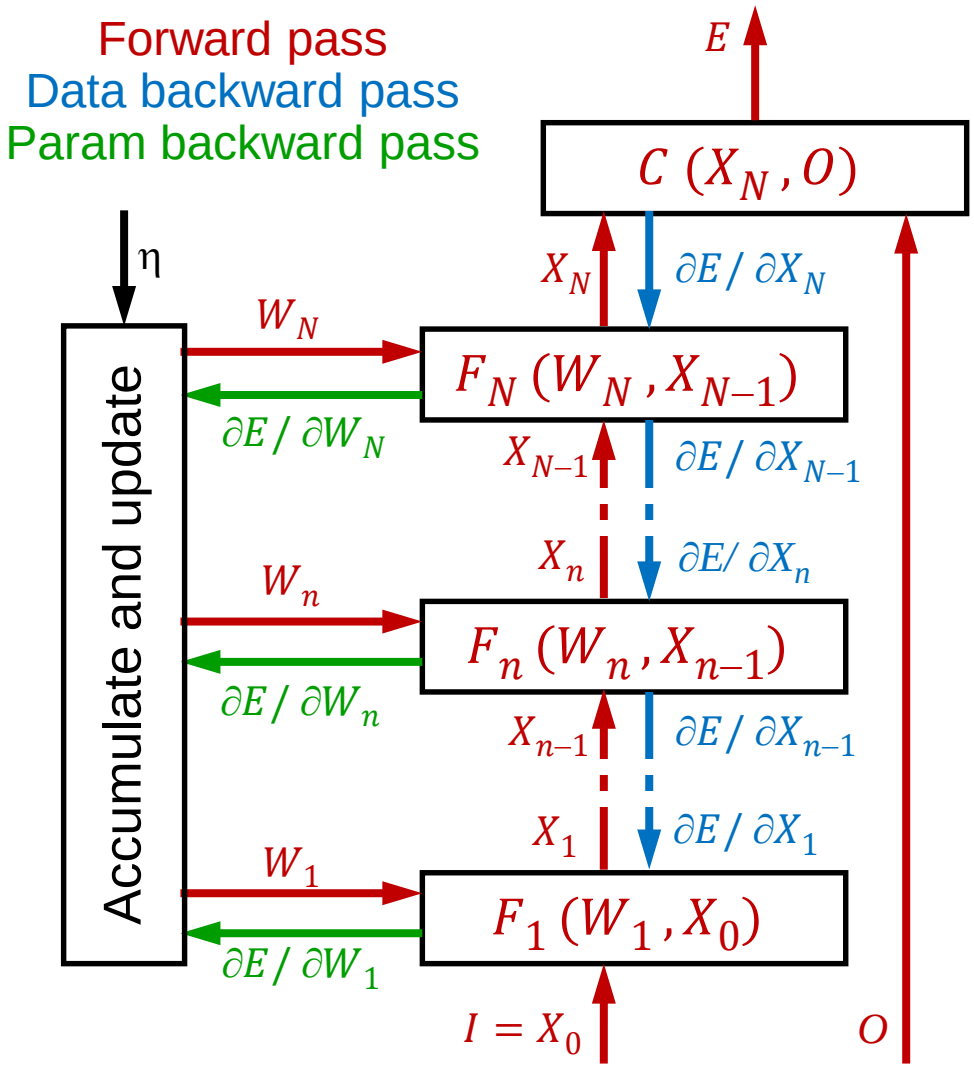
Learning rate evolution

- $W(t+1) = W(t) - \eta(t)\frac{\partial E}{\partial W}\big(W(t)\big)$
- Large learning rate: instability
- Small learning rate: very slow convergence
- Variable learning rate: learning rate decay policy
- Most often: step strategy: iterate “constant during a number of epochs, then divide by a given factor”
- Possibly different learning rates for different layers or for different types of parameters, generally with common evolution

Stochastic gradient descent and batch processing

- $E(W) = \sum_i (F(W, I_i) - O_i)^2 = \sum_i E_i(W)$
- $W(t+1) = W(t) - \eta(t) \frac{\partial E}{\partial W}(t) = W(t) - \sum_i \eta(t) \frac{\partial E_i}{\partial W}(t)$
- Global update (epoch): sum of per sample updates
- Classical GD: update W globally after all I samples have been processed ($1 \leq i \leq I$)
- Stochastic GD: update W after each processed sample → immediate effect, faster convergence
- Batch: update W after a given number (typically between 32 and 256) of processed samples → parallelism

Error back-propagation (adapted from Yann LeCun)



Forward pass, for $1 \leq n \leq N$:

$X_n = F_n(W_n, X_{n-1})$

$E = C\ (X_N, O)$

We need gradients with respect to X_n. For N:

$$\frac{\partial E}{\partial X_N} = \frac{\partial C(X_N, O)}{\partial X_N}$$

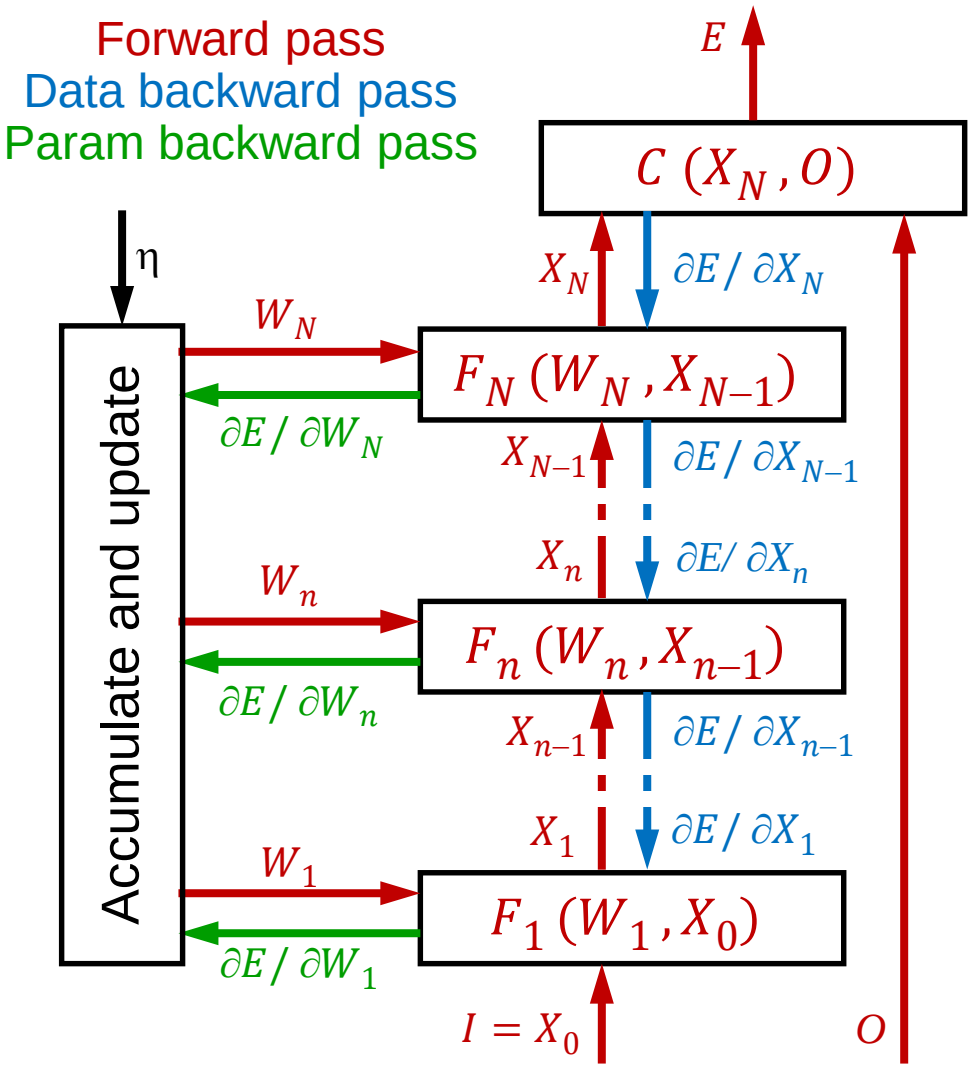
Then backward recurrence:

$$\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(W_n, X_{n-1})}{\partial X_{n-1}}$$

Gradients with respect to W_n. For $1 \leq n \leq N$:

$$\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(W_n, X_{n-1})}{\partial W_n}$$

Error back-propagation (adapted from Yann LeCun)



Forward pass, for $1 \leq n \leq N$:

$X_n = F_n\,(W_n, X_{n-1})$

$E = C\ (X_N, O)$

We need gradients with respect to X_n. For N:

$$\frac{\partial E}{\partial X_N} = \frac{\partial C}{\partial X_N}$$

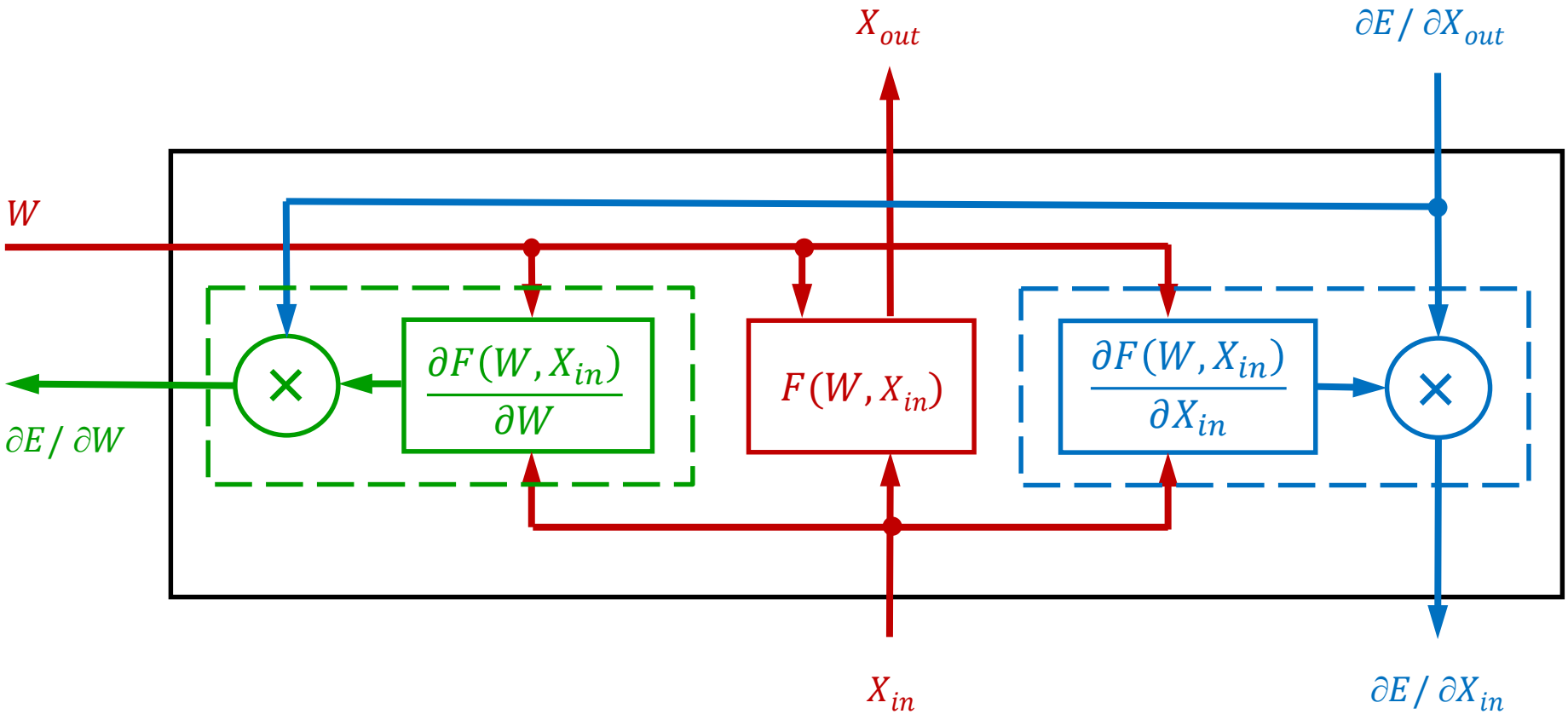
Then backward recurrence:

$$\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n}\frac{\partial X_n}{\partial X_{n-1}}$$

Gradients with respect to W_n. For $1 \leq n \leq N$:

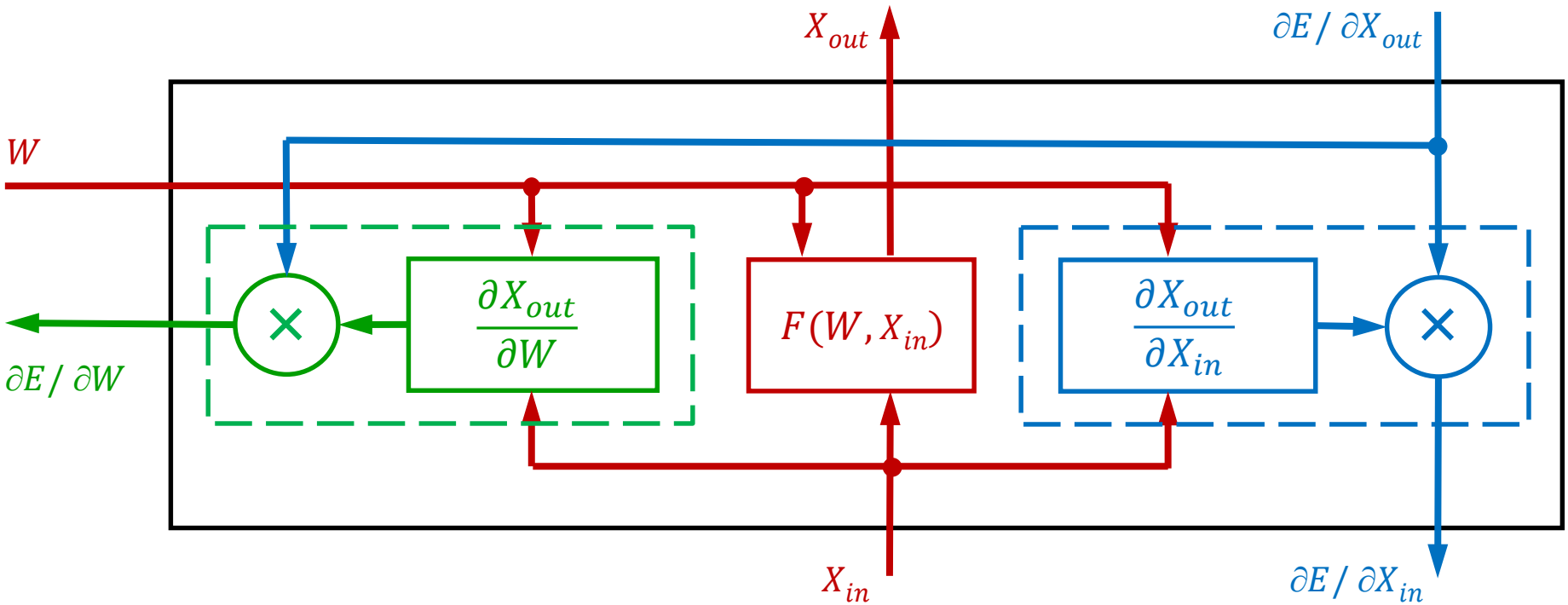
$$\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n}\frac{\partial X_n}{\partial W_n}$$

Layer module (adapted from Yann LeCun)



Notes: $X_{in} \equiv X_{n-1}$, $X_{out} \equiv X_n$, $W \equiv W_n$ and $F \equiv F_n$

Layer module (adapted from Yann LeCun)



$$\frac{\partial F(W, X_{in})}{\partial X_{in}} \equiv \frac{\partial X_{out}}{\partial X_{in}} \qquad \frac{\partial E}{\partial X_{in}} = \frac{\partial E}{\partial X_{out}} \frac{\partial X_{out}}{\partial X_{in}}$$

$$\frac{\partial F(W, X_{in})}{\partial W} \equiv \frac{\partial X_{out}}{\partial W} \qquad \frac{\partial E}{\partial W} = \frac{\partial E}{\partial X_{out}} \frac{\partial X_{out}}{\partial W}$$

Layer module (adapted from Yann LeCun)

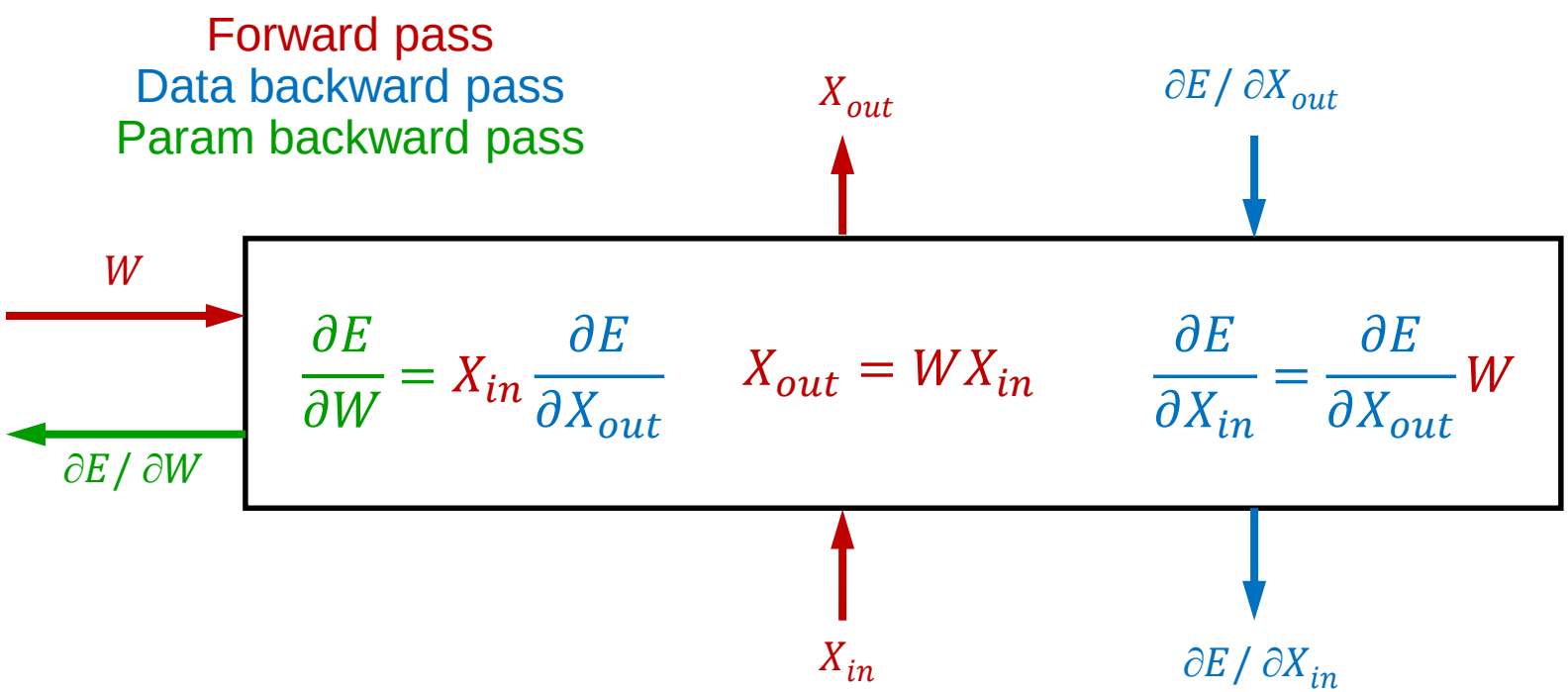
Gradient back-propagation rule:

The gradient relative to the input (either W or X_{in}) is equal to the gradient relative to the output (X_{out}) times the Jacobian of the transfer function (respectively $\frac{\partial X_{out}}{\partial W}$ or $\frac{\partial X_{out}}{\partial X_{in}}$, left multiplication)

$$\frac{\partial F(W, X_{in})}{\partial X_{in}} \equiv \frac{\partial X_{out}}{\partial X_{in}} \qquad \frac{\partial E}{\partial X_{in}} = \frac{\partial E}{\partial X_{out}} \frac{\partial X_{out}}{\partial X_{in}}$$

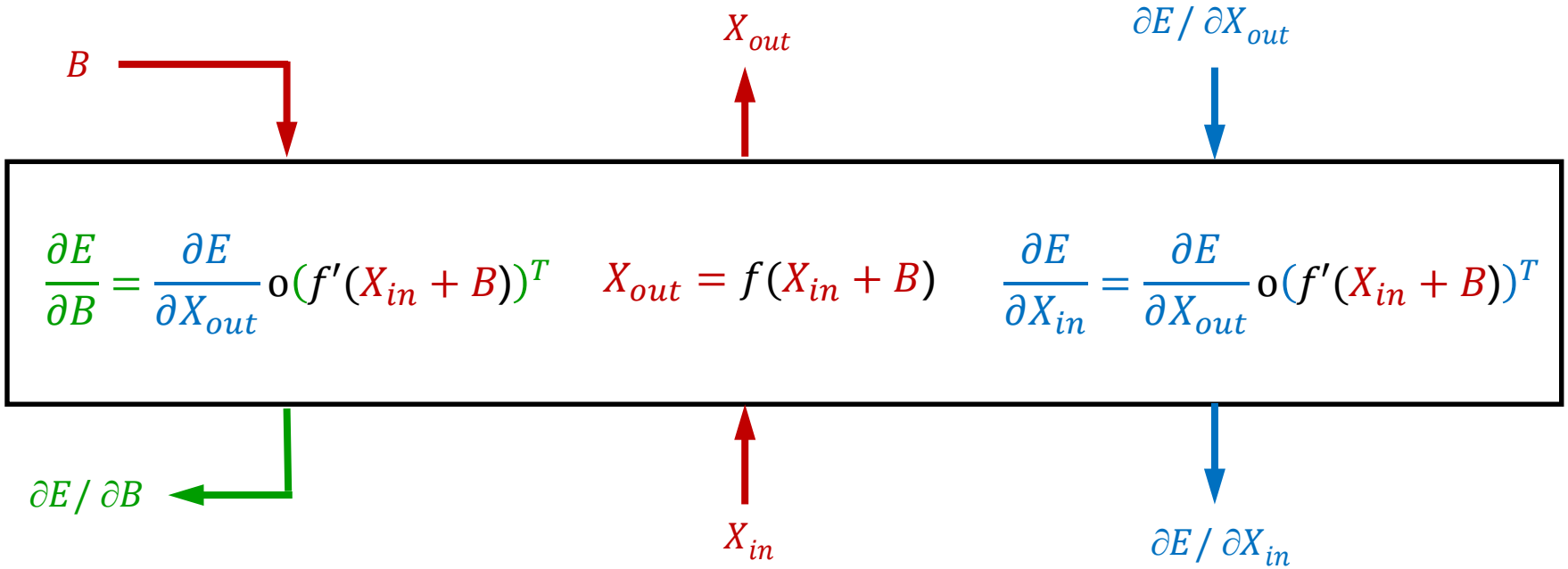
$$\frac{\partial F(W, X_{in})}{\partial W} \equiv \frac{\partial X_{out}}{\partial W} \qquad \frac{\partial E}{\partial W} = \frac{\partial E}{\partial X_{out}} \frac{\partial X_{out}}{\partial W}$$

Linear module (adapted from Yann LeCun)



Note: X_{in} and X_{out} are regular (column) vectors and W is a matrix while $\partial E / \partial X_{in}$ and $\partial E / \partial X_{out}$ are transpose (row) vectors, this is because $\mathrm{d}E = (\partial E / \partial X).\mathrm{d}X$. $\partial E / \partial W$ is a transpose matrix which is the outer product of the regular and transpose vectors X_{in} and $\partial E / \partial X_{out}$.

Pointwise module (adapted from Yann LeCun)



Notes: B is a bias vector on the input. X_{in}, X_{out} and B are regular (column) vectors all of the same size while $\partial E / \partial X_{in}$ and $\partial E / \partial X_{out}$ and $\partial E / \partial B$ are transpose vectors also of the same size. f is a scalar function applied pointwise on $X_{in} + B$. f' is the derivative of f and is also applied pointwise. The multiplication by $(f'(X_{in} + B))^T$ is also performed pointwise (Hadamard product denoted “o” here).

Non-linear functions

- Sigmoid: $z = \frac{1}{1+e^{y}}$
- Hyperbolic tangent: $z = \tanh y$
- Rectified Linear Unit (ReLU): $z = \max(0, y)$
- Programmable ReLU (PReLU) : $z = \max(\alpha y, y)$ with α learned (i.e. $\alpha \subset W$)
- …
- Appropriate non-linear functions leads to better performance and/or faster convergence
- Avoid vanishing / exploding gradients

Convolutional layers

- Alternative to the “all to all” connections
- Preserves the image topology via “feature maps”
- Each layer is a “stack” of features maps
- Each map points is connected to the map points of a neighborhood in the previous layer
- Weights between maps are shared so that they are invariant by translation
- Resolution changes across layers: stride and pooling
- Example: AlexNet

Classical image convolution (2D to 2D)

- Classical image convolution (2D to 2D):

$$O(i,j) = (K * I)(i,j) = \sum_{(m,n)} K(m,n) I(i-m, j-n)$$

- Convolutional layer (3D to 3D):
- m and n : within a window around the current location, corresponding to the filter size
- $K(m,n)$: noyau de convolution
- Example: (circular) Gabor filter:

$$K(m,n) = \frac{1}{2\pi\sigma^2} . e^{-\frac{m^2+n^2}{2\sigma^2}} . e^{2\pi i \frac{m.\cos\theta + n.\sin\theta}{\lambda}}$$

Set of image convolutions (2D to 3D)

- Set of image convolution (2D to 3D):

$$O(l,i,j) = (K(l) * I)(i,j) = \sum_{(m,n)} K(l,m,n) I(i-m, j-n)$$

- l : index of the convolution map
- Example: Set of (circular) Gabor filters:

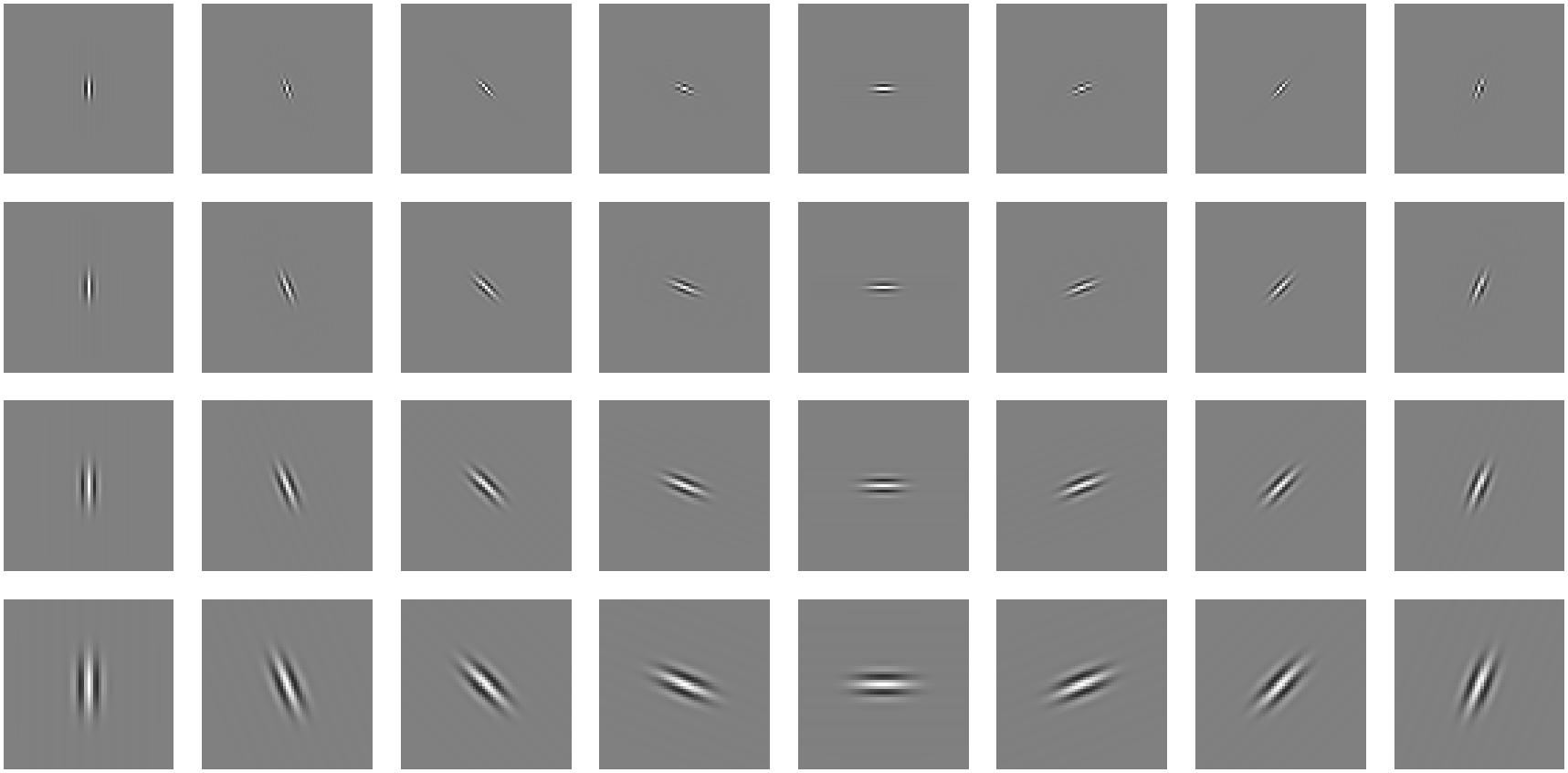
$$K(l,m,n) = \frac{1}{2\pi\sigma_l^2} . e^{-\frac{m^2+n^2}{2\sigma_l^2}} . e^{2\pi i \frac{m.\cos\theta_l + n.\sin\theta_l}{\lambda_l}}$$

$(\sigma_l, \lambda_l, \theta_l)_{(1 \le l \le L)}$: set of (circular) Gabor filter parameters

practical filter size: $\pm 4\sigma$

Example Gabor Filter Kernels

Example of (elliptic) filters with 8 orientations and 4 scales



Convolutional layers

- Set of image convolution (2D to 3D):

$$O(l,i,j) = (K(l) * I)(i,j) = \sum_{(m,n)} K(l,m,n) I(i-m, j-n)$$

- Convolutional layer: multiple maps (planes) both in input and output (3D to 3D, plus bias):

$$O(l,i,j) = B(l) + \sum_{(k,m,n)} K(k,l,m,n) I(k, i-m, j-n)$$

- k and l: indices of the feature maps in the input and output layers

- m and n: within a window around the current location, corresponding to the feature size

Convolutional layers

- Convolutional layer: multiple maps (planes) both in input and output (3D to 3D, plus bias):

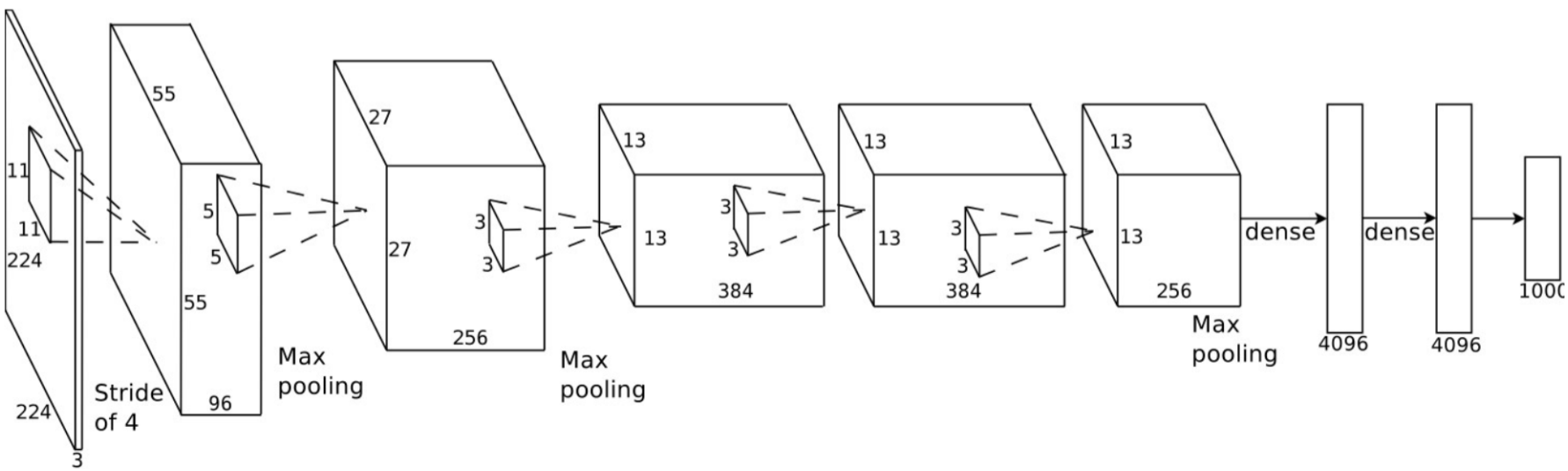
$$O(l,i,j) = B(l) + \sum_{(k,m,n)} K(k,l,m,n) I(k,i-m,j-n)$$

- Operation relative to (m,n) : convolution
- Operation relative to (k,l) : matrix multiplication plus bias (equals affine transform)
- Combination of:
  - Convolution within the image plane, image topology
  - Classical all to all “perpendicularly” to the image plane, no topology
- If image size and filter size = 1: fully connected “all to all”

ImageNet Challenge 2012

[Krizhevsky et al., 2012]

- 7 hidden layers, 650K units, 60M parameters (W)
- GPU implementation (50× speed-up over CPU)
- Trained on two GPUs for a week

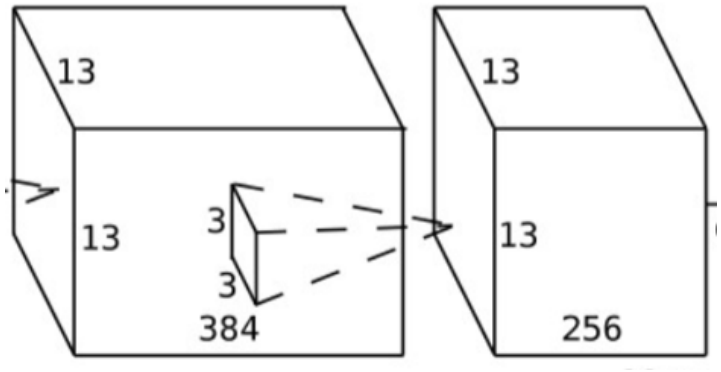


A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Convolutional layers

- The convolution layer kernel is: $(D+2)$-dimensional for D-dimensional input data, e.g. $D=2$ for still images, $D=3$ for videos or scanner images.
- For color images, the RGB (or YUV or HSV …) planes directly enter the first layer as a 2D volume of size 3 × width × height
- There is one unit (neuron) per “pixel” in the output D-dimensional topology
- All these units have the same $(D+2)$-dimensional kernel, i.e. kernels are invariant by translation in the D-dimensional topology (convolution)

AlexNet “conv5” example



- Number of units (output “image” size):
  output image width (13) × output image height (13) = 169

- Number of weights in a unit (= number of weights in a layer):
  number of input planes (384) × number of output planes (256) × filter width (3) × filter height (3) = 884736 (bias not included)

- Number of connections:
  number of units × number of weights in a unit = 149520384

Resolution changes and side effects

- Side (border) effect:
  - crop the output “image” relative to the input one and/or
  - pad the image if the filter expand outside
- Resolution change (generally reduction):
  - Stride: subsample, e.g. compute only one out of N, and/or
  - Pool: compute all and apply an associative operator to compute a single value for the low resolution location from the high resolution ones
- Common pooling operators: maximum or average
- Pooling correspond to a separate back-propagation module (as the linear and non-linear parts of a layer)

Dropout

- Regularization technique
- During training, at each epoch, neutralize a given (typically 0.2 to 0.5) proportion of randomly selected connections
- During prediction, keep all of them with a multiplicative compensating factor
- Avoid concentration of the activation on particular connections
- Much more robust operation
- Faster training, better performance

Softmax

- Normalization of output as probabilities (positive values summing to 1) for the multi-class problem (i.e. target categories are mutually exclusive)
- $z_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$
- Not suited for the multi-label case (i.e. target categories are not mutually exclusive)
- Associated loss function is cross-entropy

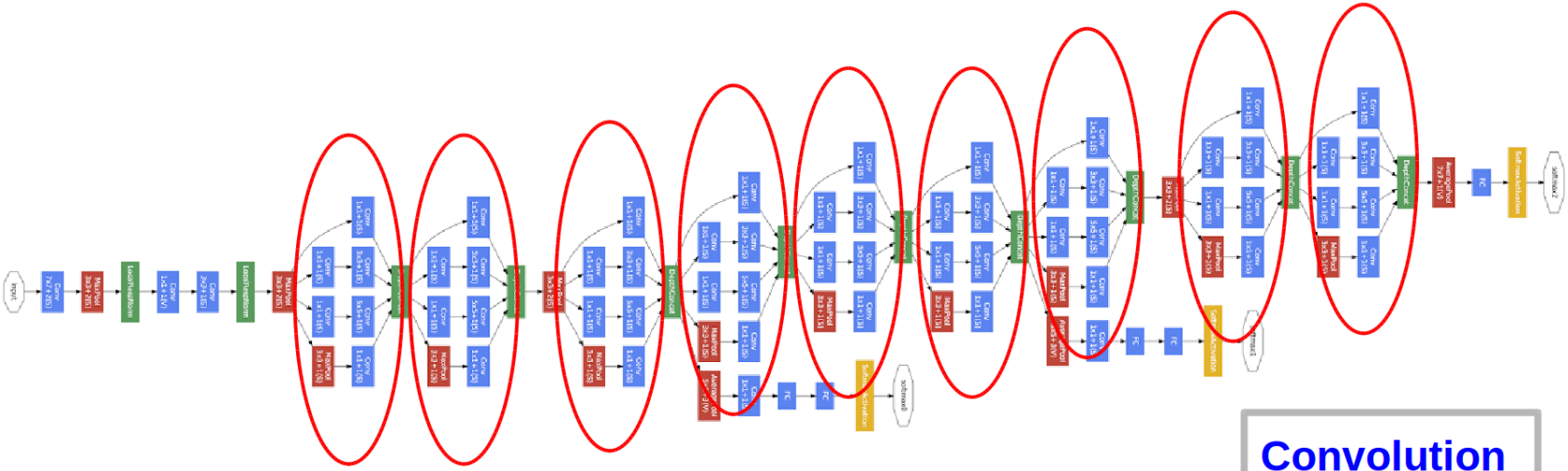
Yann LeCun recommendations

- Use ReLU non-linearities (tanh and logistic are falling out of favor)
- Use cross-entropy loss for classification
- Use Stochastic Gradient Descent on minibatches
- Shuffle the training samples
- Normalize the input variables (zero mean, unit variance)
- Schedule to decrease the learning rate
- Use a bit of L1 or L2 regularization on the weights (or a combination)
  - But it's best to turn it on after a couple of epochs
- Use “dropout” for regularization
  - Hinton et al 2012 http://arxiv.org/abs/1207.0580
- Lots more in [LeCun et al. “Efficient Backprop” 1998]
- Lots, lots more in “Neural Networks, Tricks of the Trade” (2012 edition)edited by G. Montavon, G. B. Orr, and K-R Müller (Springer)

Recent trends

- VGG and GoogLeNet (16-19 and 22 layers)
- Residual networks (152 layers with “shortcuts”)
- Stochastic depth networks (up to 1202 layers)
- Dense Networks
- Weakly supervised / unsupervised learning
- Generative adversarial networks

GoogLeNet (very deep)

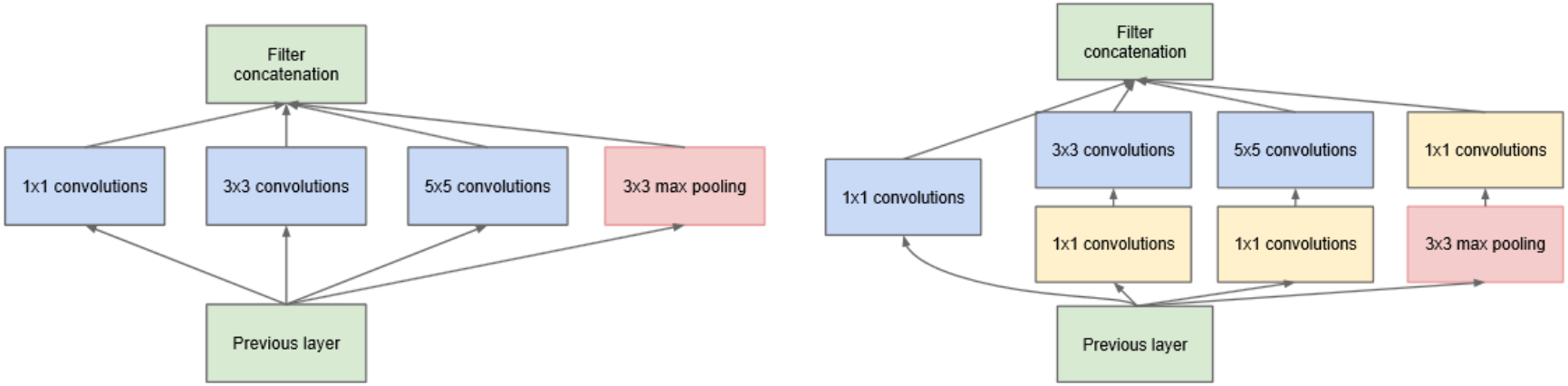


9 “inception” modules

Convolution
Pooling
Softmax
Concat/Normalize

Christian Szegedy et al.: *Going Deeper with Convolutions*, CVPR 2014.

GoogLeNet (very deep)



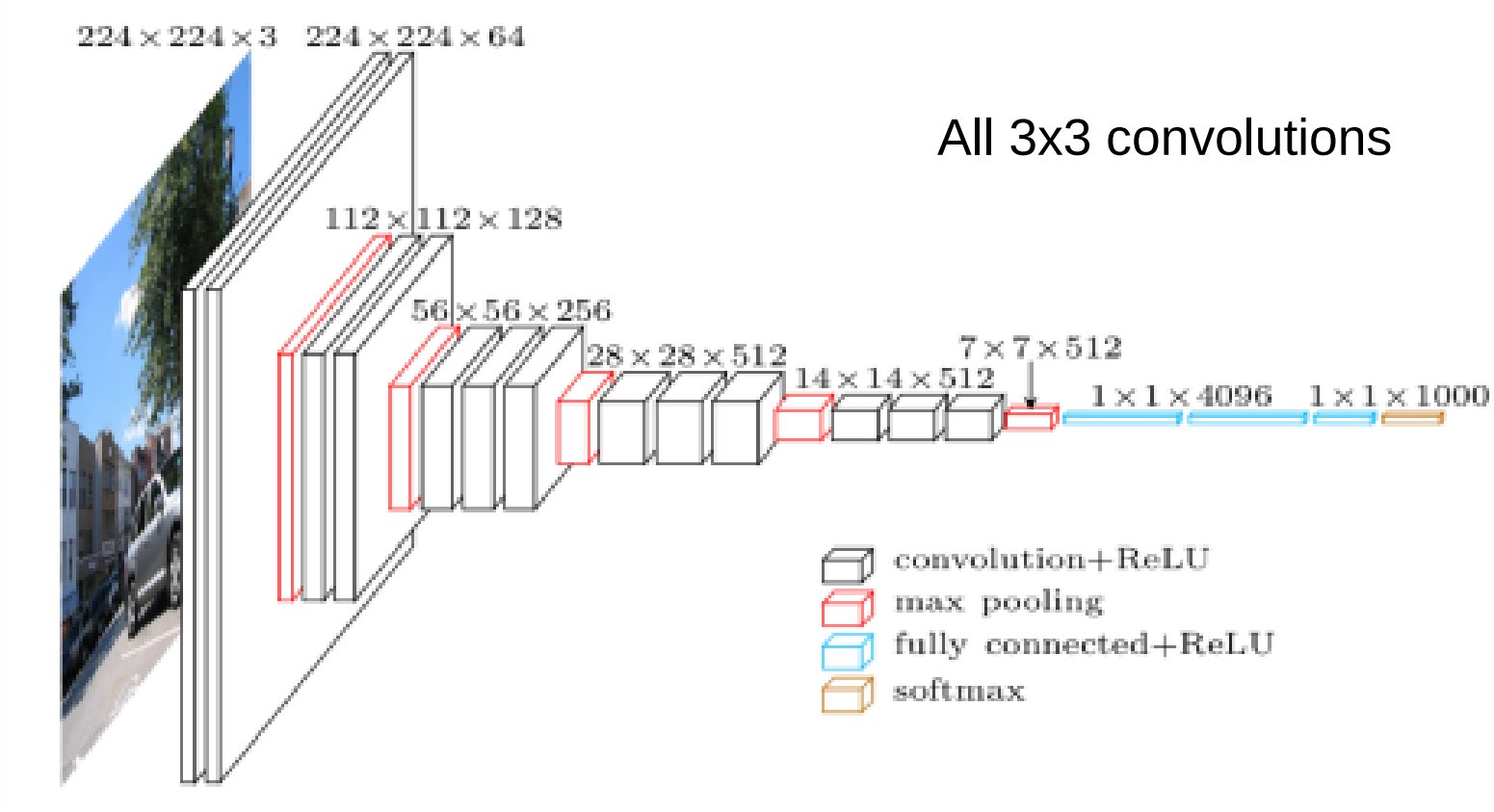
(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

Reminder: 1x1 convolutions actually implements an all-to-all between the input and output maps (pixel-wise all-to-all)

Christian Szegedy et al.: *Going Deeper with Convolutions*, CVPR 2014.

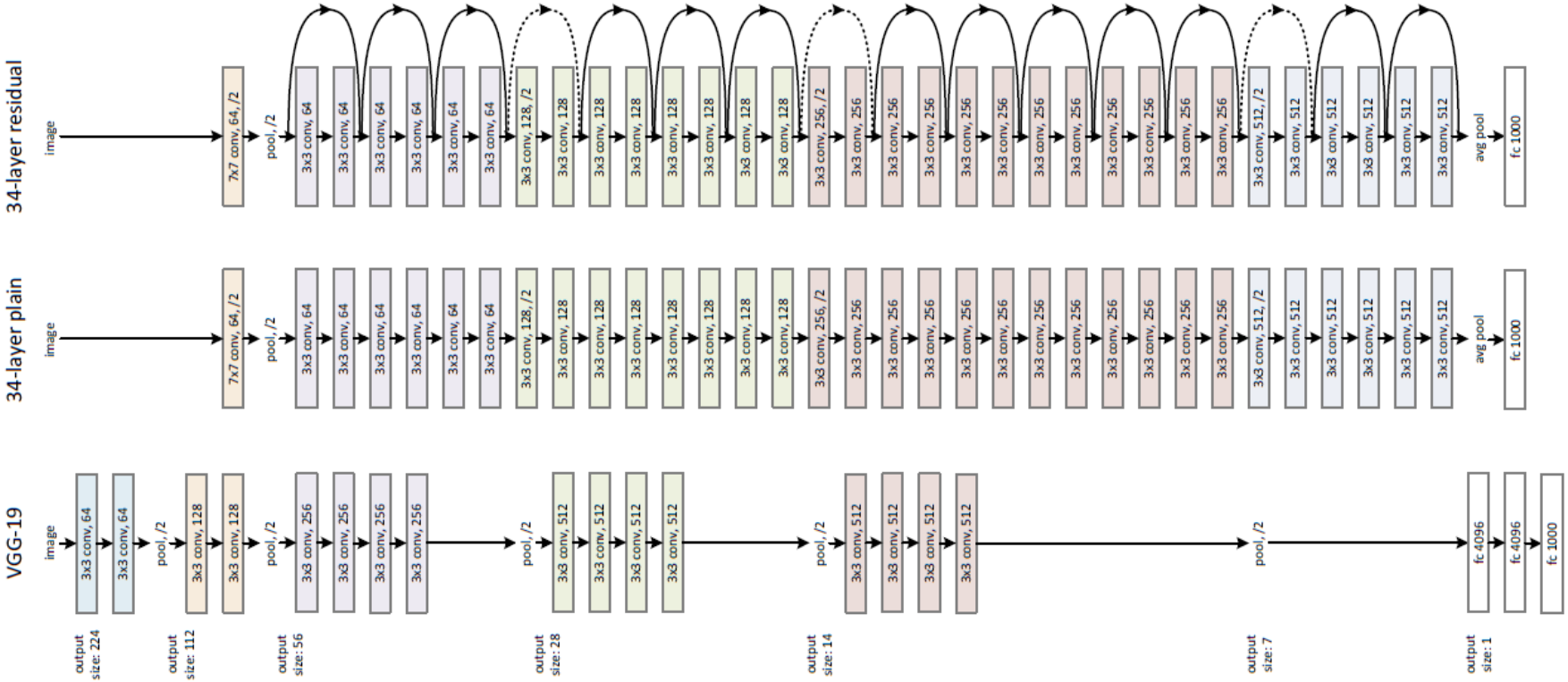
VGG Network (very deep)



All 3x3 convolutions

Simonyan and Zisserman, Andrew: *Very Deep Convolutional Networks for Large-Scale Image Recognition*, CVPR 2014.

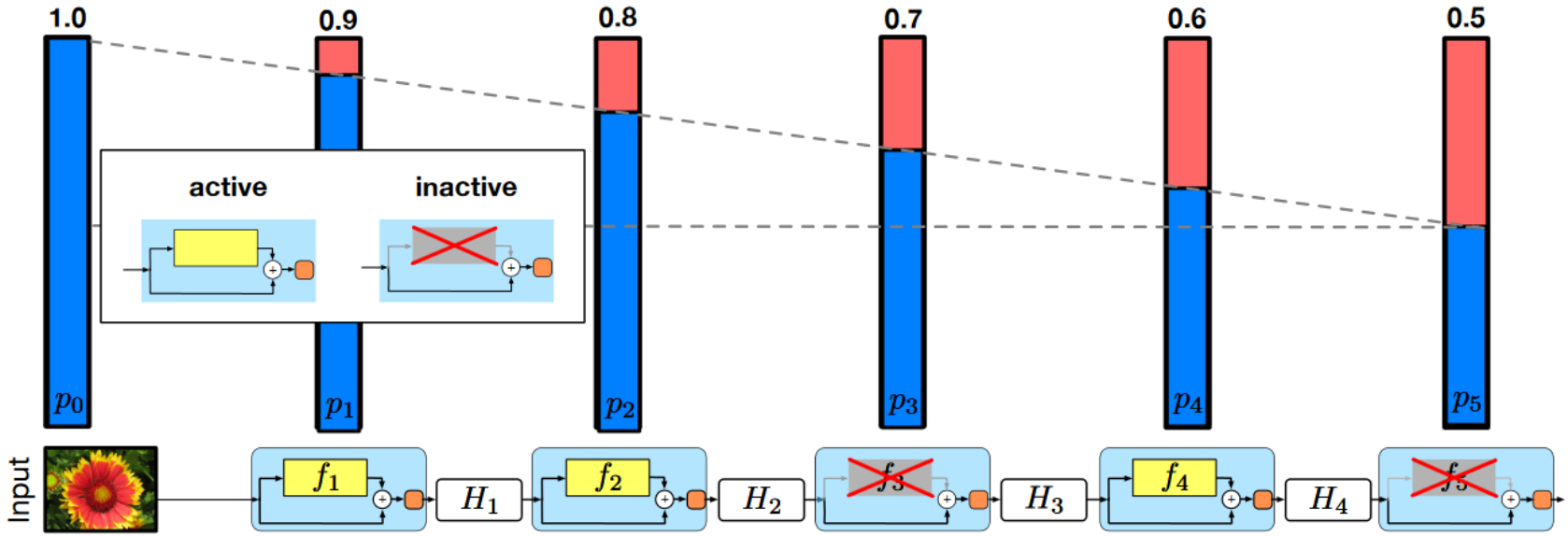
Residual networks (ultra deep)



Ultra deep network with “shortcuts”

He, Zhang, Ren and Sun: Deep Residual Learning for Image Recognition, CVPR 2015

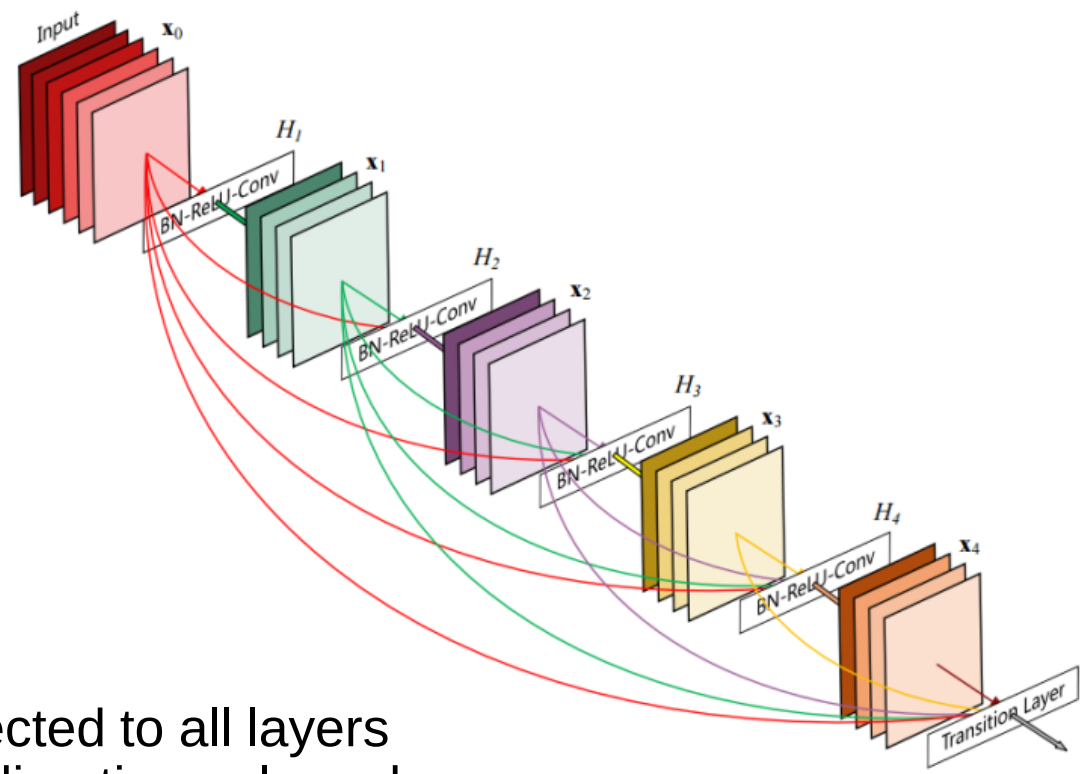
Stochastic depth networks (extremely deep)



ResNet with stochastic depth
“Dropout at the layer level”

Huang et al.: *Deep Networks with Stochastic Depth*, CVPR 2016

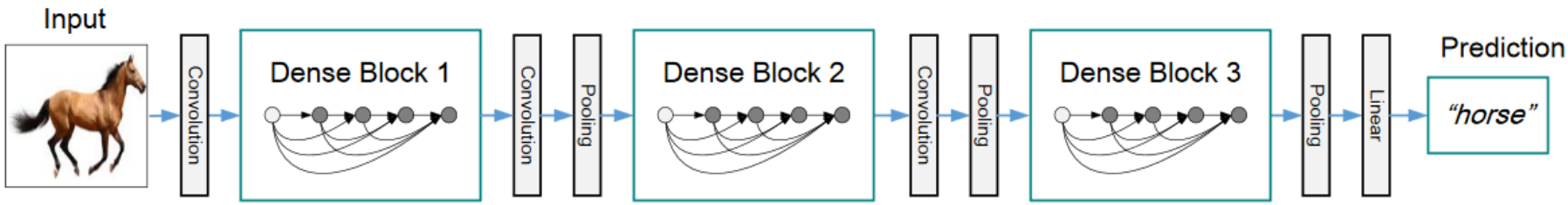
Dense networks



All layers connected to all layers (in the forward direction only and without resolution change

Huang et al.: *Densely Connected Convolutional Networks*, CVPR 2016

Dense networks



A deep DenseNet with three dense blocks
The layers between blocks are transition layers that change the resolution via convolution and pooling

Huang et al.: *Densely Connected Convolutional Networks*, CVPR 2016

Weakly / unsupervised learning

- Webly Supervised Learning of Convolutional Networks
  Xinlei Chen and Abhinav Gupta
  arXiv:1505.01554, May 2015
- Effective training of convolutional networks using noisy Web images
  Phong D. Vo, Alexandru Ginsca, Hervé Le Borgne, Adrian Popescu
  CBMI, June 2015
- Learning from Massive Noisy Labeled Data for Image Classification
  Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang
  CVPR, June 2015
- Harnessing Noisy Web Images for Deep Representation
  Phong D. Vo, Alexandru Ginsca, Hervé Le Borgne, Adrian Popescu
  arXiv:1512.04785, July 2016
- Learning Visual Features from Large Weakly Supervised Data
  Armand Joulin, Laurens van der Maaten, Allan Jabri, and Nicolas Vasilache
  ECCV, Sep. 2016

Weakly / unsupervised learning

- Gather millions (from 1 to 100) of images from the web
- Two main strategies:
  - Query an image search engine (e.g. Google) with either target tags or descriptions → we can choose the categories
  - Download images with associated descriptions from a social network (e.g. Flickr) and extract/select tags from the description → we have to do with the available categories
- Filter the results (may use cross-validation predictions)
- Train from noisy data and compensate the loss due to noise with a gain from quantity
- Work on the quality of the category-image association
- Use classifiers or features for transfer learning

Engineered versus learned descriptors

- Classical “classification pipeline”
  - Extraction(s) – [aggregation] – optimization(s) – classifier(s) – one or more levels of fusion – re-scoring (non exhaustive example)
  - Most of the stages are explicitly engineered: the form of descriptors or processing steps has been thought and designed by a skilled engineer or researcher
  - *Lots* of experience and acquired expertise by thousands of smart people over tens of years
  - Learning concerns only the classifier(s) stages and a few hyper-parameters controlling the other ones
  - Almost everything has been tried
  - The more you incorporate, the more you get (at a cost)

Engineered versus learned descriptors

- Deep learning pipeline: MLP with about 8 layers
  - Advances in computing power (Tflops): large networks possible
  - Algorithmic advance: combination of convolutional layers for the lower stages with all-to-all layers; the topology of the image is preserved in the lower layers with weights shared between the units within a layer
  - Algorithmic advances: NN researchers finally find out how to have back-propagation working for MLP with more than three layers
  - Image pixels are entered *directly* into the first layer
  - The first (resp. intermediate, last) layers practically compute low-level (resp. intermediate level, semantic) descriptors
  - Everything is made using a unique and homogeneous architecture
  - A single network can be used for detecting many target concepts
  - All the level are jointly optimized at once
  - Requires *huge* amounts of training data

Deep Learning and IAR

- Indexing for key-word-based search
  - Get an estimate of presence probability for an as large as possible set of concepts / categories
  - Map any query to a subset of them
  - Score the multimedia samples according to the presence probabilities of the selected ones
- Query by example
  - Use last layers values (output or last but one or last but two) as semantic feature vectors (descriptors) for the query and the candidate
  - Classical QBE with Euclidean distance or scalar product