Deep Learning
History and Building blocks

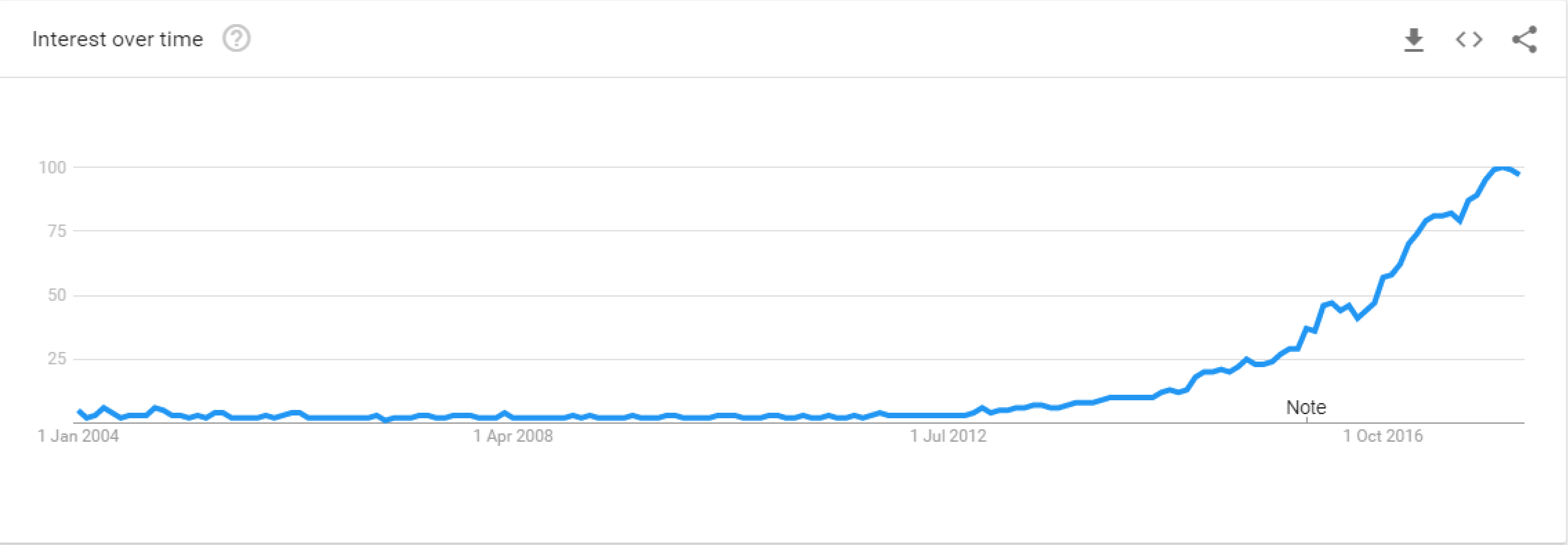
Adrien Foucart

INFO-H-501

Deep Learning : History and Building blocks

- What is "Deep Learning" and where does it come from ?
- Deep Learning for Image Analysis: the basics
- Building blocks
- Examples & practical overview

A recent technology?



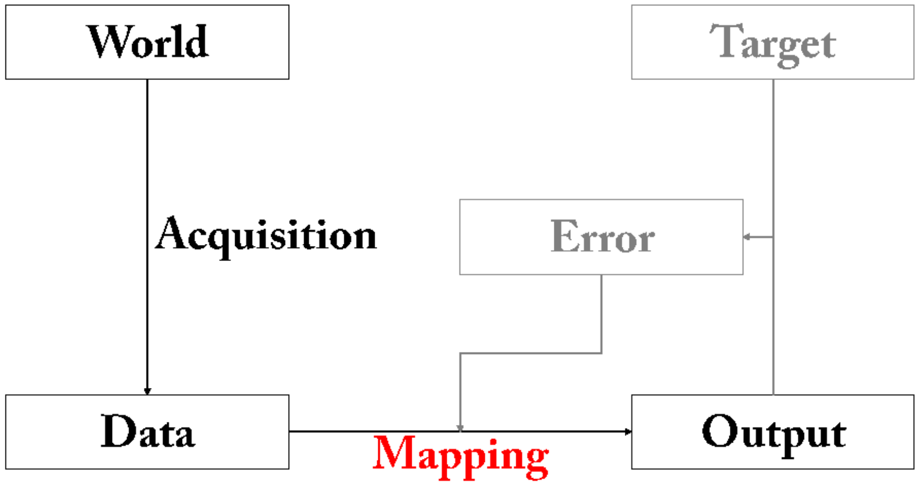
"Deep Learning" interest over time (from Google Trends)

What is Deep Learning?

First: what is "learning"?

- Transfer of knowledge ("supervised learning")
- Experience ("unsupervised learning")

→ using available **data** to do a certain task and/or produce a desired **output**.



Humans are great at learning

Using **vision/touch** to **grab an object** and **do something with it**.

© Jim Makos, Creative Commons
https://www.flickr.com/photos/jim-makos/16339006689

DARPA Robotics Challenge 2015

Computers are great at... computing?

Tasks where computers have been better than humans for a while:

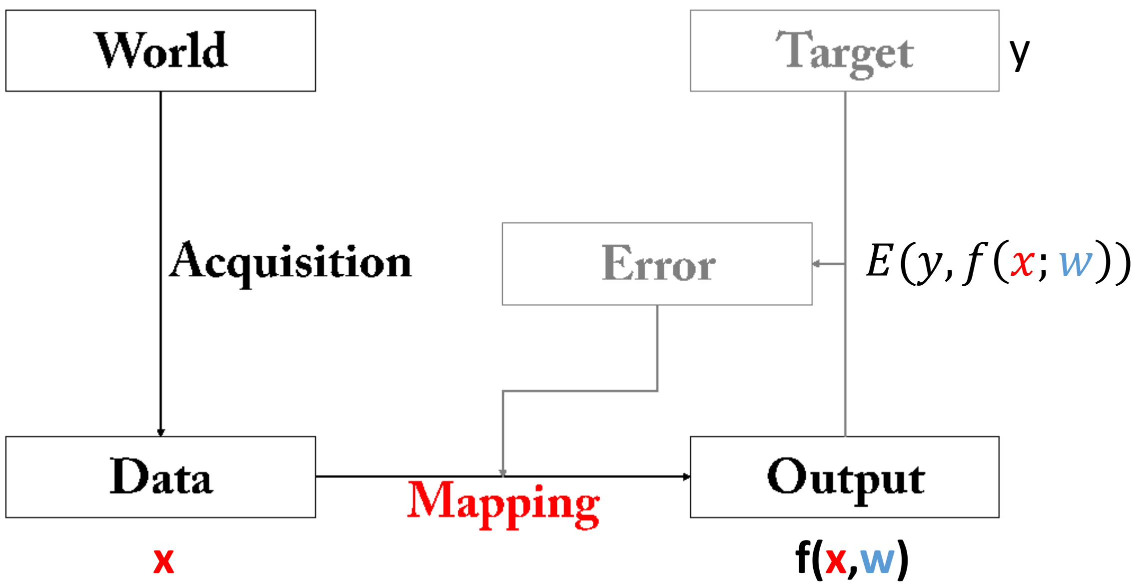
- Inverting a large matrix
- Solving numerical equations
- Chess

→ Problems which are **easy to formalize** but **hard to solve**.

Deep Blue v Kasparov, 1997
Adam Nadel/AP Images

What is Deep Learning?

For a computer, learning is **solving an optimization problem**.



What is Deep Learning?

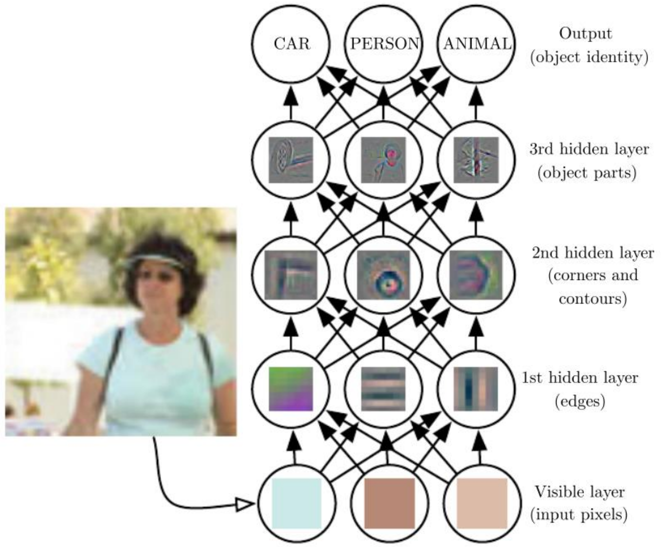
Expert system	Raw Data	Hand-crafted features		Hand-crafted decision process	Output
"Classic" Machine Learning	Raw Data	Hand-crafted features		Machine Learning algorithm with learned parameters	Output
Representation Learning	Raw Data	Learned features (e.g. Autoencoder, PCA)		Machine Learning algorithm with learned parameters	Output
Deep Learning	Raw Data	Learned Low-Level Features	…	Learned High-Level Features	Output

What is Deep Learning?

Learning by **layers of abstraction** (complex features built on simple features).

Learning **directly from the input**.

Learning mechanism **more similar to the human brain**.



M.D. Zeiler and R. Fergus. *Visualizing and understanding convolutional networks*. ECCV'14.

The long history of Deep Learning

The "**artificial human brain**" is an old idea.

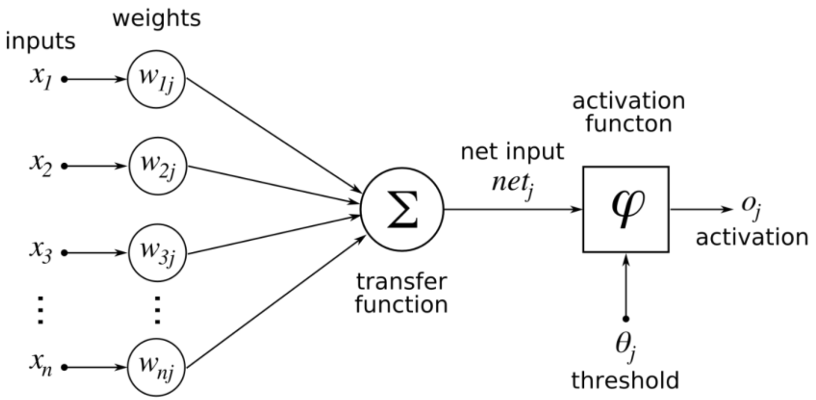
19th century : discovery of the neuron (Schwann, Purkinje, Ramon y Cajal…)

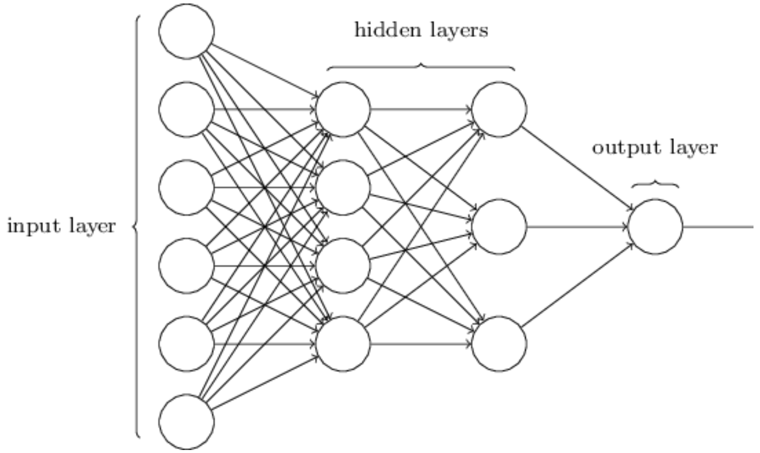
1940s-1950s : Perceptron (McCulloch & Pitts, Rosenblatt…)

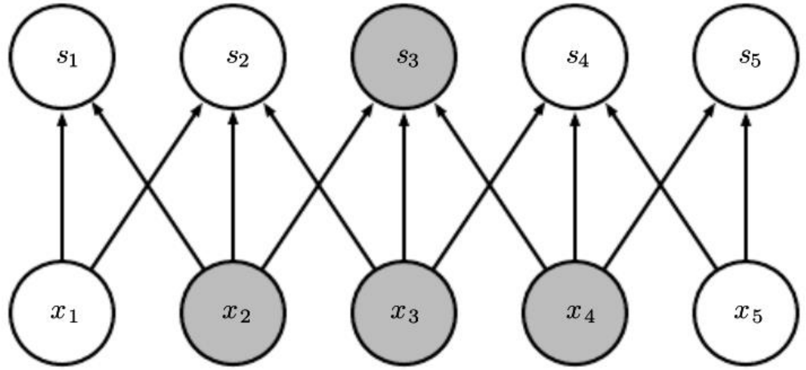
1950s-1960s : Multi-Layer Perceptrons (Ivakhnenko…)

…

1980s-1990s : Convolutional networks (Fukushima, LeCun…), backpropagation (Werbos, LeCun…), LSTMs (Schmidhuber…)

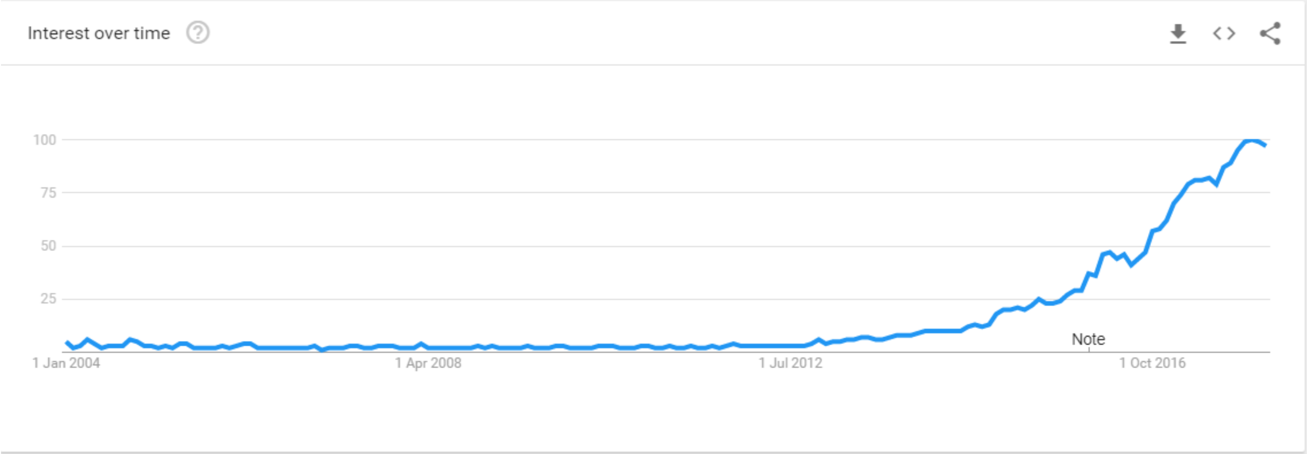






The long history of Deep Learning

Neural networks: an **interesting** idea, but not very **practical**…

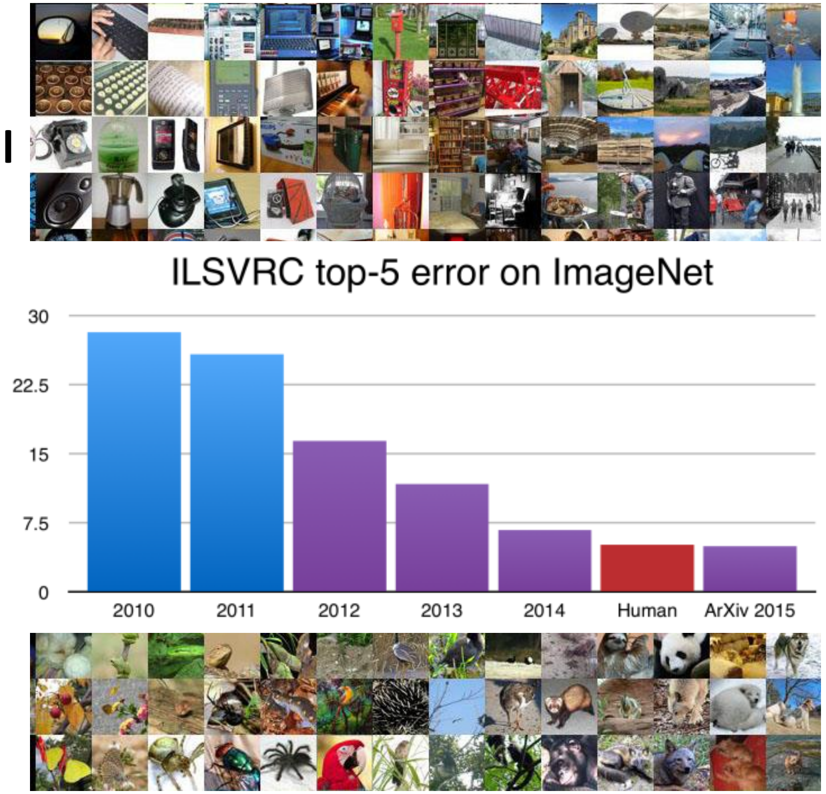


…until recently.

The long history of Deep Learning

The Deep Learning invasion:

ImageNet Large Scale Visual Recognition Challenge



Q. Le et al. *Building high-level features using large-scale unsupervised learning*. ICML, 2012.

The long history of Deep Learning

WIRED STAFF SCIENCE 06.26.12 11:15 AM

GOOGLE'S ARTIFICIAL BRAIN LEARNS TO FIND CAT VIDEOS

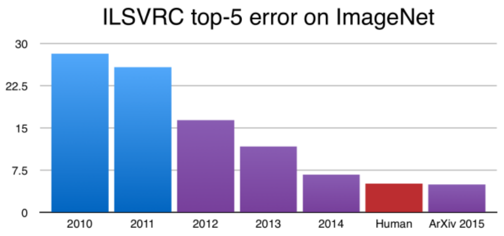
By Liat Clark, Wired UK

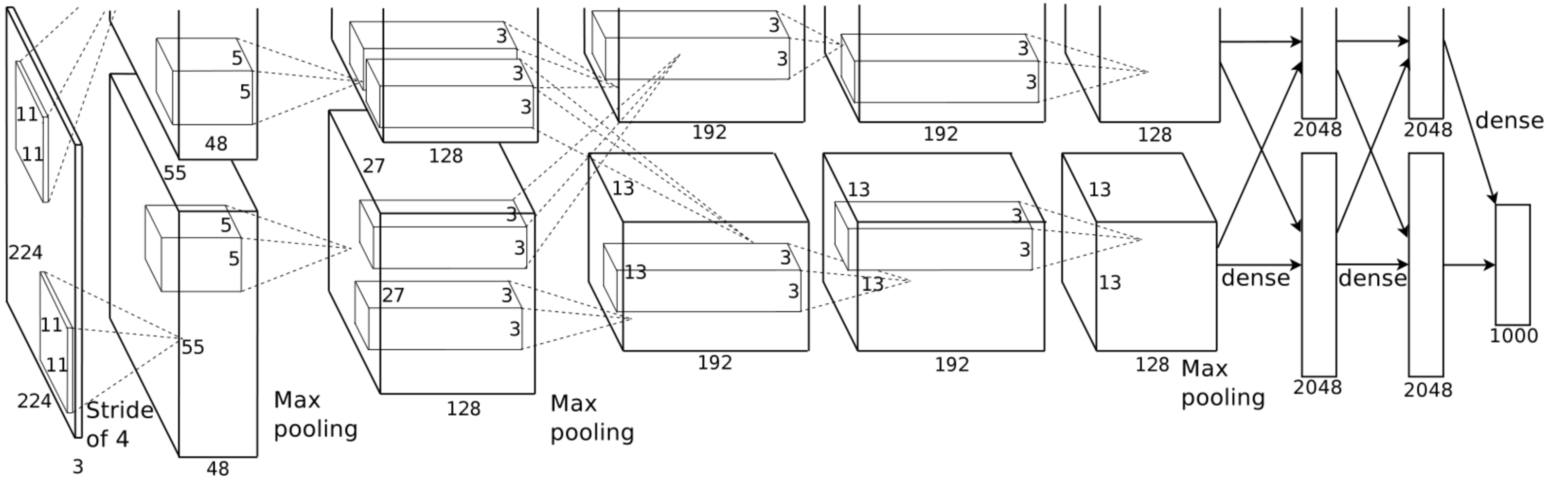
When computer scientists at Google's mysterious X lab built a neural network of 16,000 computer processors with one billion connections and let it browse YouTube, it did what many web users might do – it began to look for cats.

https://www.wired.com/2012/06/google-x-neural-network/

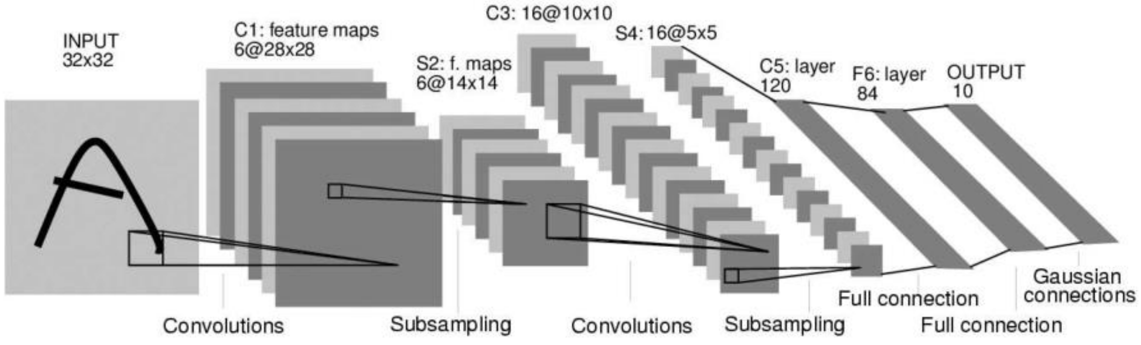
The long history of Deep Learning

AlexNet (2012)





A. Krizhevsky et al., *ImageNet Classification with Deep Convolutional Neural Networks*. In NIPS, 2012



LeNet (1998)

Y. LeCun et al., *Gradient-Based Learning Applied to Document Recognition*, Proc. of the IEEE, November 1998.

The long history of Deep Learning

An "invasion" driven by:

- More computing power
- Bigger datasets

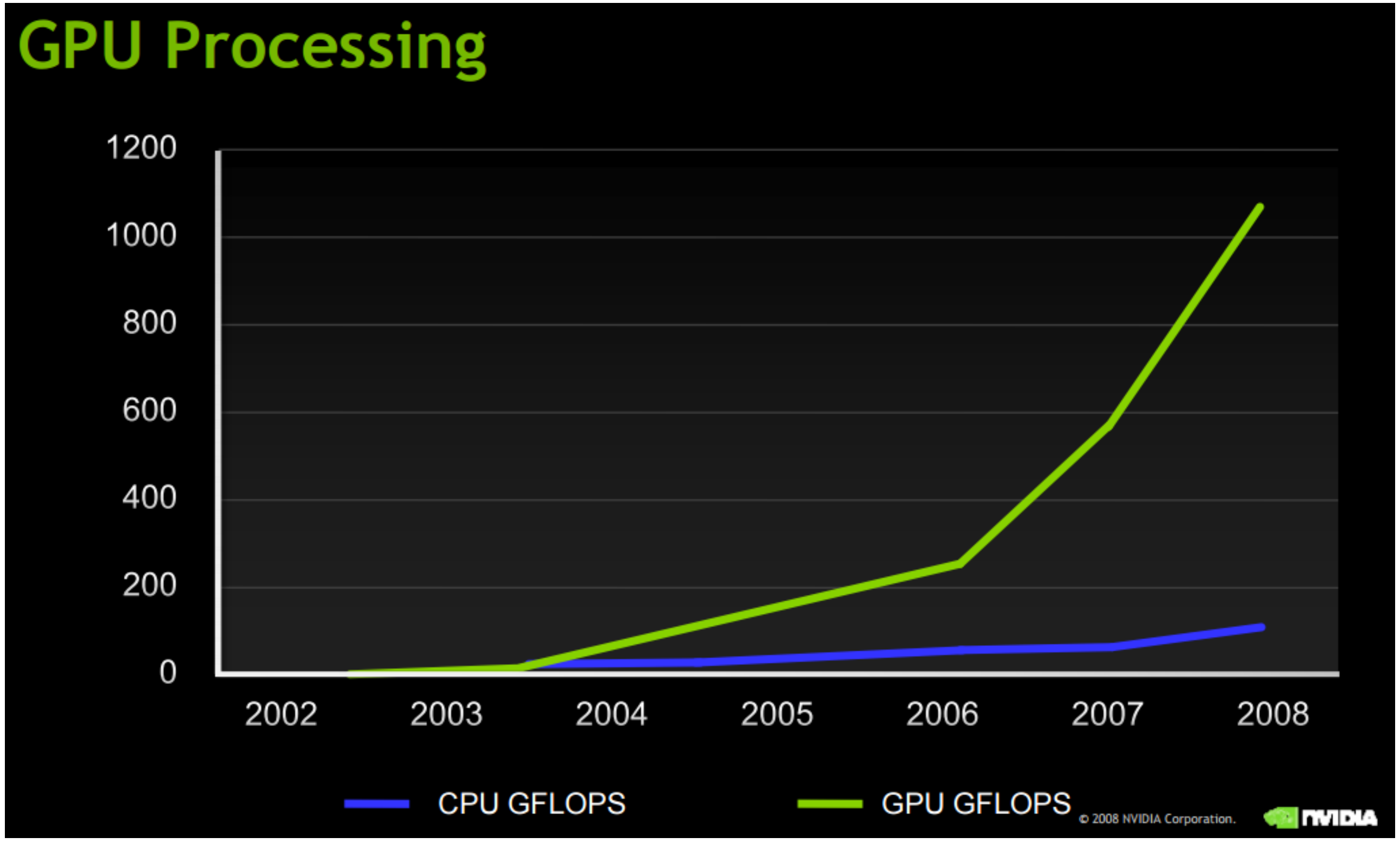
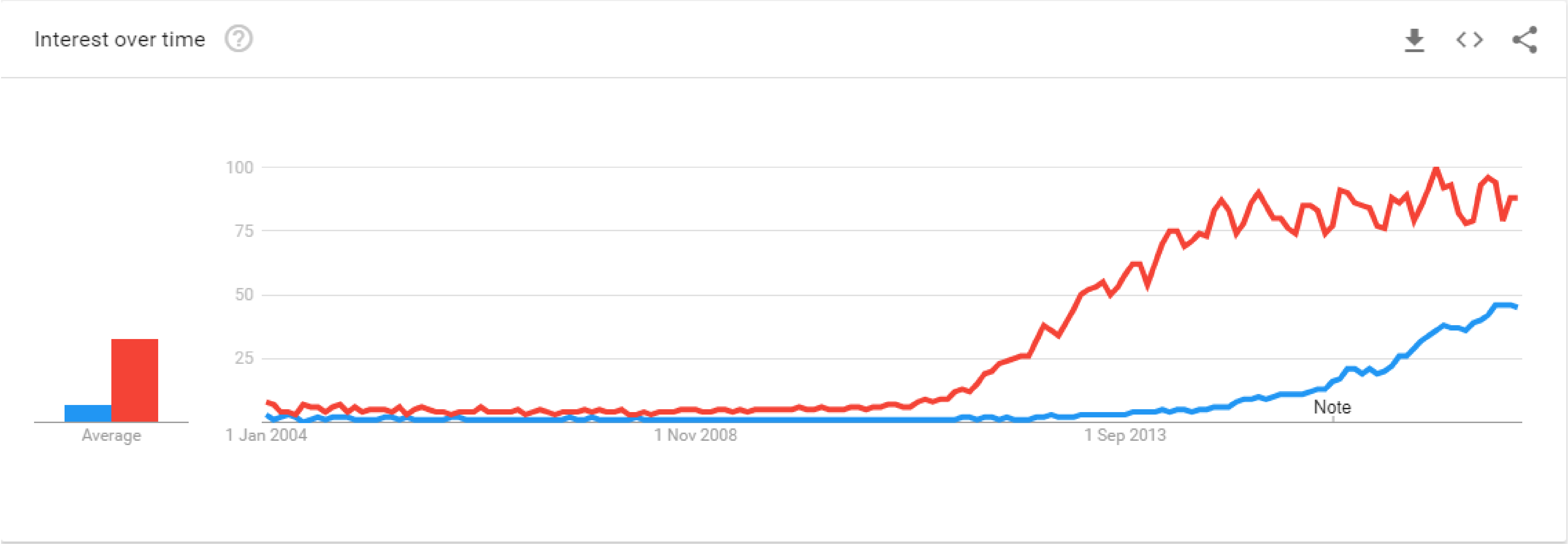


Image: NVIDIDA

Deep Learning needs Big Data

Deep Learning methods require to optimize a function in a **very high-dimensional space**. This in turn requires a **large dataset**.



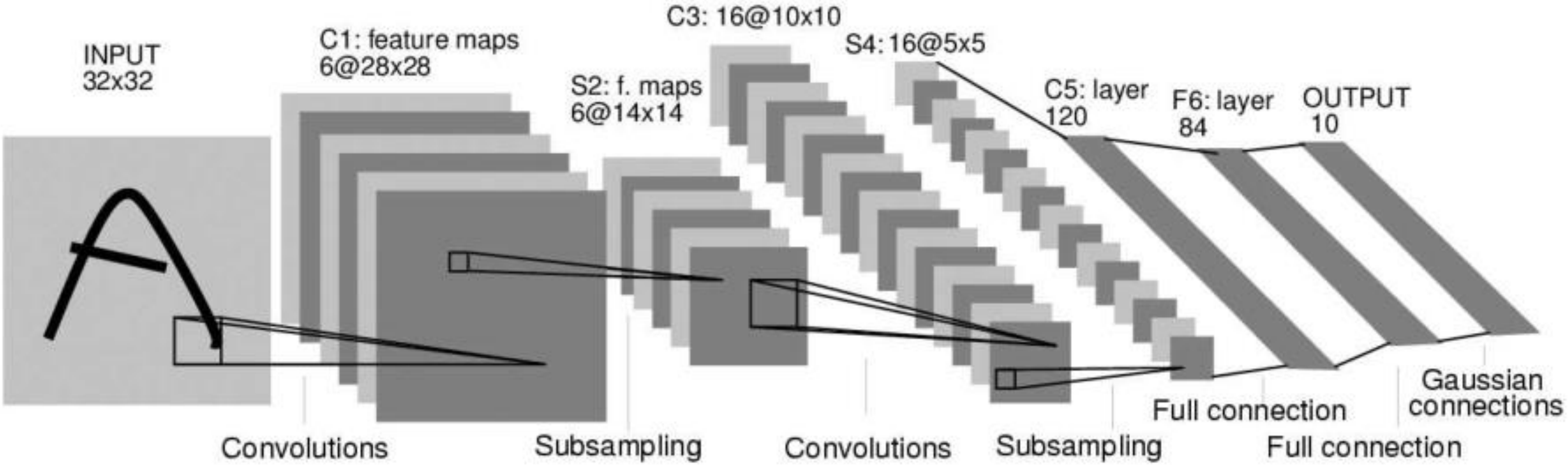
"Deep Learning" vs "Big Data" interest over time (from Google Trends)

Deep Learning : History and Building blocks

- What is "Deep Learning" and where does it come from ?
- Deep Learning for Image Analysis: the basics
- Building blocks
- Examples & practical overview

Deep Learning for Image Analysis

Deep **neural networks** with **convolutional layers** and **fully-connected layers** (or not).



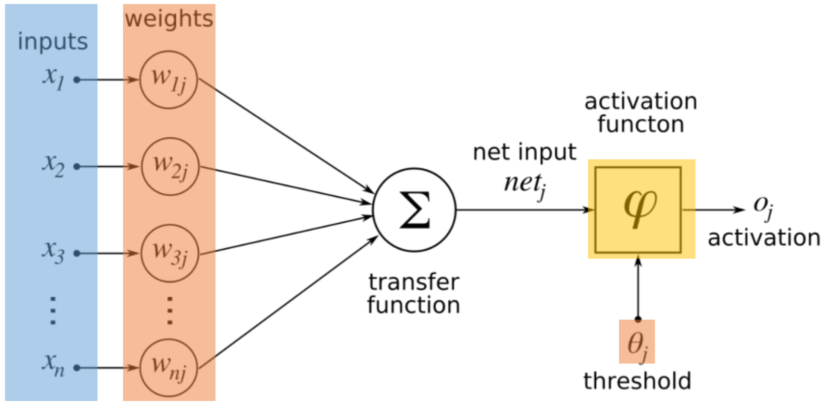
Y. LeCun et al., *Gradient-Based Learning Applied to Document Recognition*, Proc. of the IEEE, November 1998.

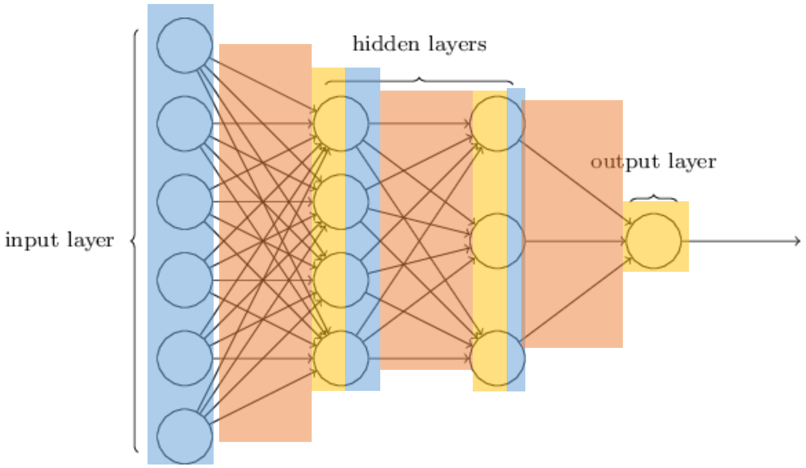
Fully Connected Layers (= MLP)

A **neuron** is defined by its **inputs** (incoming connections), its **weights**, and its **activation function**.

In a **fully-connected** layer, all outputs from layer n-1 are inputs of layer n.

$$o_j = \phi(\sum_i (w_{ij} x_i + b_j))$$



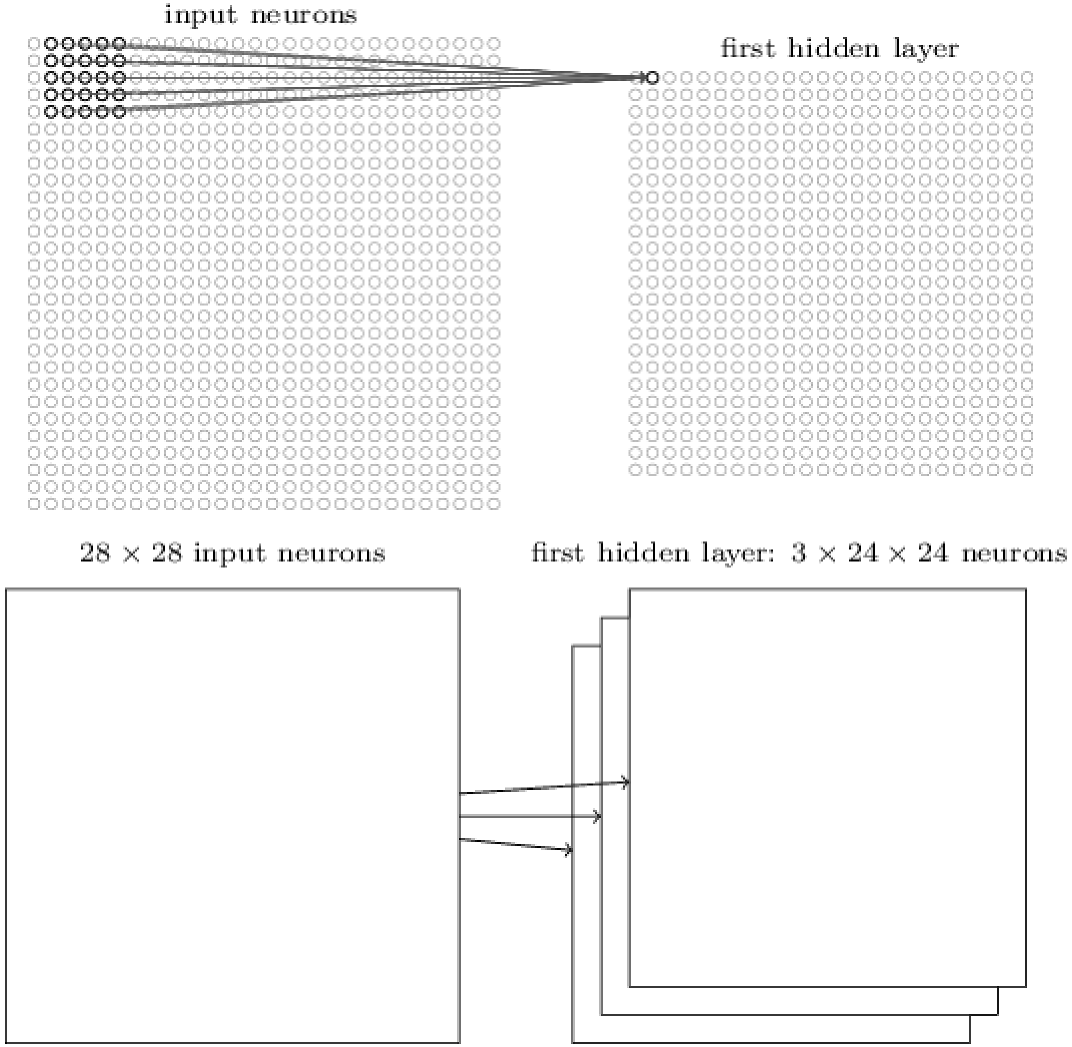


Convolutional Layers (shared weights)

In a **convolutional** layer, connections are only made in a **local receptive field**.

$$o_{l,m} = \phi(\sum_{ij}(w_{ij}x_{l+i,m+j} + b_{l,m}))$$

All neurons from the same **feature map** share the **same weights**. The feature maps are the result of the convolution of the input image by the weights of the connections.



Michael Nielsen, http://neuralnetworksanddeeplearning.com/chap6.html

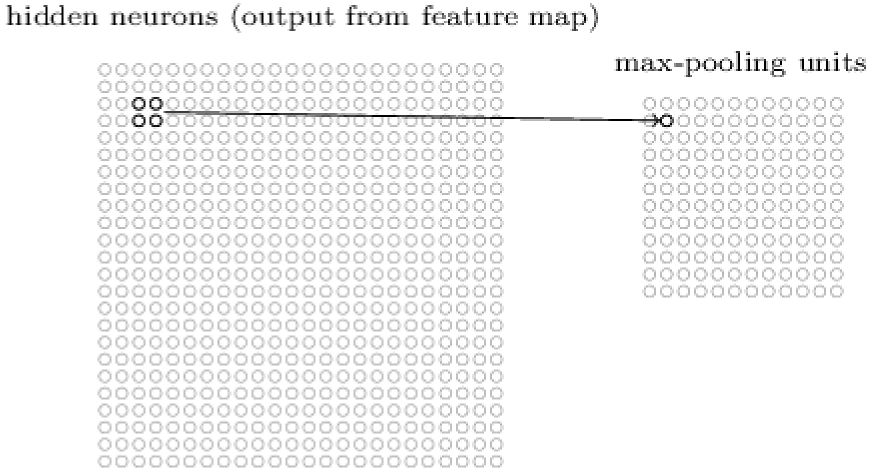
Convolutional Layers (shared weights)

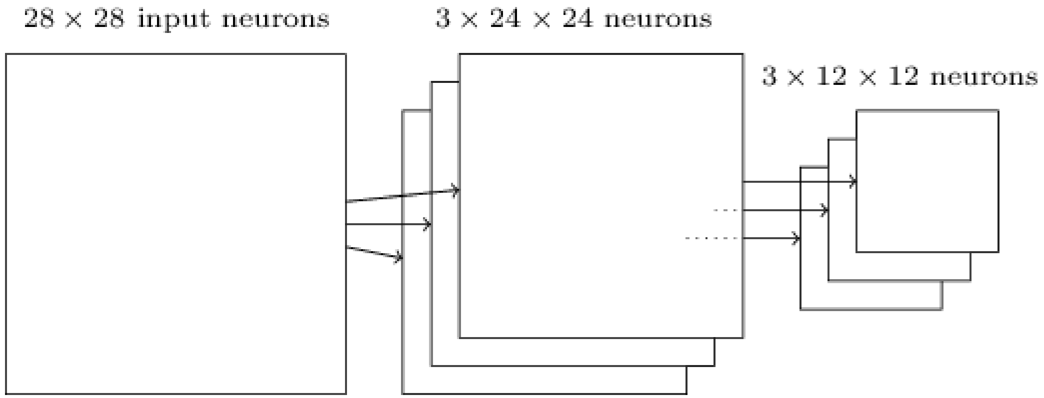
Why convolutional layers?

- Less parameters (→ faster convergence).
- Spatial relationships preserved.
- « It’s how vision works in our brains » (if we oversimplify it a lot).
- It gives really good results in computer vision problems.

Convolutional Layers (shared weights)

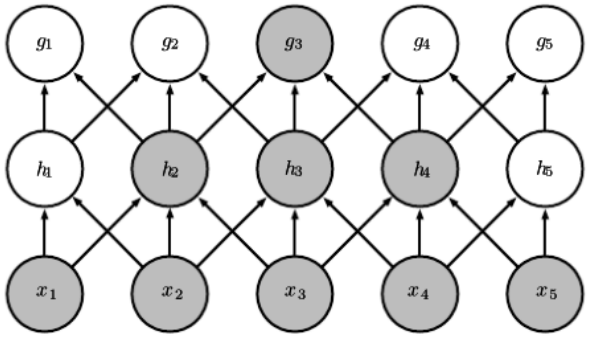
Convolutional layers are often used in conjunction with **pooling layers**.

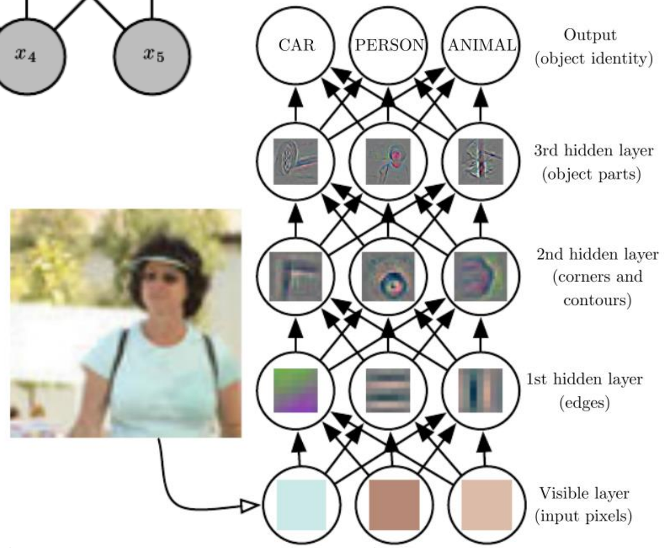




Michael Nielsen, http://neuralnetworksanddeeplearning.com/chap6.html

Goodfellow, Bengio, Courville. *Deep Learning*. MIT Press, 2016.

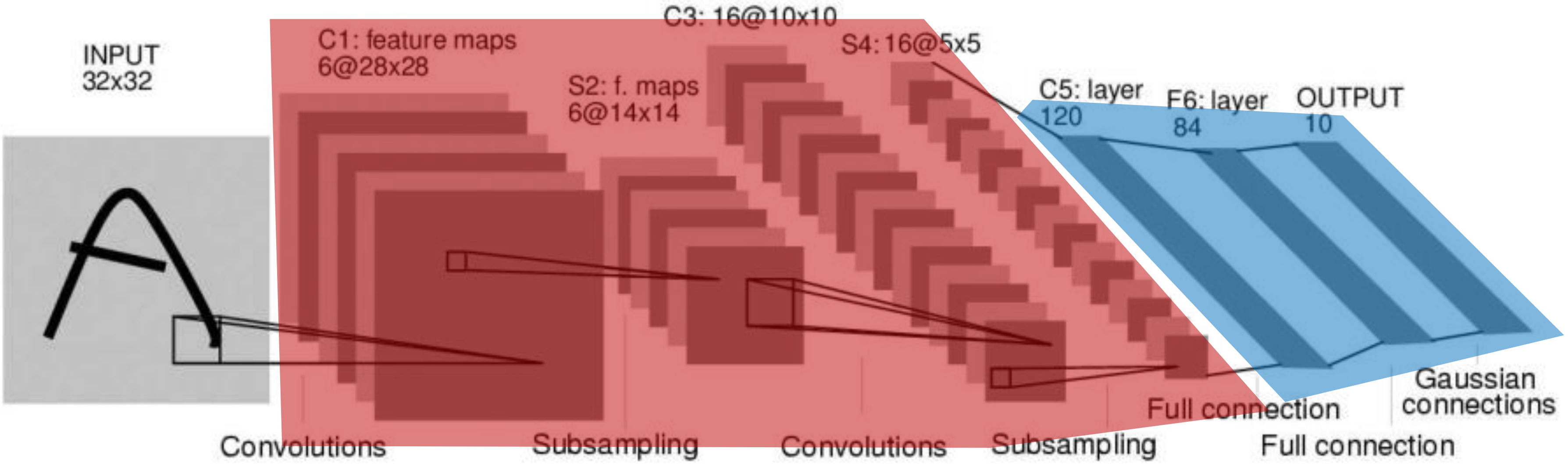




M.D. Zeiler and R. Fergus. *Visualizing and understanding convolutional networks*. ECCV'14.

Convolutional Layers (shared weights)

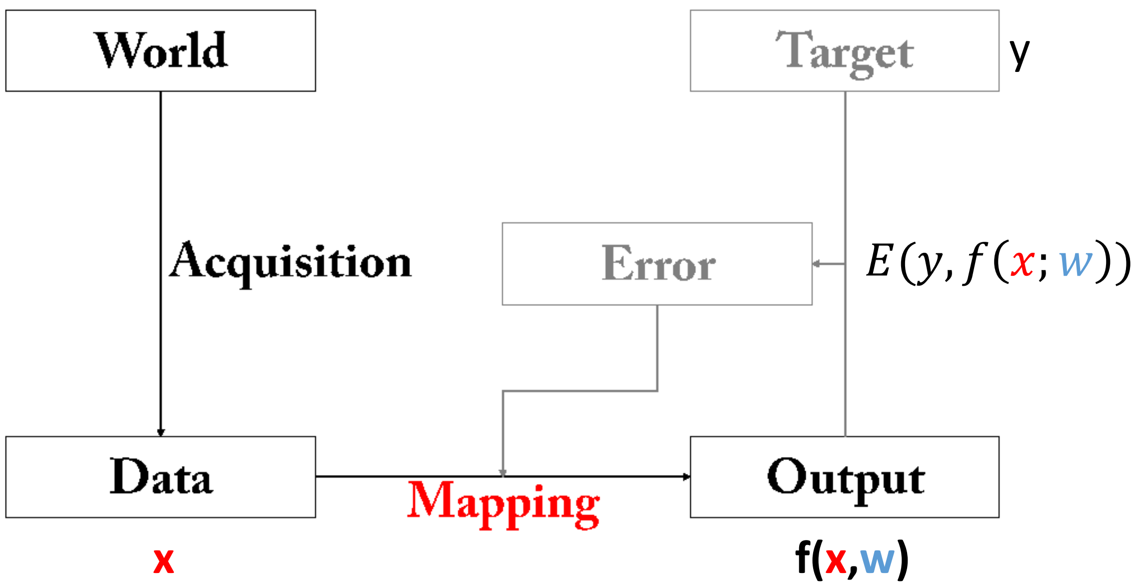
Convolutional layers and **pooling** layers form the **feature learning** part of the architecture. The features are then used by **fully connected** layers to obtain the **classification output**.



Y. LeCun et al., *Gradient-Based Learning Applied to Document Recognition*, Proc. of the IEEE, November 1998.

How does it learn?

Learning is an **optimization problem**. The most common method to solve these problems is **gradient descent**.

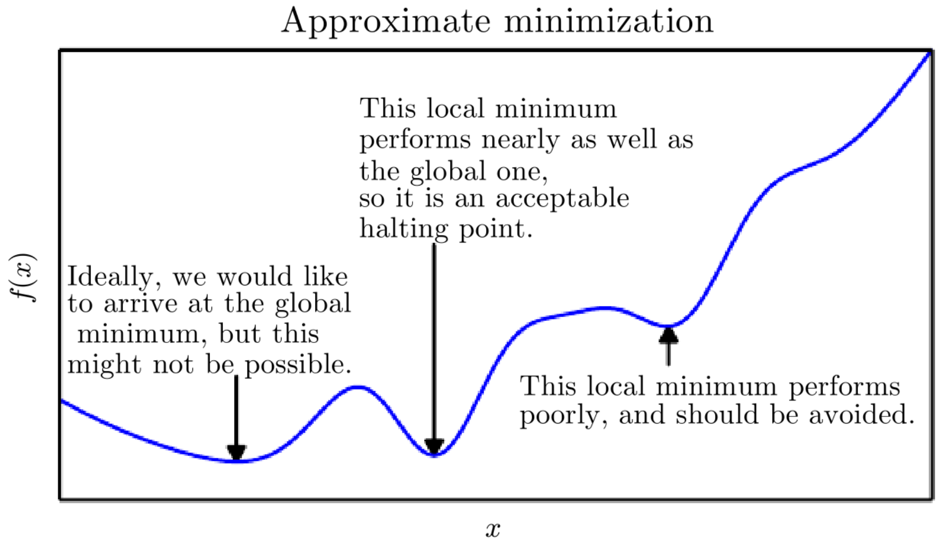


How does it learn?

Given the error function E: $E(y, f(x; w))$

Compute gradients relative to weights: $\frac{\partial E}{\partial w}$

Update weights: $w \leftarrow w - \eta \frac{\partial E}{\partial w}$

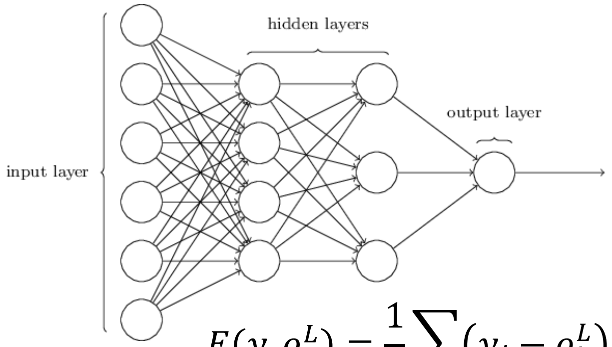


Goodfellow, Bengio, Courville. *Deep Learning*. [MIT Press](), 2016.

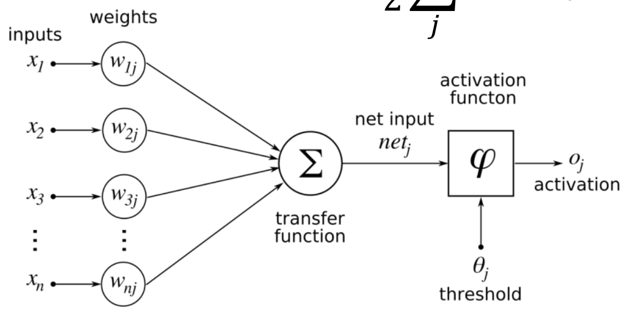
Problem: how do we compute those gradients?

How does it learn?

$$w \leftarrow w - \eta \frac{\partial E}{\partial w} \; ?$$



$$E(y, o^L) = \frac{1}{2}\sum_j (y_j - o_j^L)^2$$



$$net_j^l = \sum_i (w_{i,j} o_j^{l-1} + \theta_j^l)$$

$$o_j^l = \phi(net_j^l)$$

Compute error of the **output layer neurons**:

$$\delta_j^l = \frac{\partial E}{\partial net_j^l} = \frac{\partial E}{\partial o_j^L}\frac{\partial o_j^L}{\partial net_j^l} = \frac{\partial E}{\partial o_j^L}\phi'(net_j^L)$$

Back-propagate error to previous layers:

$$\delta_j^l = \sum_i w_{j,i}^{l+1}\delta_i^{l+1}\phi'(net_j^l)$$

Compute **gradient** relative to **weights** and **biases**:

$$\frac{\partial E}{\partial \theta_j^l} = \frac{\partial E}{\partial net_j^l}\frac{\partial net_j^l}{\partial \theta_j^l} = \delta_j^l.1$$

$$\frac{\partial E}{\partial w_{i,j}^l} = \frac{\partial E}{\partial net_j^l}\frac{\partial net_j^l}{\partial w_{i,j}^l} = \delta_j^l o_j^{l-1}$$

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

$$\theta \leftarrow \theta - \eta \frac{\partial E}{\partial \theta}$$

How does it learn?

If we have many layers, the gradient in the first layers will have terms like: $w^{l+1}w^{l+2}w^{l+3} \dots w^{l+n}\, \phi'(net^{l}\,)\phi'(net^{l+1})\phi'(net^{l+2}) \dots \phi'\,(net^{l+n-1})$

If the weights are < 1 and/or the derivative of the activation function is < 1, those terms will all be very close to zero. This is known as the **vanishing gradients problem**, and it can prevent networks from learning.

How does it learn?

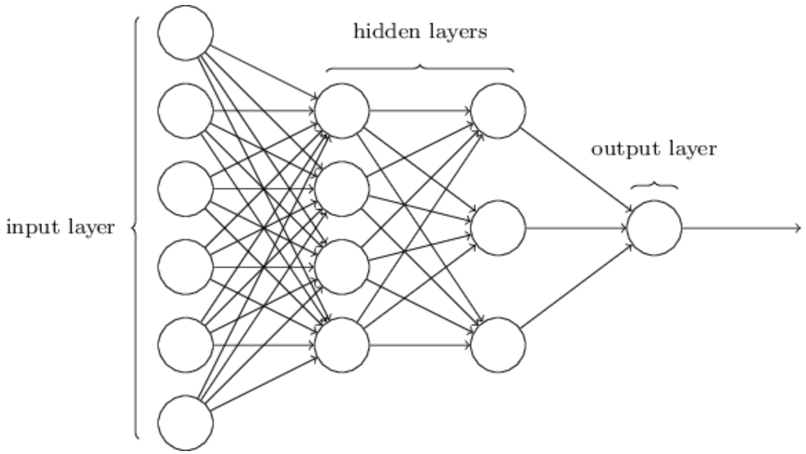
Solutions to the **vanishing gradient**:

- More data, more GPU, more time.
- Activation functions.
- Network architectures.
- Weight initialization, optimization algorithm…

Deep Learning : History and Building blocks

- What is "Deep Learning" and where does it come from ?
- Deep Learning for Image Analysis: the basics
- Building blocks
- Examples & practical overview

The Deep Learning Toolbox



Network architecture

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus Copyright © Sebastian Raschka 2016 (http://sebastianraschka.com)	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Activation functions

$$E\big(y, f(x; w)\big) = ?$$

Cost function

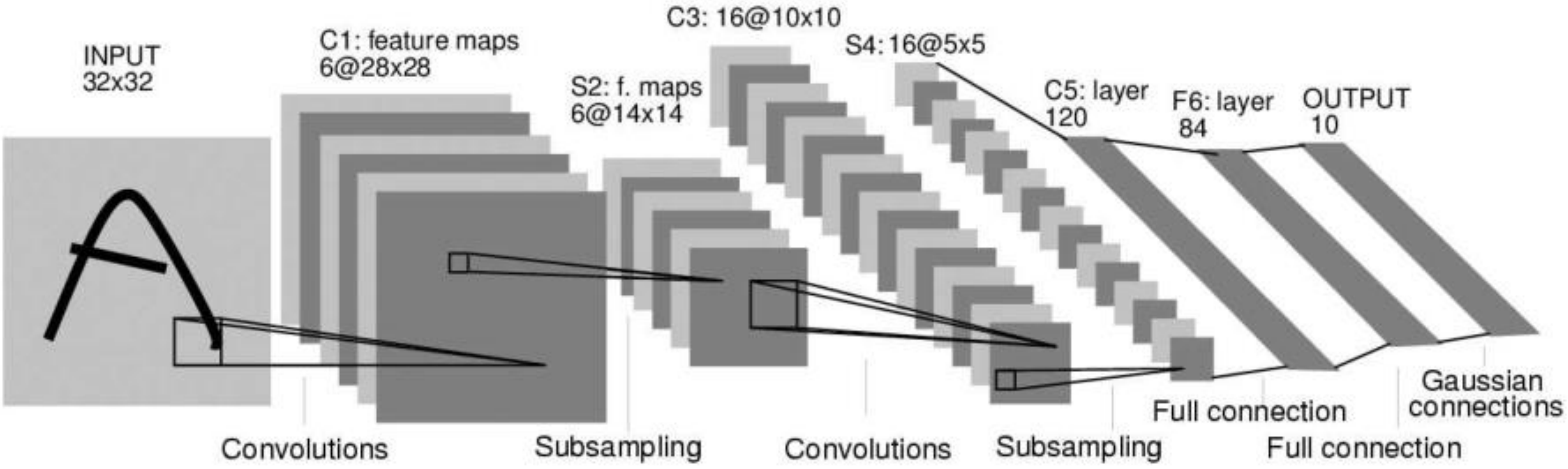
+ Regularization, initialization, optimization…

Network architectures

Input = 2D images

Output = Classification

Convolutions + MaxPooling + Fully-Connected

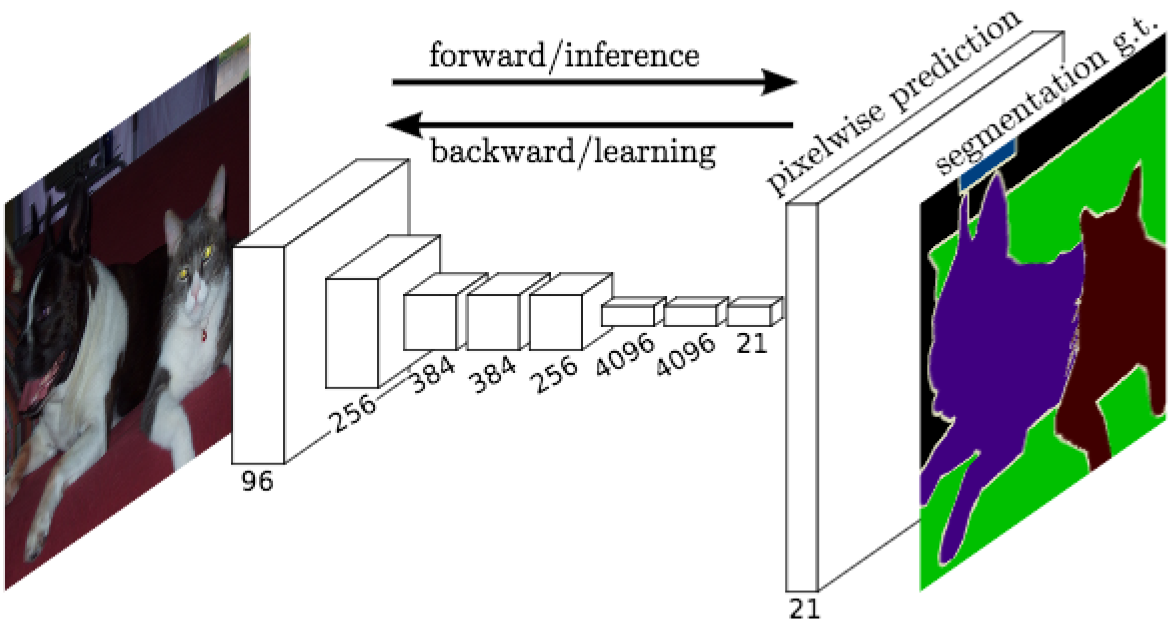


Network architectures

Input = 2D images

Output = Segmentation or Image

Fully-convolutional network: convolutions + maxpooling + upsampling

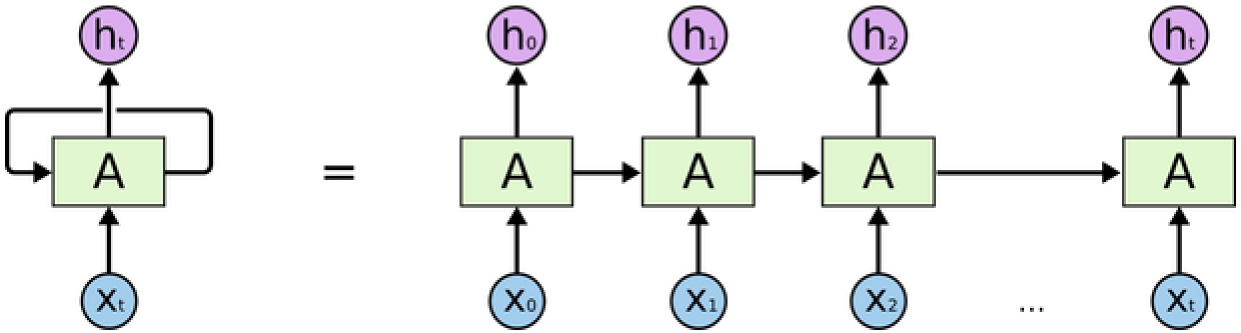


https://medium.com/self-driving-cars/literature-review-fully-convolutional-networks-d0a11fe0a7aa

Network architectures

Input = Data + Time

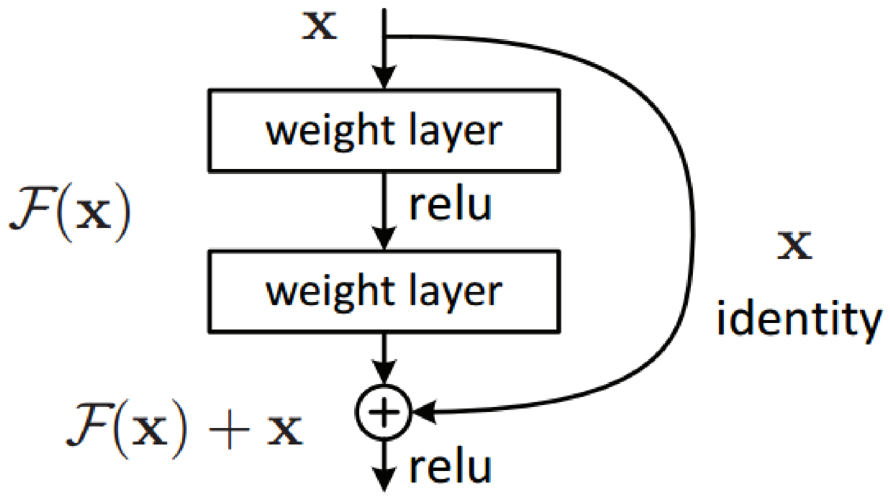
Recurrent Network / LSTM



https://pydeeplearning.weebly.com/blog/basic-architecture-of-rnn-and-lstm

Network architectures

Shortcuts through the network (e.g. Residual Units) -> faster convergence, less vanishing gradients.

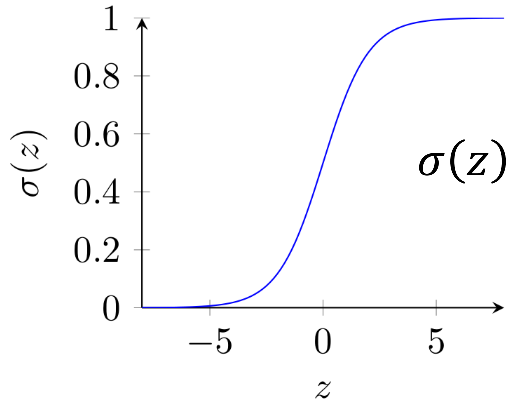


He et al. *Deep Residual Learning for Image Recognition*. 2015.

Activation functions

The classic: sigmoid function.

→ "soft threshold"

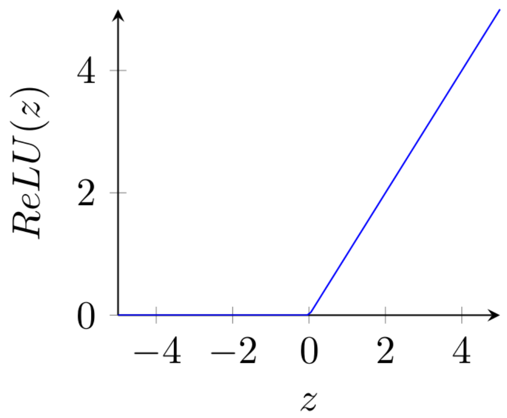


$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Small gradients (Derivative <= 0.25)
- Small "active" zone

Most common today: ReLU

$$ReLU(z) = \max(0, z)$$



- Derivative = 0 or 1
- Very fast to compute

Output layer: **softmax**.

→ logistic regression

→ $0 \leq y \leq 1$ and $\sum_i y_i = 1$

$$y_i(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Cost functions

Mean Square Error (if output = "image" -> regression problem)

$$E\big(y, f(x; w)\big) = -\frac{1}{2}\sum(y - f)^2$$

Cross-entropy (if output = logits/probabilities)

$$E\big(y, f(x; w)\big) = \sum f . log(y)$$

Can be more complex: weighted cross-entropy, multiple outputs/costs, etc...

+ Regularization

Regularization

Goal → prevent overfitting

- In the cost function : L1 (sparsity) or L2 (small weights) norm.

$$E^*\big(y, f(x; w)\big) = E + \alpha \sum_k \frac{1}{k} |w_k| \qquad\qquad E^*\big(y, f(x; w)\big) = E + \alpha \sum_k w_k^2$$

- In the dataset (data augmentation)
- In the architecture (dropout)
- In the optimization process (early stopping)

Initialization and stopping

What should be the **initial values** of the weights and biases?

- Small and random.
- Depend on the number of inputs & outputs
- E.g. Xavier Initializer, Variance Scaling Initializer…

When do we **stop the** optimization process?

- (Cross-)validation: when the validation accuracy stops rising.

Optimization

Variations on **gradient descent**:

- Momentums
- Adaptive learning rate

$$w \leftarrow w - \eta \frac{\partial E}{\partial w} + \ldots$$

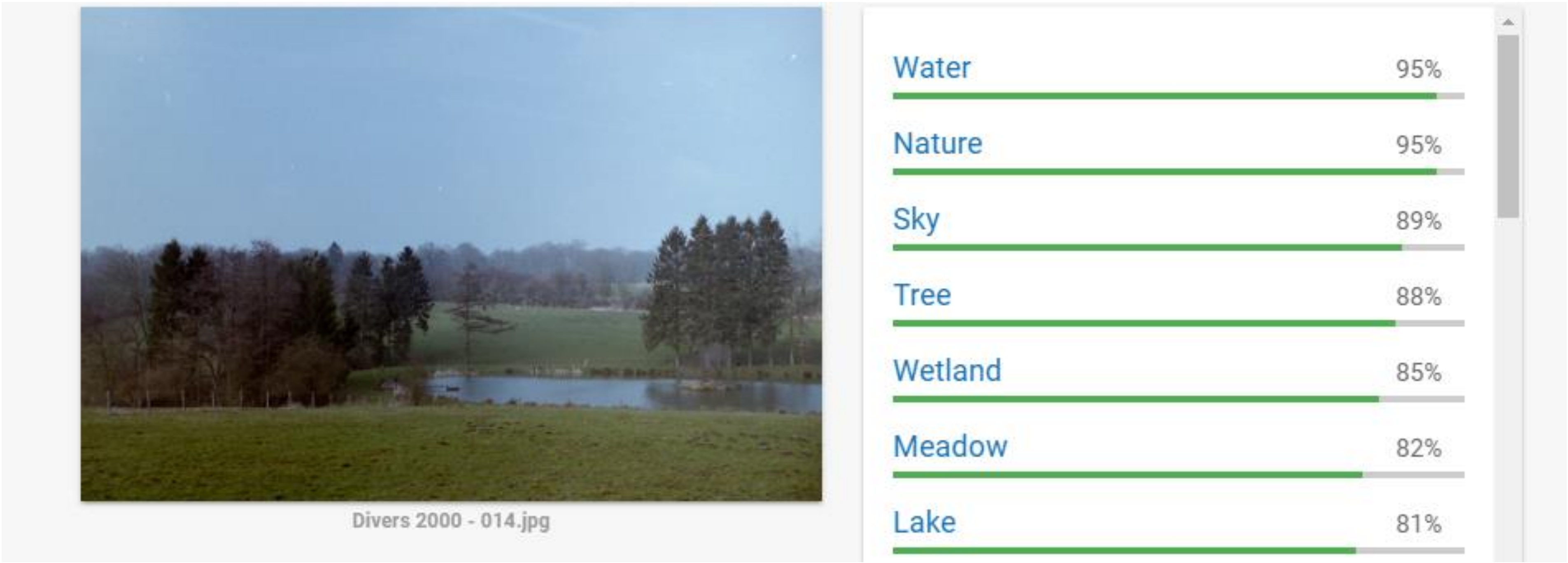
e.g. Adam, AdaDelta…

A good **initial learning rate** is very important to make sure that the network can learn. If it's too low, the weights will not change; if it's too high, it may quickly get stuck. For most networks, $\eta = 10^{-4}$ is a good bet.

Deep Learning : History and Building blocks

- What is "Deep Learning" and where does it come from ?
- Deep Learning for Image Analysis: the basics
- Building blocks
- Examples & practical overview

Example: Image classification / detection



https://cloud.google.com/vision/

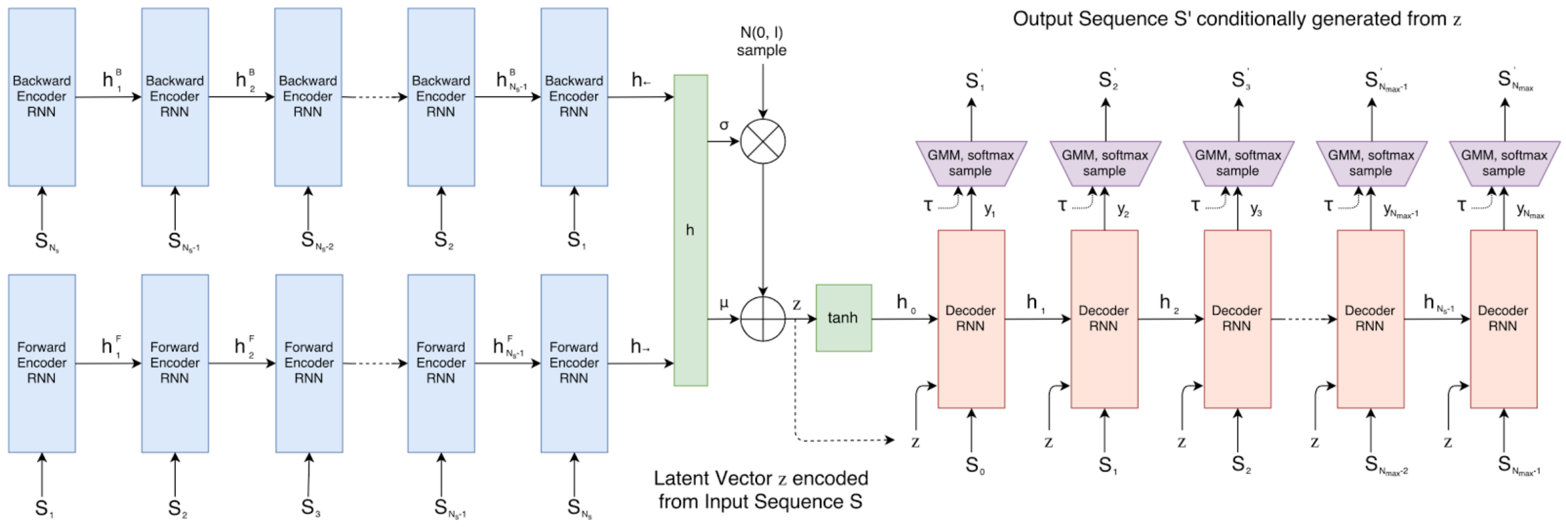
Input = image
Output = class detection probabilities
Convolutions + fully-connected layers

Example: Sketch generation

Input = sequence of vectors

Output = sequence of vectors

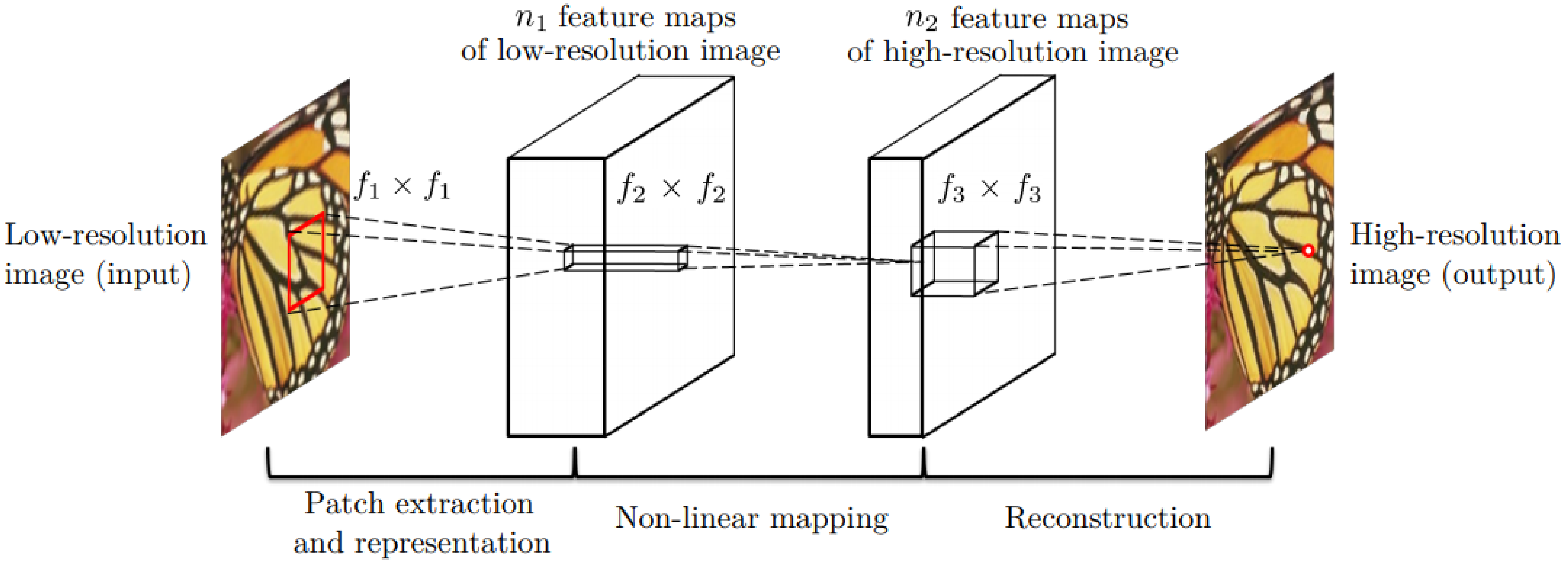
RNN



https://research.googleblog.com/2017/04/teaching-machines-to-draw.html

https://magenta.tensorflow.org/assets/sketch_rnn_demo/index.html

Example: Super-Resolution



Dong et al. *Image Super-Resolution Using Deep Convolutional Networks*. 2015. https://arxiv.org/abs/1501.00092

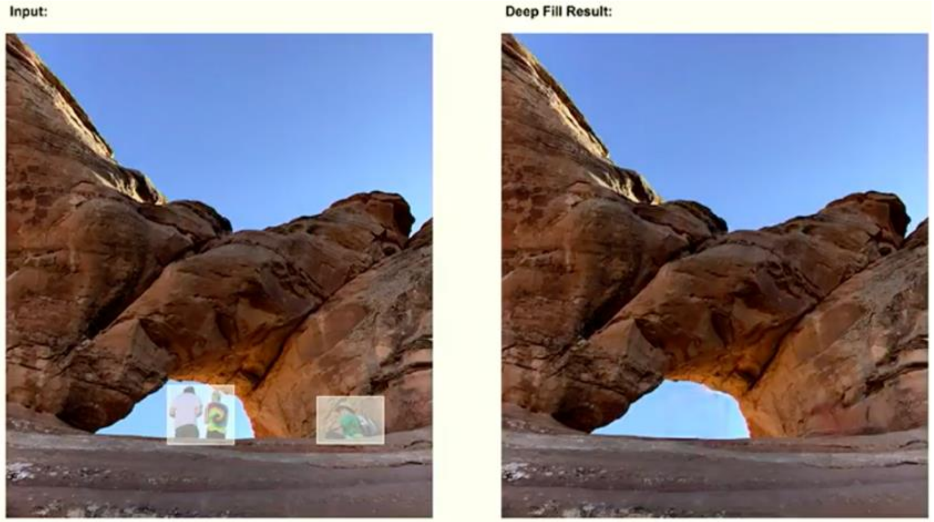
Input = low-resolution image

Output = high-resolution image

Fully-convolutional network

Example: Denoising/Inpainting

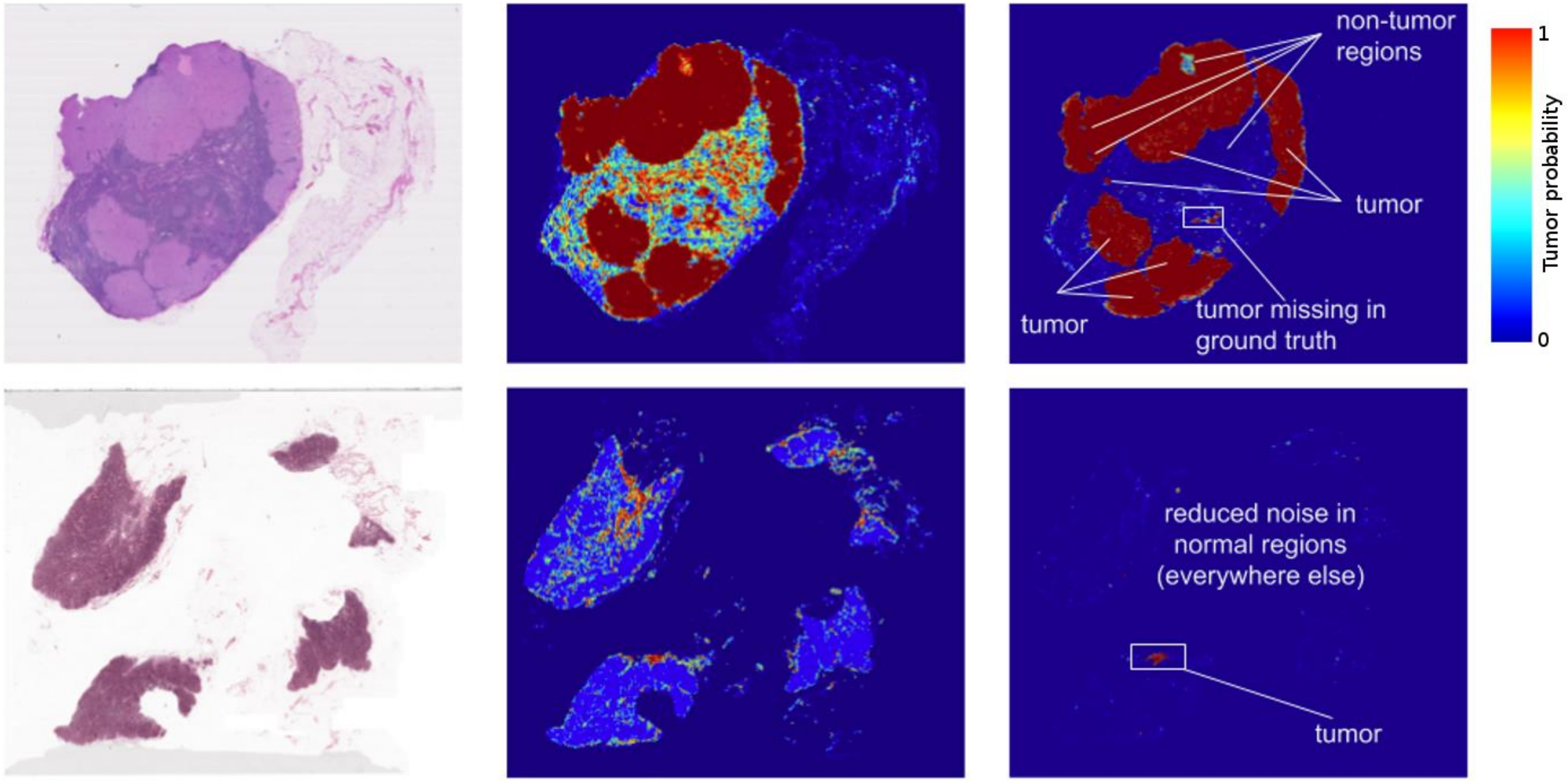
Mao et al. *Image Restoration Using Convolutional Auto-encoders with Symmetric Skip Connections*. 2016. https://arxiv.org/pdf/1606.08921.pdf



https://www.dpreview.com/news/2758068086/adobe-s-project-deep-fill-is-an-incredible-ai-powered-content-aware-fill

Input = noisy image
Output = "clean" image
Fully-convolutional network

Example: Tumour segmentation



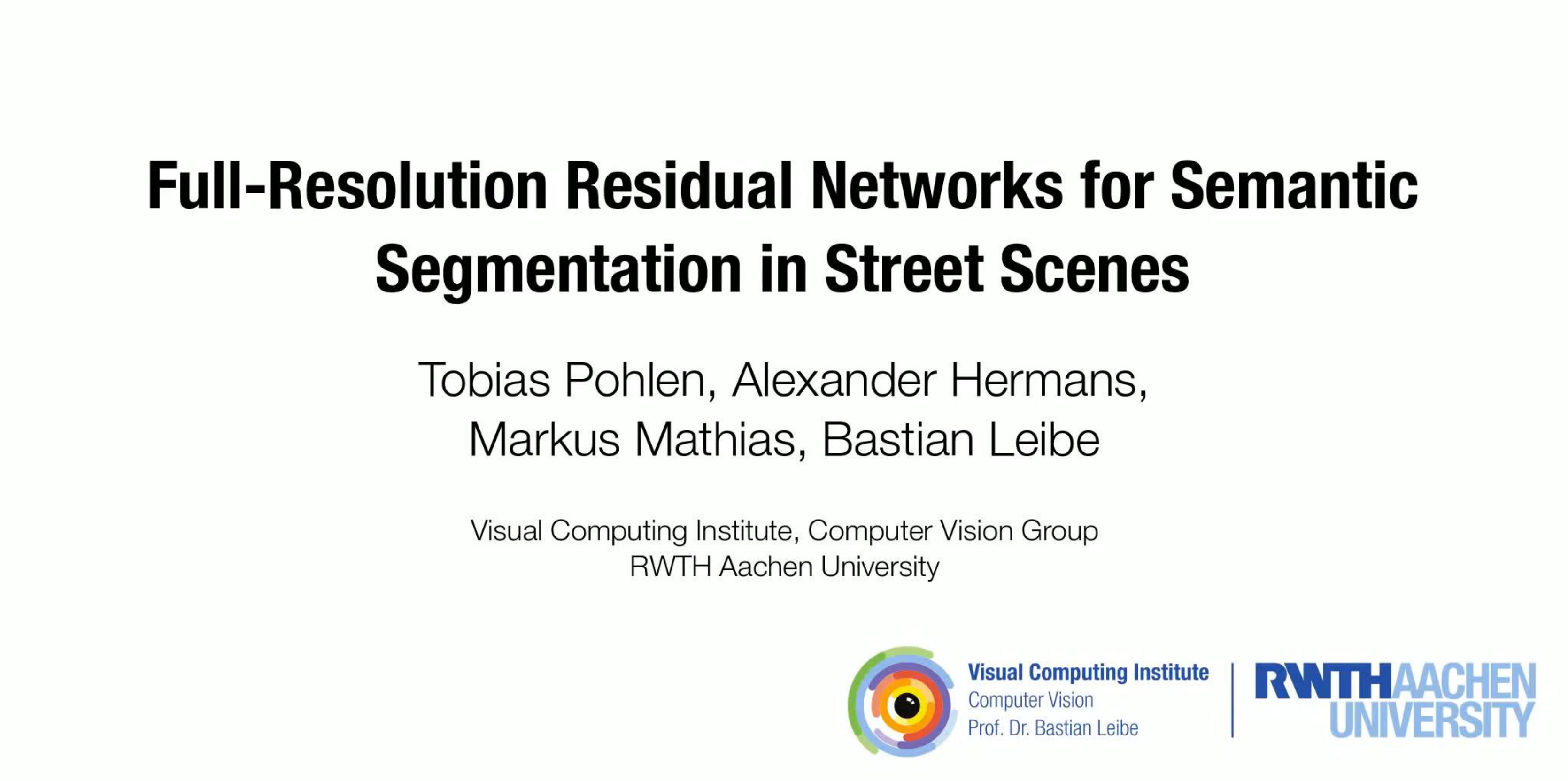
https://research.googleblog.com/2017/03/assisting-pathologists-in-detecting.html

Input = image patches

Output = full image segmentation probability map

Convolutions + fully-connected per patch with "sliding window"

Example: Image segmentation



Input = images

Output = pixel multi-class segmentation

Fully-convolutional network w/ residual units

https://www.youtube.com/watch?v=PNzQ4PNZSzc

https://arxiv.org/pdf/1611.08323.pdf

The pros and cons of Deep Learning

- Lots of ressources necessary
- Lots of data necessary
- Difficult to interpret the parameters ("black box" effect)
- Difficult to find "the best way to approach a problem"
- Takes a lot of time to train

+ Works really, really well !
+ Solves problems which are difficult to approach with traditional methods.
+ Able to generalize well from difficult datasets.

Further reading…

Michael Nielsen. *Neural Networks and Deep Learning*. Online book. http://neuralnetworksanddeeplearning.com/

Goodfellow et al. *Deep Learning*. MIT Press. http://www.deeplearningbook.org/

Jürgen Schmidhuber. *Deep learning in neural networks: An overview*. Neural Networks, 2014. https://www.sciencedirect.com/science/article/pii/S0893608014002135

LeCun, Bengio and Hinton. *Deep learning*. Nature, 2015. https://www.nature.com/articles/nature14539

Sze, Chen, Yang and Elmer. *Efficient Processing of Deep Neural Networks: a Tutorial and Survey*. 2017. https://arxiv.org/abs/1703.09039

Zeiler and Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. https://arxiv.org/abs/1311.2901

Practical overview

theano PYTORCH Caffe

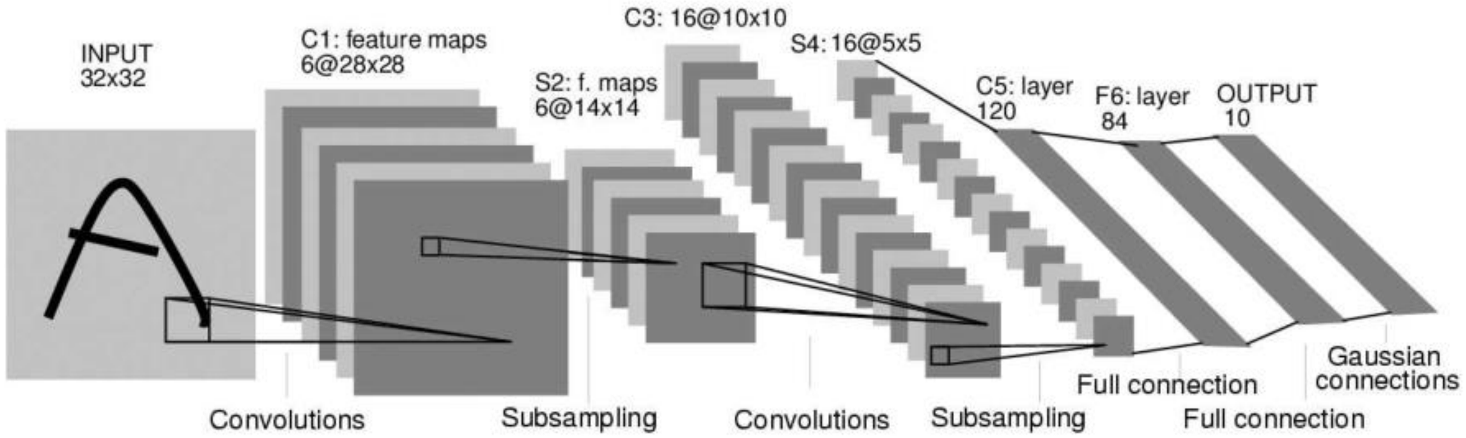
Keras TensorFlow

Low-level libraries: creating the network from the base mathematical operations.

High-level libraries: defining the network architecture using standard blocks

All: GPU optimization

Practical overview




```
X = tf.placeholder(tf.float32, [None, 32, 32], name='input')
Y = tf.placeholder(tf.float32, [None, 10], name='target')

net = tf.layers.conv2d(X, 6, 5, activation=tf.nn.tanh)
net = tf.layers.max_pool2d(net, 2, 2)
net = tf.layers.conv2d(net, 16, 5,  activationn=tf.nn.tanh)
net = tf.layers.max_pool2d(net, 2, 2)
net = tf.reshape(net, [-1, int(np.prod(net.get_shape()[1:]))])
net = tf.layers.dense(net, 120, activation=tf.nn.tanh)
net = tf.layers.dense(net, 84, activation=tf.nn.tanh)
net = tf.layers.dense(net, 10, activation=tf.nn.tanh)
```

Practical overview

```
X = tf.placeholder(tf.float32, [None, 32, 32], name='input')
Y = tf.placeholder(tf.float32, [None, 10], name='target')

net = tf.layers.conv2d(X, 6, 5, activation=tf.nn.tanh)
net = tf.layers.max_pool2d(net, 2, 2)
net = tf.layers.conv2d(net, 16, 5,  activationn=tf.nn.tanh)
net = tf.layers.max_pool2d(net, 2, 2)
net = tf.reshape(net, [-1, int(np.prod(net.get_shape()[1:]))])
net = tf.layers.dense(net, 120, activation=tf.nn.tanh)
net = tf.layers.dense(net, 84, activation=tf.nn.tanh)
net = tf.layers.dense(net, 10, activation=tf.nn.tanh)
```

```
loss = tf.reduce_sum(tf.nn.softmax_cross_entropy_with_logits(logits=net, labels=Y))
optimizer = tf.train.GradientDescentOptimizer(1e-4)
trainingStep = optimizer.minimize(loss)
```

```
for e in range(N_EPOCHS):
    for data_x,data_y in DATASET:
        trainingStep.run(session=sess, feed_dict={X: data_x, Y: data_y})
```

```
net.eval(session=sess, feed_dict={X: test_x})
```

To conclude

Deep Learning is still Machine Learning. Setting up a good Machine Learning / Image analysis workflow often remains the most important & difficult part of the process.

- Good handling of the **dataset** (training / validation / test)
- **Pre-processing** & **post-processing** if necessary
- **Loss function** adapted to the problem you're trying to solve.