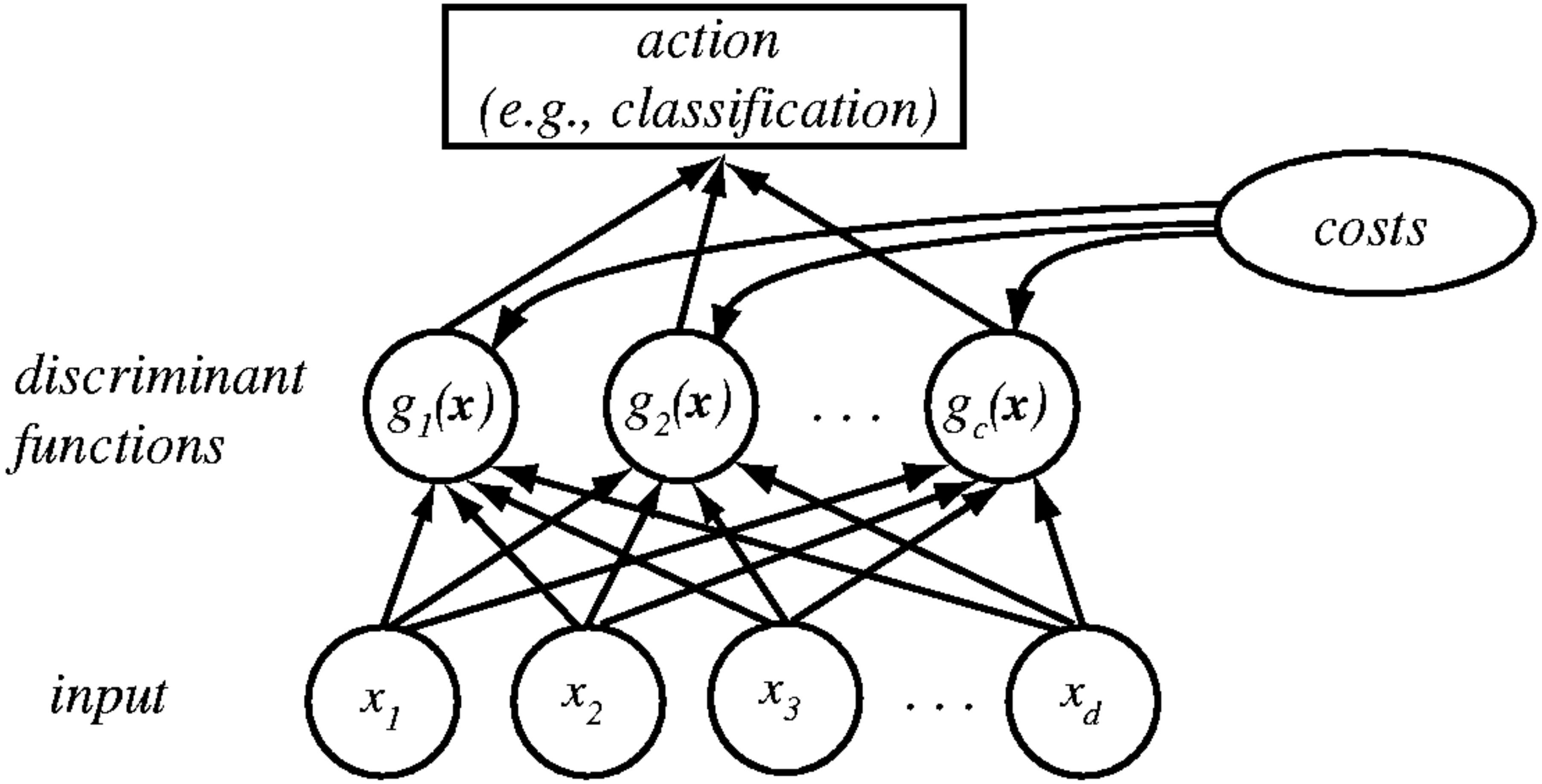
Deep Learning and Convolutional Neural Networks

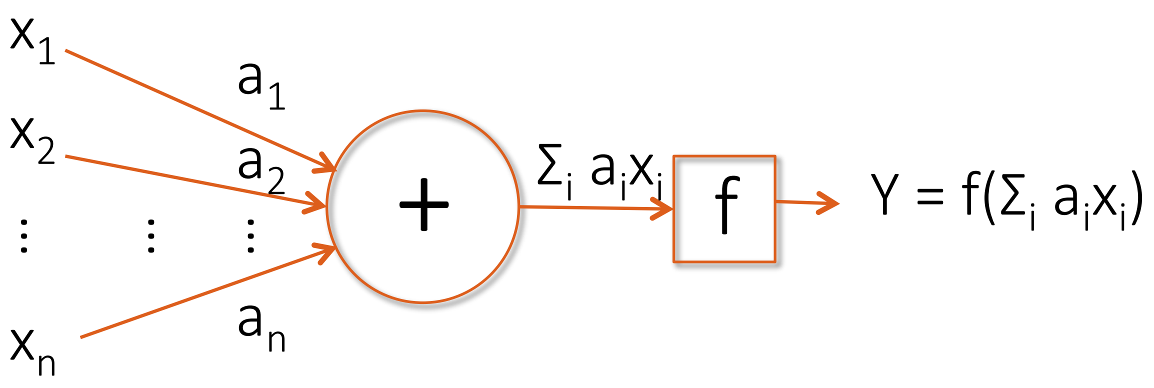
N. Petkov

Classification with discriminant functions



(from Duda, Hart, Stork (2001) Pattern classification)

Perceptron [Rosenblatt, 1957]



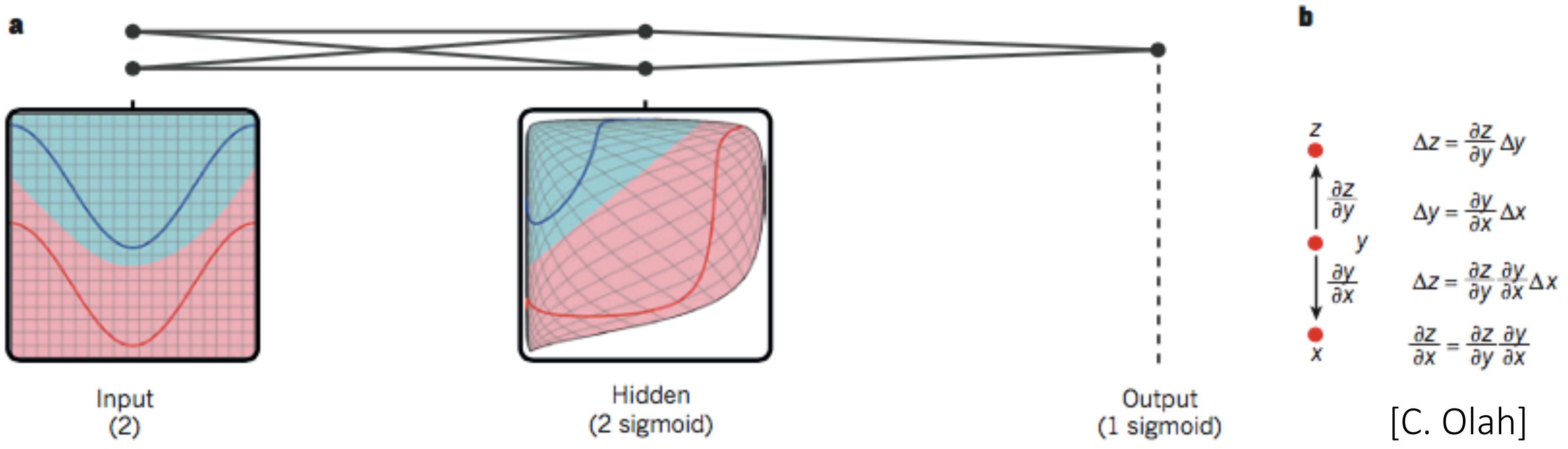
Simplest type of a trainable discriminant function with internal parameters.

Linear combination of the input followed by some non-linearity, e.g. squashing function f: $Y = f(\Sigma_i a_i x_i)$

Drawbacks:

- only for linearly separable classes (example: not for XOR function)
- cannot deal with input variance, e.g. shift of an image

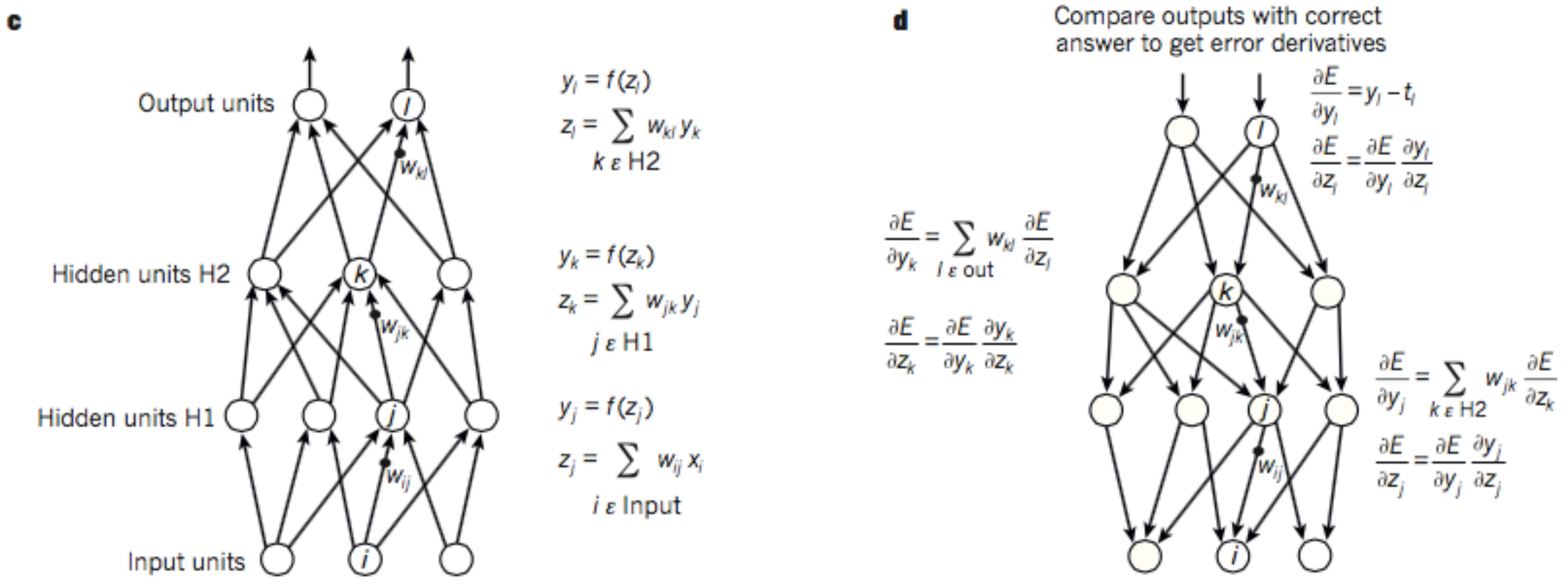
Networks with a hidden layer


a
Input
(2)
Hidden
(2 sigmoid)
Output
(1 sigmoid)
b
z
y
x
$\frac{\partial z}{\partial y}$
$\frac{\partial y}{\partial x}$
$\Delta z = \frac{\partial z}{\partial y}\Delta y$
$\Delta y = \frac{\partial y}{\partial x}\Delta x$
$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$
$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$

[C. Olah]

Two classes that are not linearly separable in the original raw-data space become linearly separable in the hidden layer representation due to a non-liner transform.

Multi-layer feedforward networks


c
Output units
Hidden units H2
Hidden units H1
Input units
$y_l = f(z_l)$
$z_l = \sum_{k \in H2} w_{kl} y_k$
$y_k = f(z_k)$
$z_k = \sum_{j \in H1} w_{jk} y_j$
$y_j = f(z_j)$
$z_j = \sum_{i \in \text{Input}} w_{ij} x_i$
d
Compare outputs with correct answer to get error derivatives
$\frac{\partial E}{\partial y_l} = y_l - t_l$
$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$
$\frac{\partial E}{\partial y_k} = \sum_{l \in \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$
$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$
$\frac{\partial E}{\partial y_j} = \sum_{k \in H2} w_{jk} \frac{\partial E}{\partial z_k}$
$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$

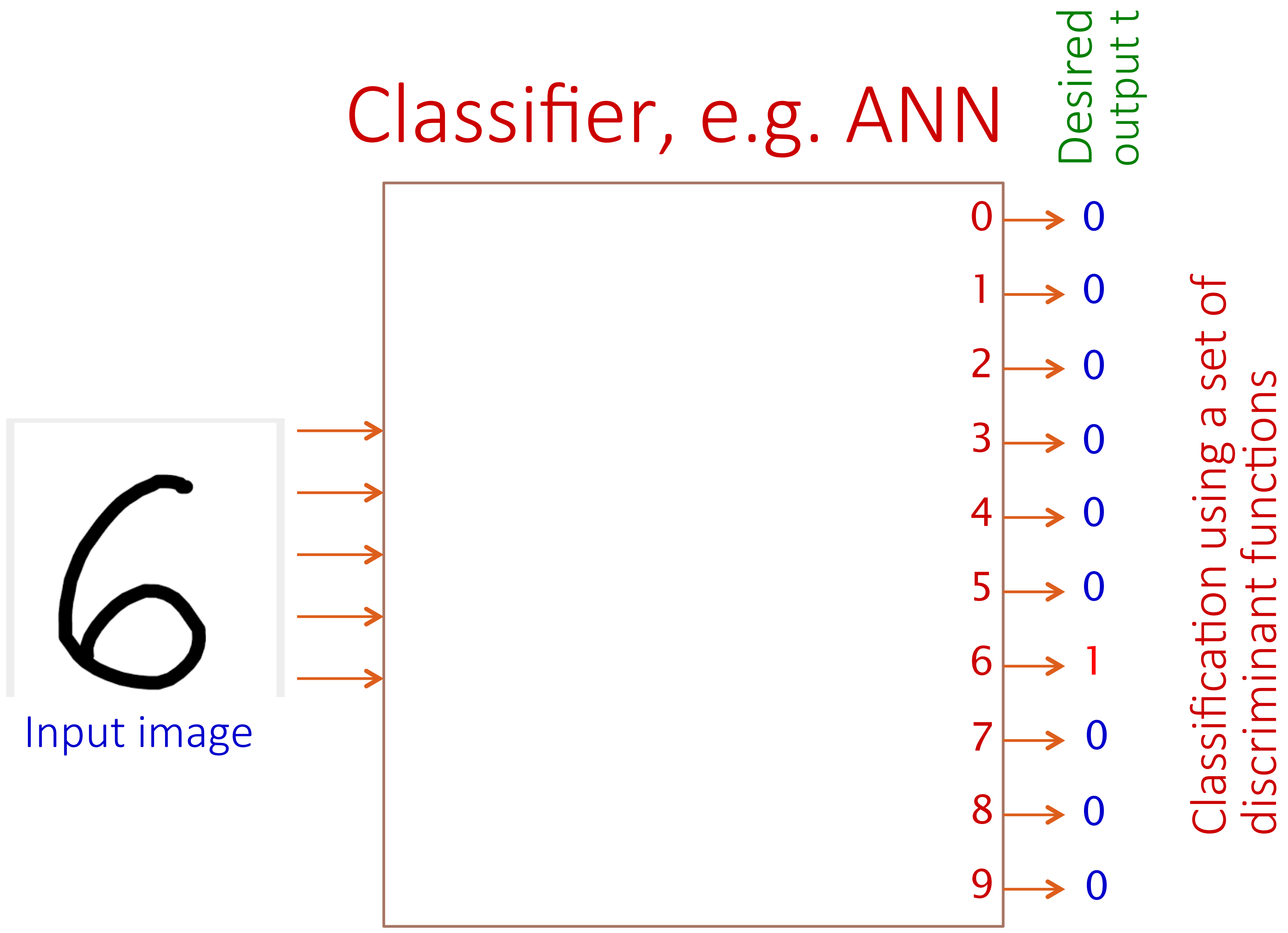
[LeCun et al.]

ANN - Artificial Neural Network

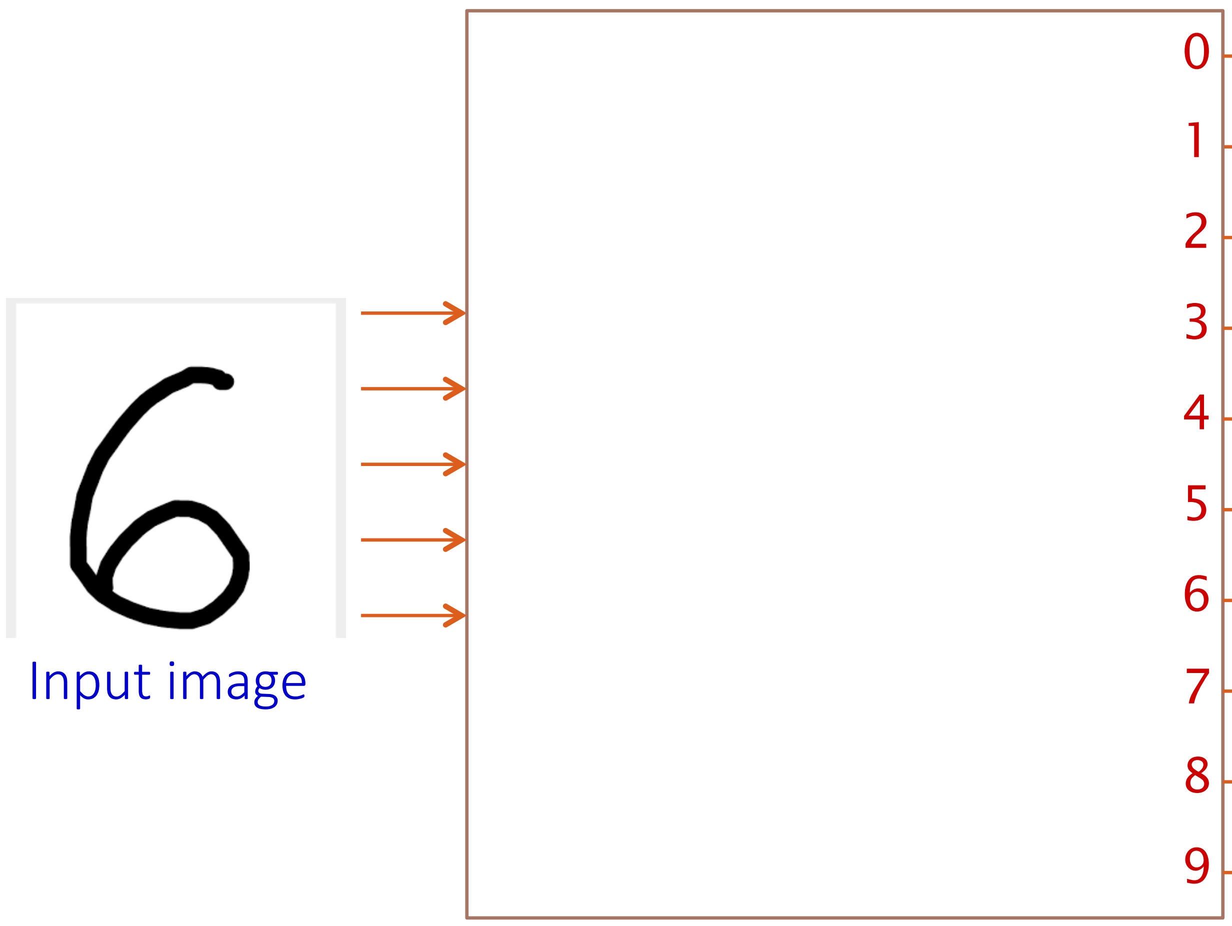
Learning/training = Adjustment of weights by

- error back-propagation [Rumelhart et al., 1986]
- some other method, e.g. stochastic gradient descent

How to train a classifier?

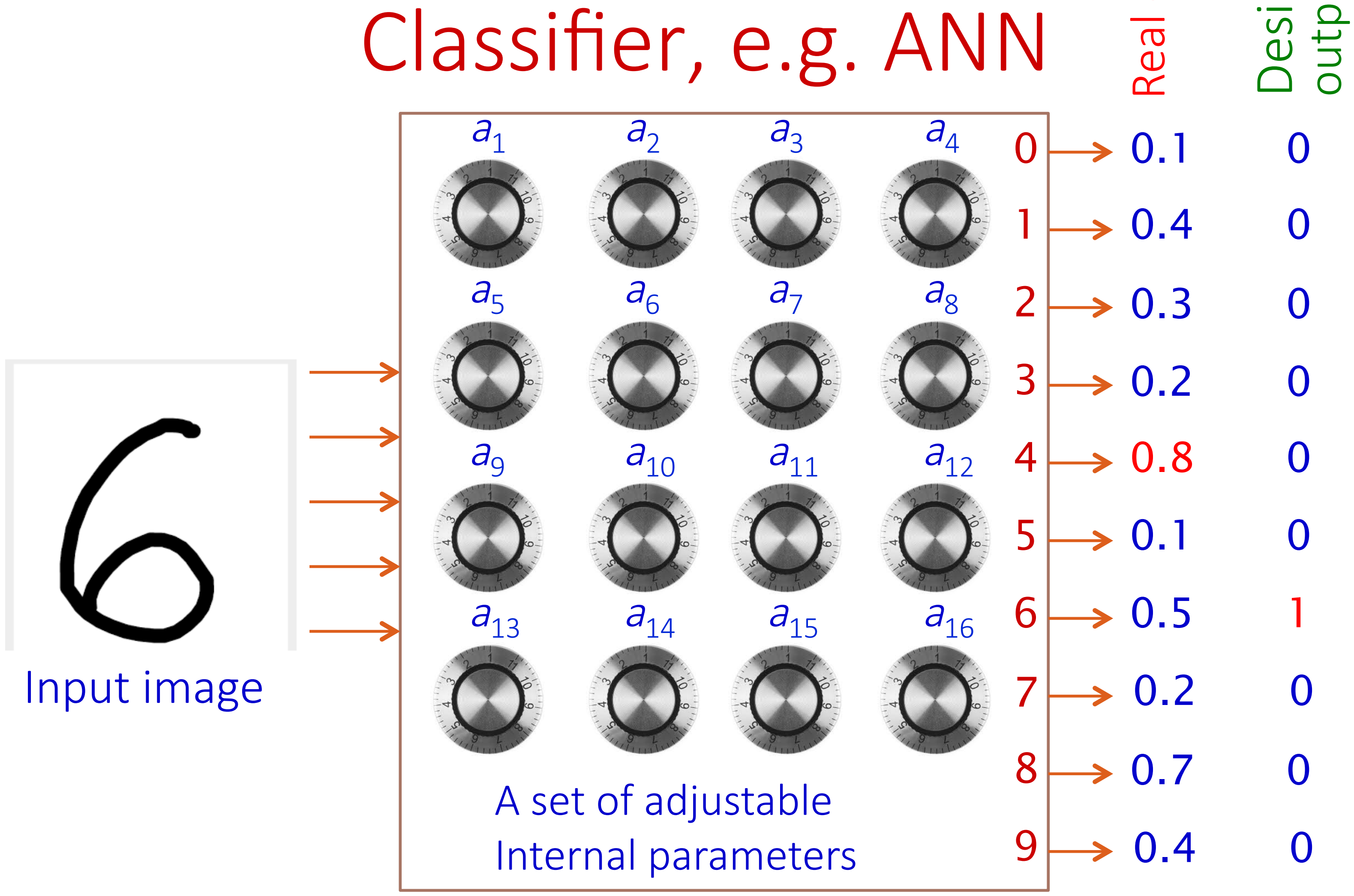


How to train a classifier?



	Real output y	Desired output t
0	0.1	0
1	0.4	0
2	0.3	0
3	0.2	0
4	0.8	0
5	0.1	0
6	0.5	1
7	0.2	0
8	0.7	0
9	0.4	0

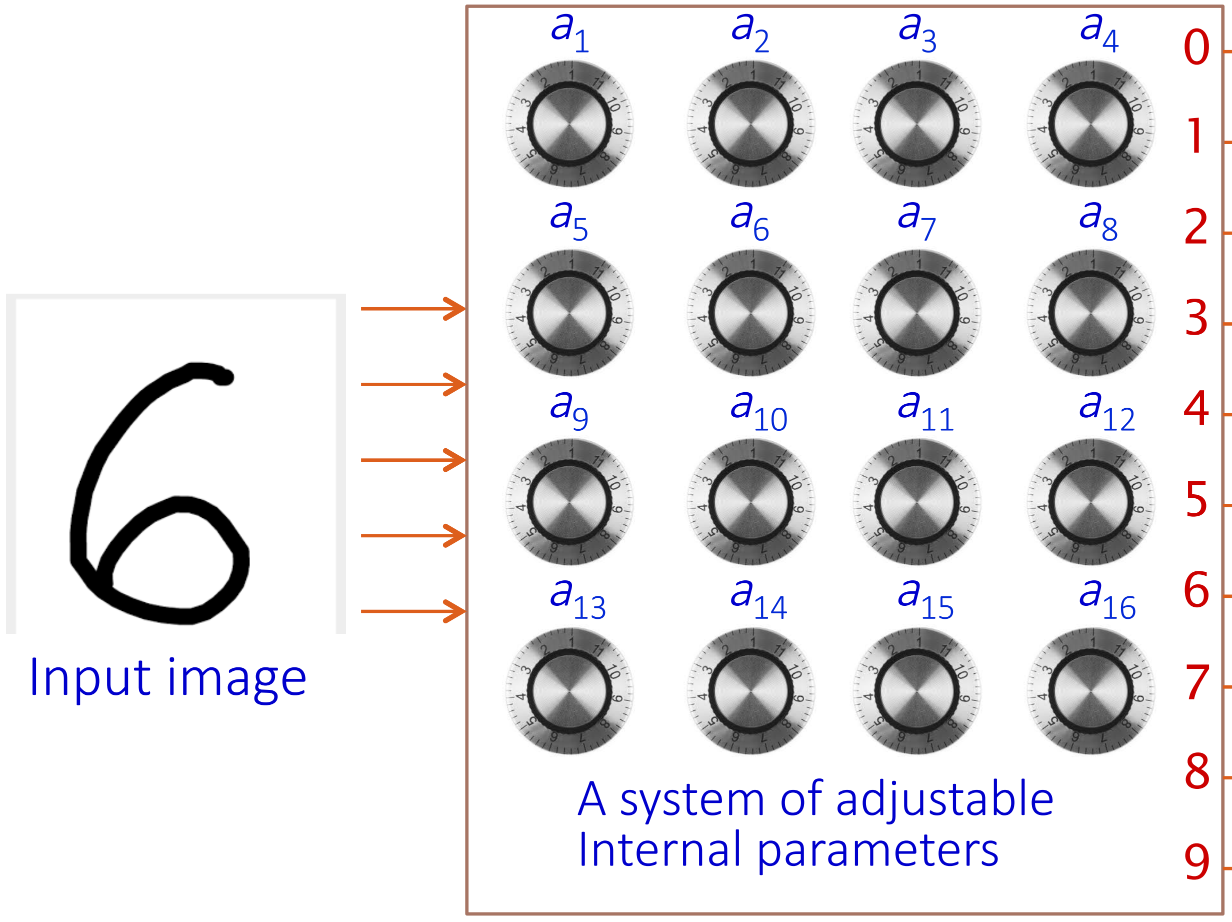
How to train a classifier?



	Real output y	Desired output t
0	0.1	0
1	0.4	0
2	0.3	0
3	0.2	0
4	0.8	0
5	0.1	0
6	0.5	1
7	0.2	0
8	0.7	0
9	0.4	0

Supervised learning by stochastic gradient descent

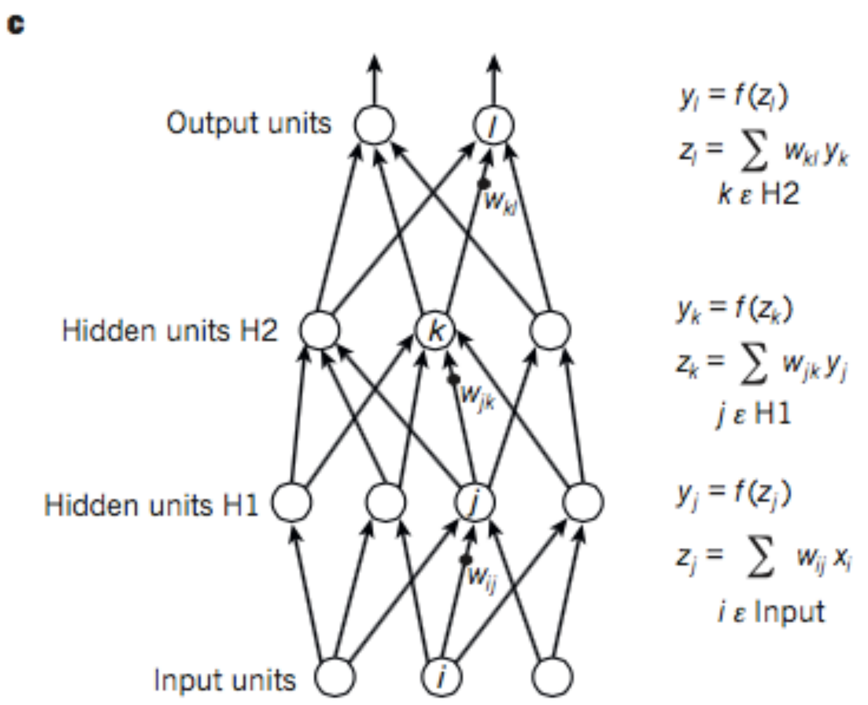
Classifier, e.g. ANN

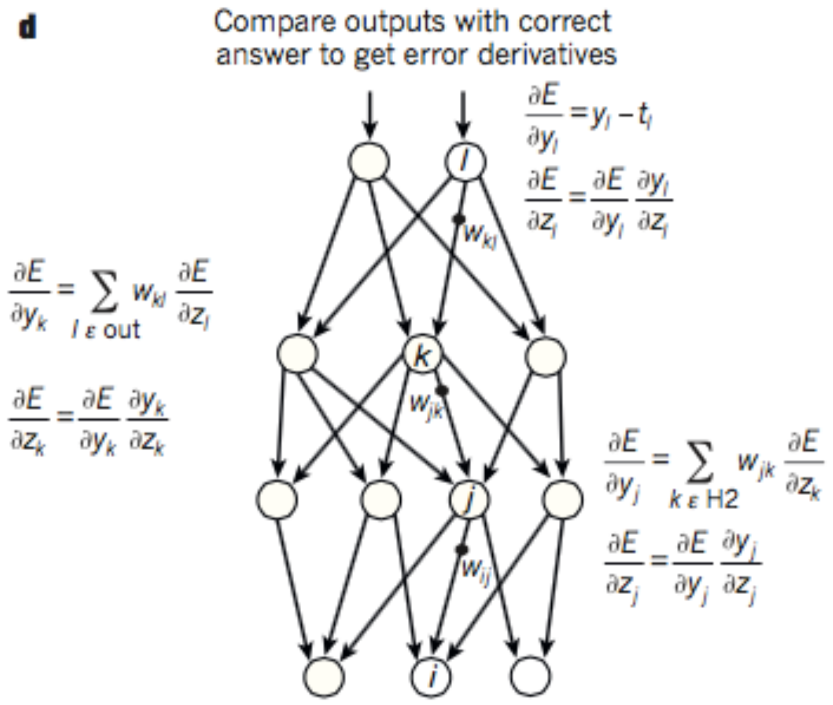


	Real output y	Desired output t
0	0.1	0
1	0.4	0
2	0.3	0
3	0.2	0
4	0.8	0
5	0.1	0
6	0.5	1
7	0.2	0
8	0.7	0
9	0.4	0

Use grad$_a$(d) to adjust a_1, a_2, … ⇐ Error d(t−y)

Multi-layer feedforward networks



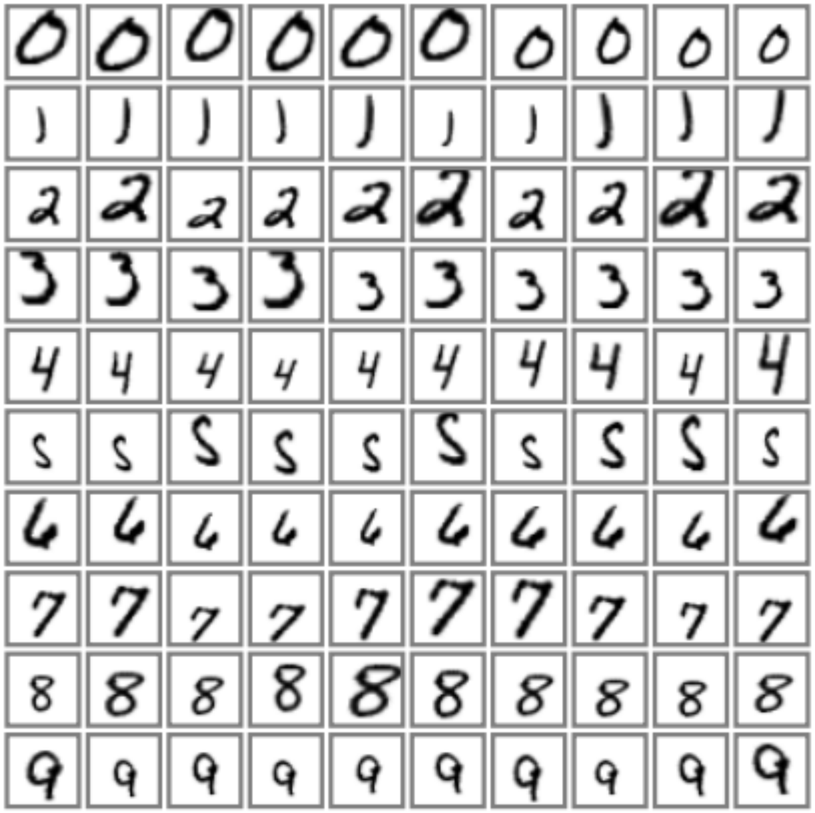

d
Compare outputs with correct answer to get error derivatives
$\frac{\partial E}{\partial y_l} = y_l - t_l$
$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$
$\frac{\partial E}{\partial y_k} = \sum_{l \in \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$
$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$
$\frac{\partial E}{\partial y_j} = \sum_{k \in H2} w_{jk} \frac{\partial E}{\partial z_k}$
$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$
l
k
j
i
w_{kl}
w_{jk}
w_{ij}

[LeCun et al.]

- Great enthusiasm for ANN 1985-1995(?),
- Cool-down after 1995: ANNs could not deal with raw data in many situations, e.g. images

Convolutional Neural Networks (CNN/ConvNet)

Handwritten digit recognition



MNIST data set

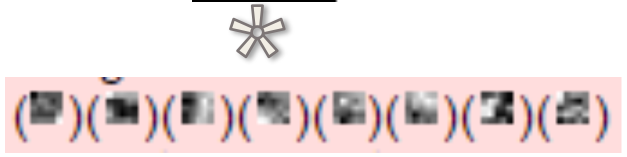
~60,000 training images
~10,000 test images

First paper on CNN trained by back-propagation [LeCun et al., 1990]

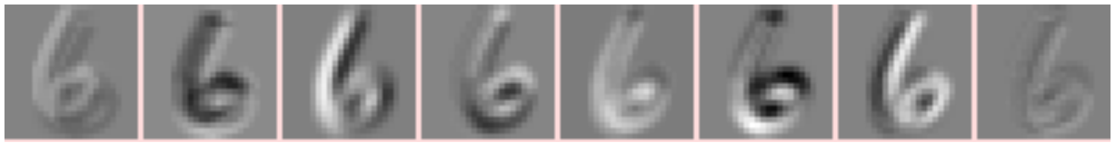
(1st) CNN for handwritten digit recognition

Processing | Representation

Input image (24x24)

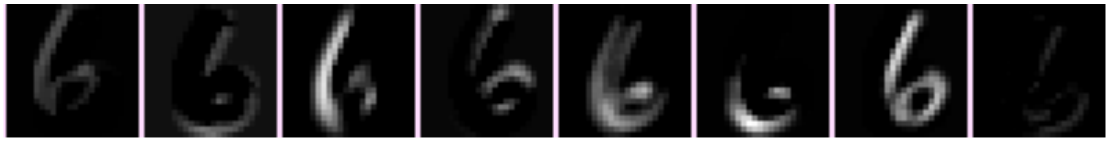


CONV — Convolution with 8 filters of size 5x5x1



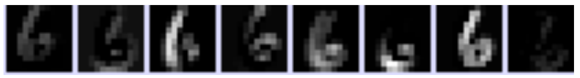
Convolution results (24x24x8)

ReLU — Rectified linear unit — Half-wave rectification



Feature maps (24x24x8)

POOL — Down-sampling by max-pooling in 2x2 blocks

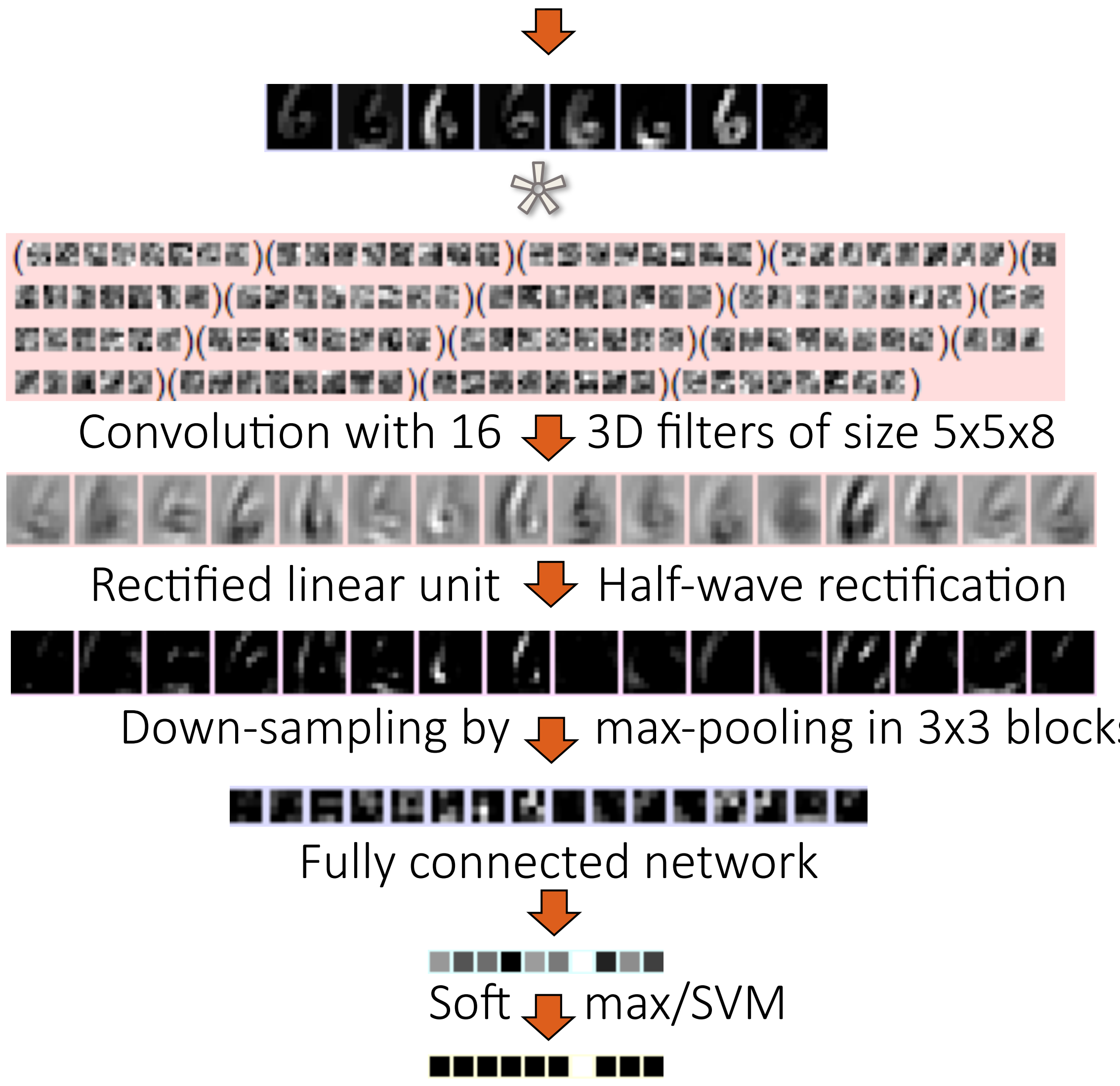


Down-sampled feature maps (12x12x8)

[LeCun et al.]

Processing

Representation



Down-sampled feature maps (12x12x8)

CONV — Convolution with 16 3D filters of size 5x5x8

Convolution results (12x12x16)

ReLU — Rectified linear unit Half-wave rectification

Feature maps (12x12x16)

POOL — Down-sampling by max-pooling in 3x3 blocks

Down-sampled feature maps (4x4x16)

FC — Fully connected network

Graded class scores (10)

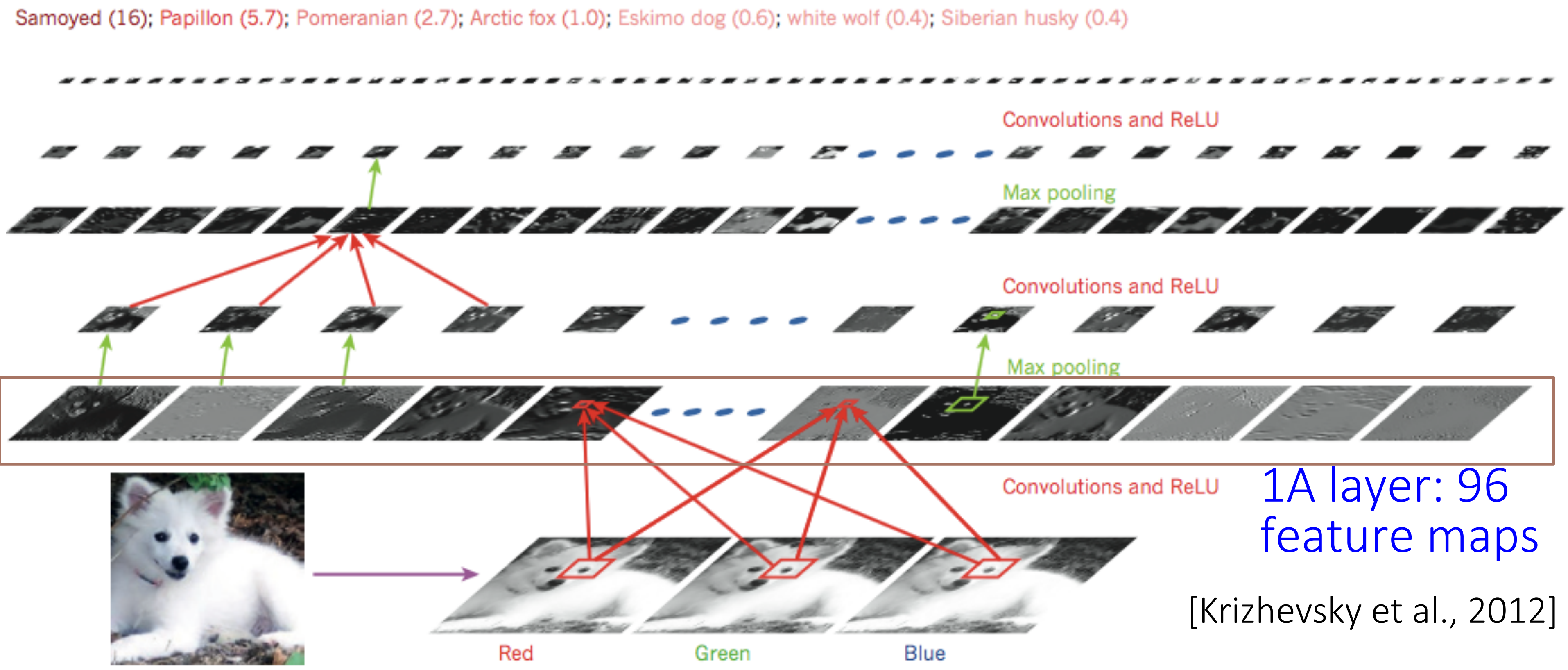
Soft max/SVM

Binary output units (10)

000001000
(Sparse encoding of 6)

[LeCun et al.]

Convolutional Neural Network - ANN with a certain structure


Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic fox (1.0); Eskimo dog (0.6); white wolf (0.4); Siberian husky (0.4)
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Red
Green
Blue

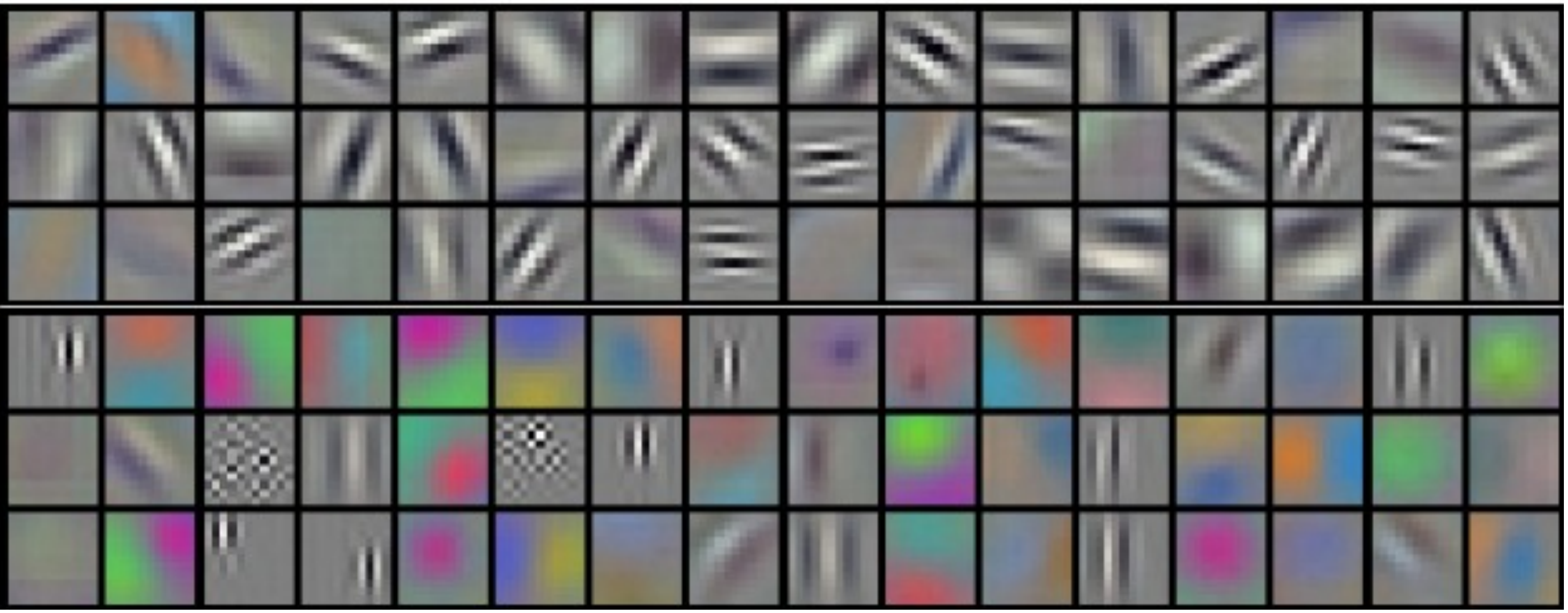
1A layer: 96 feature maps

[Krizhevsky et al., 2012]

1A layer: the units correspond to pixels in multiple feature maps, the pixel/unit values in one map are computed by (e.g. 11x11x3) convolution and ReLU (rectified liner unit); different maps correspond to different convolution kernels with trainable weights

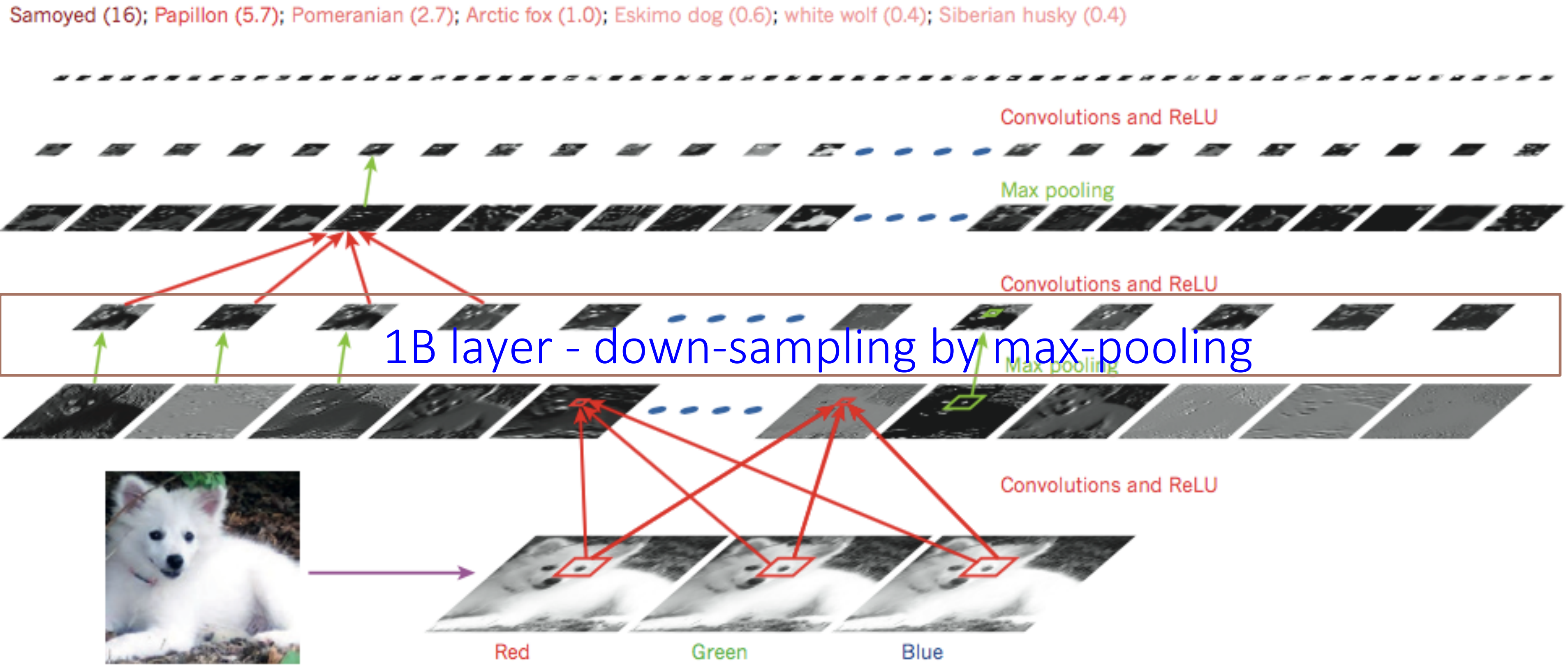
[LeCun et al, 2015]

1A Convolutional layer



96 convolution kernels filters of size 11×11×3 [Krizhevsky et al., 2012]

1B layer: down-sampling of feature maps by 'max-pooling'

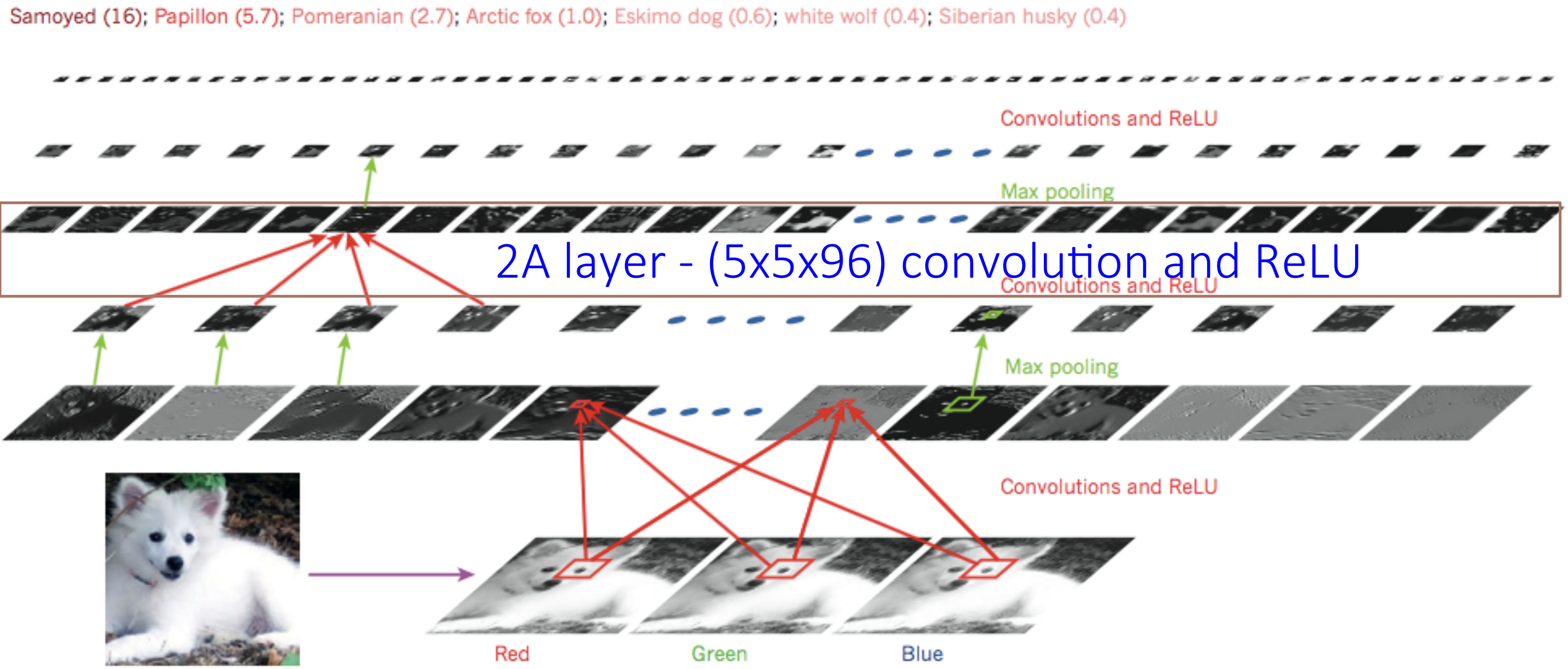


Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic fox (1.0); Eskimo dog (0.6); white wolf (0.4); Siberian husky (0.4)
Convolutions and ReLU
Max pooling
Convolutions and ReLU
1B layer - down-sampling by max-pooling
Max pooling
Convolutions and ReLU
Red
Green
Blue

- Pooling/downsampling (e.g. 4x4) of reduction of the number of intermediate features that was increased by a factor of 96/3=32 in the 1st convolutional layer
- Max for tolerance to feature position (compensates small shifts)

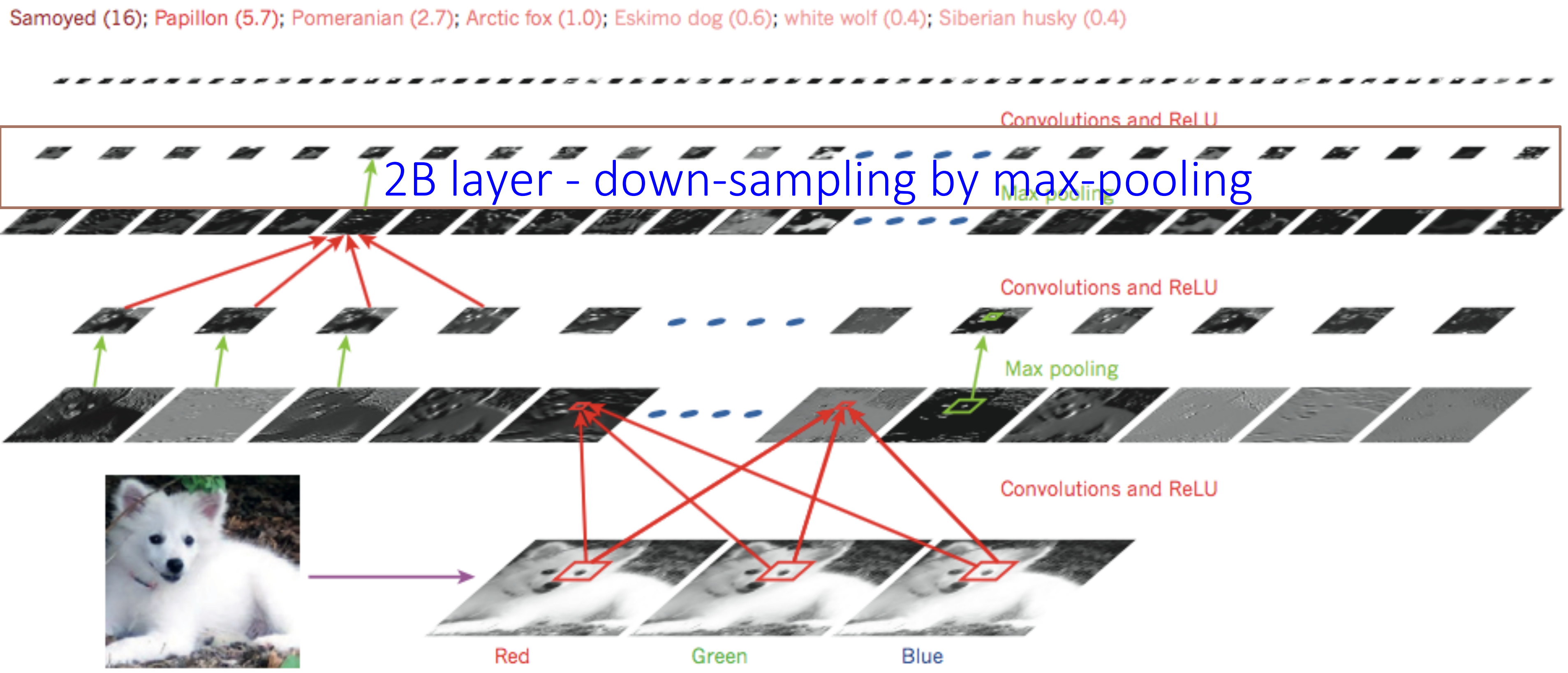
[LeCun et al, 2015]

2A layer: convolution and ReLU


Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic fox (1.0); Eskimo dog (0.6); white wolf (0.4); Siberian husky (0.4)
Convolutions and ReLU
Max pooling
2A layer - (5x5x96) convolution and ReLU
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Red
Green
Blue

[LeCun et al, 2015]

2B layer: down-sampling of feature maps by 'max-pooling'


Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic fox (1.0); Eskimo dog (0.6); white wolf (0.4); Siberian husky (0.4)
Convolutions and ReLU
2B layer - down-sampling by max-pooling
Max pooling
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Red
Green
Blue

[LeCun et al, 2015]

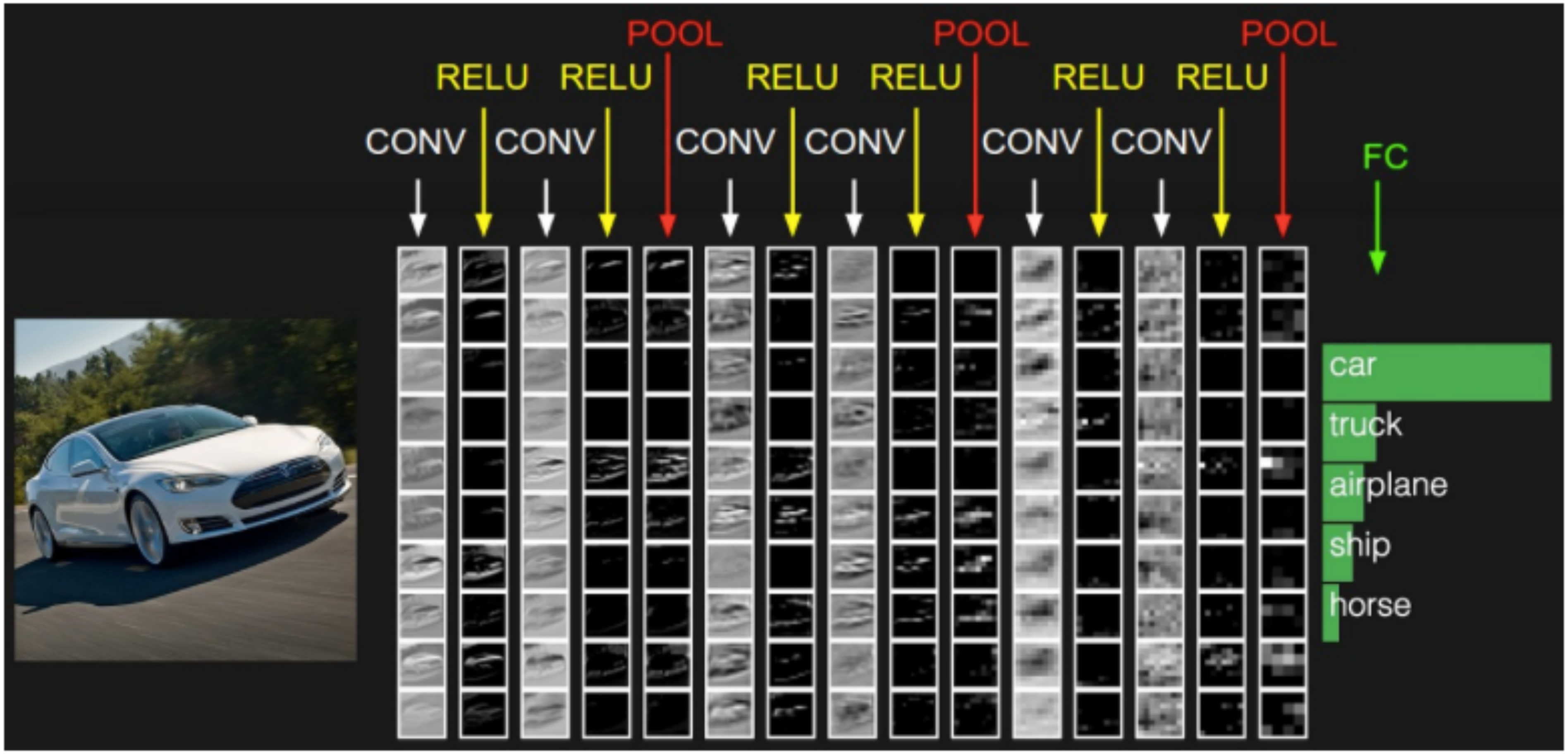
Representations learned by deep learning CNN

Example - image input:

- Input layer: (3 matrices of) pixel color values
- 1st layer: presence/absence of edges of particular orientation at particular position (similar to neurons in area V1/V2)
- 2nd layer: local arrangements of edges (similar to area V4)
- 3rd layer: assemblies of local edge arrangements (similar to posterior TEO) that may correspond parts of objects
- 4th layer: combinations of parts of objects that may represent complete objects (similar to anterior TEO)

Key aspect: these layers of features are not designed by human engineers, they are learned from data using a general-purpose learning procedure!

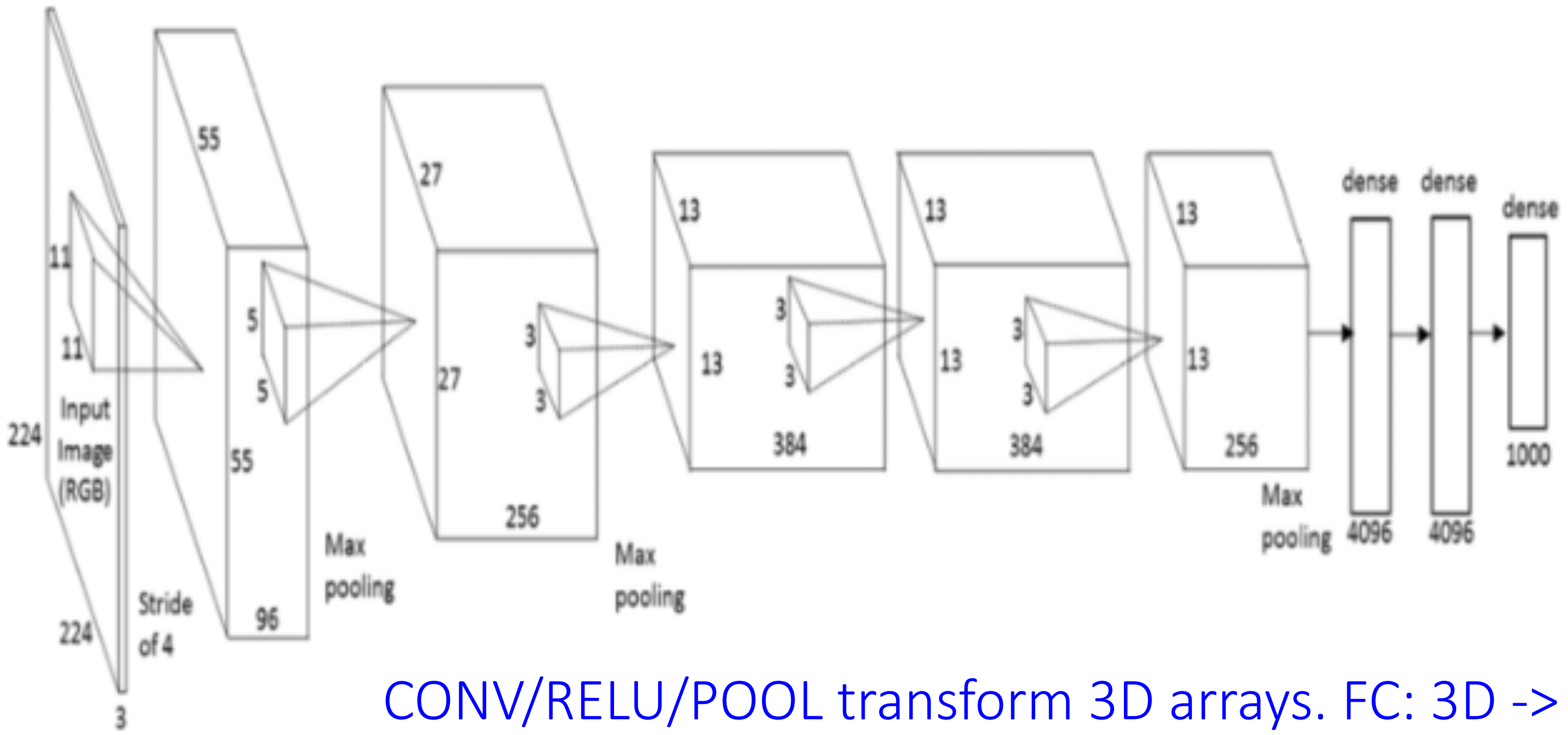
Examples of CNN architectures



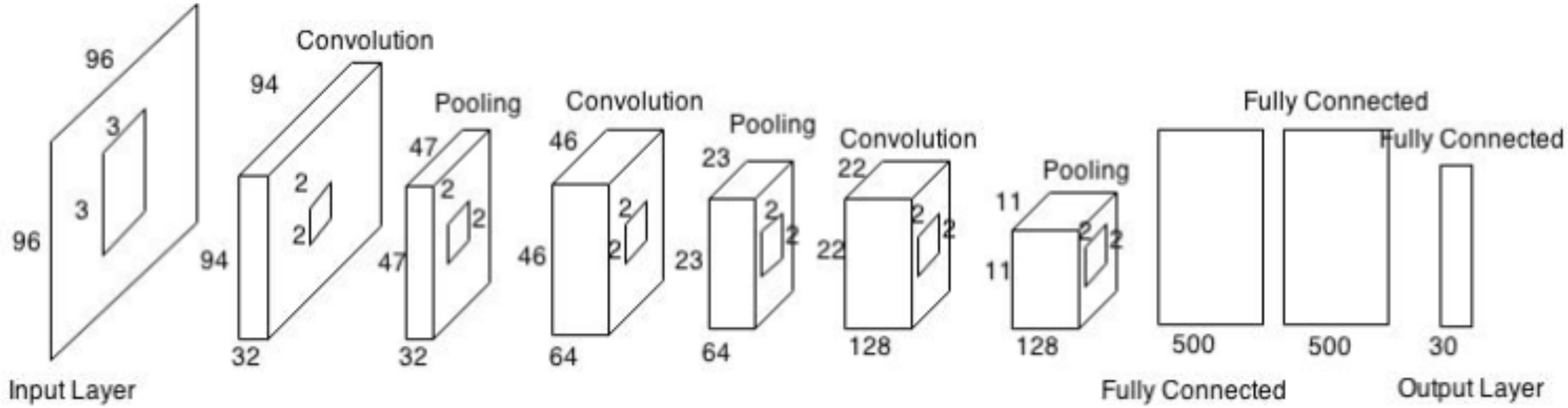
A CNN consists of a sequence of CONV, RELU, POOL and FC layers.

[http://cs231n.github.io/convolutional-networks/
]

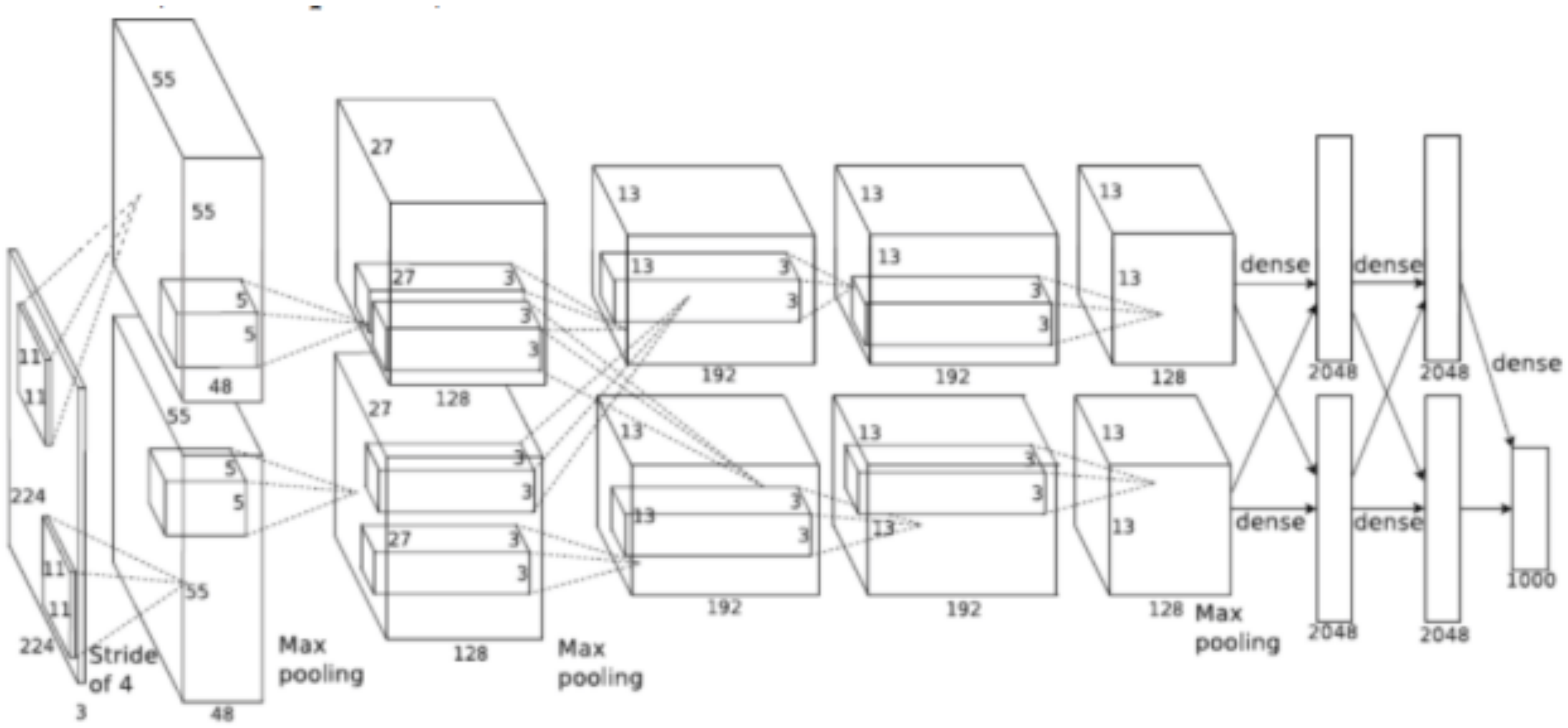
Examples of CNN architectures



CONV/RELU/POOL transform 3D arrays. FC: 3D -> 1D.



Examples of CNN architectures



Recent CNN architectures have 10-20 layers and hundreds of millions of weights

Convolutional Networks architectures that have a name

- LeNet, 1990's, used to read zip codes, digits, etc.

- AlexNet popularized Convolutional Networks in Computer Vision. In ImageNet ILSVRC challenge 2012 significantly outperformed the second runner-up (top 5 error of 16% vs. 26%). Similar to LeNet, but deeper, bigger, and featured convolutional layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).

[http://cs231n.github.io/convolutional-networks/
]

Convolutional Networks architectures that have a name (cont.)

- ZF Net: ILSVRC 2013 winner, an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.
- GoogLeNet, ILSVRC 2014 winner. Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M). Uses average pooling, eliminated a large amount of parameters that do not seem to matter much. Several follow-up versions, most recently Inception-v4.

[http://cs231n.github.io/convolutional-networks/
]

Convolutional Networks architectures that have a name (cont.)

- VGGNet, ILSVRC 2014 winner. Showed that the depth of the network is a critical component for good performance. 16 CONV/FC layers, only performs 3x3 convolutions and 2x2 pooling. Pre-trained model is available for plug and play use in Caffe. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

[http://cs231n.github.io/convolutional-networks/
]

Convolutional Networks architectures that have a name (cont.)

- ResNet. Winner of ILSVRC 2015. It features special skip connections and a heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network. The reader is also referred to Kaiming's presentation (video, slides), and some recent experiments that reproduce these networks in Torch. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 10, 2016).

[http://cs231n.github.io/convolutional-networks/
]

VGGNet in detail

Composed of CONV layers that perform 3x3 convolutions with stride 1 and pad 1, and of POOL layers that perform 2x2 max pooling with stride 2 (and no padding).

[http://cs231n.github.io/convolutional-networks/
]

VGGNet in detail

INPUT: [224x224x3] memory: 224*224*3=150K weights: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K weights: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K weights: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K weights: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K weights: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K weights: 0
FC: [1x1x4096] memory: 4096 weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 weights: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 weights: 4096*1000 = 4,096,000

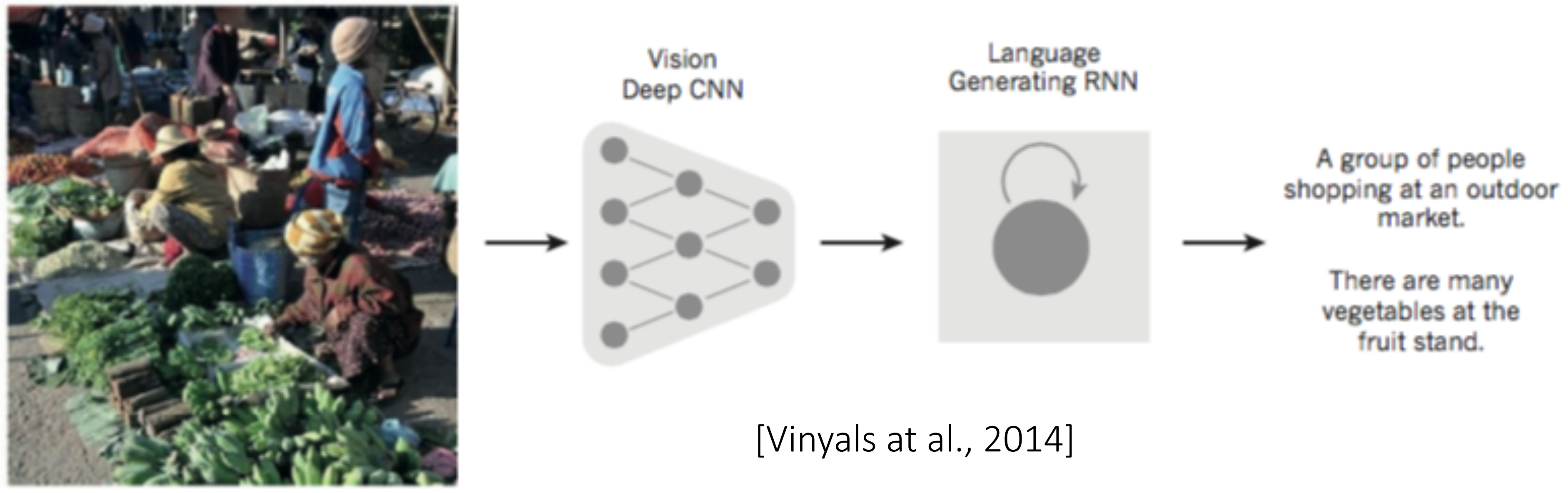
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

[http://cs231n.github.io/convolutional-networks/

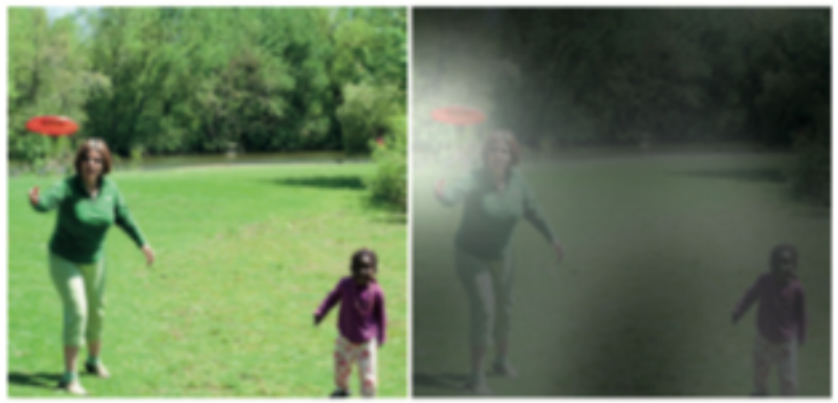
Image understanding with deep CNN

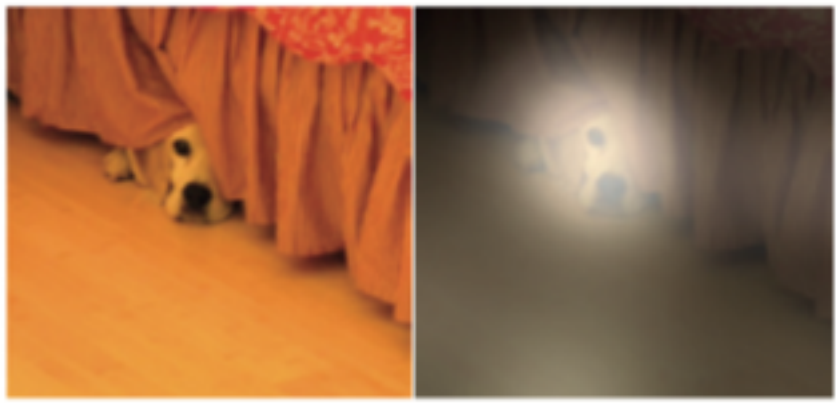
- Since 2005, successes in detection/segmentation/recognition of objects and regions in images in tasks with abundant labeled data (traffic signs, pedestrians, faces)
- Breakthrough in 2012 ImageNet competition (data set of 1 million images, 1000 classes): halving existing error rates
- CNNs are now the dominant approach for visual recognition tasks
- Used by Google, Facebook, Microsoft, IBM, Yahoo!, Twitter, Adobe, and many start-ups
- NVIDIA, Mobileye, Intel, Qualcomm, Samsung are developing CNN chips to enable real-time vision applications in smartphones, cameras, robots and self-driving cars (déjà vu: Siemens ANN chip 1990)
- Challenges: which is the appropriate architecture, huge data needed (techniques to generate more training examples by deforming existing ones)

Image understanding with deep CNN


Vision
Deep CNN
Language
Generating RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

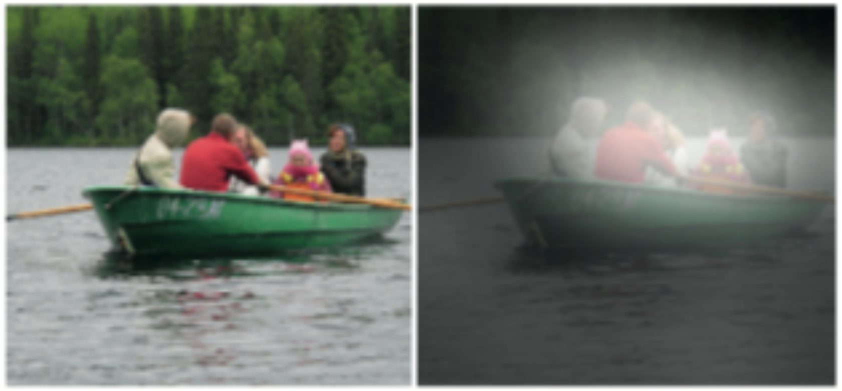
[Vinyals at al., 2014]

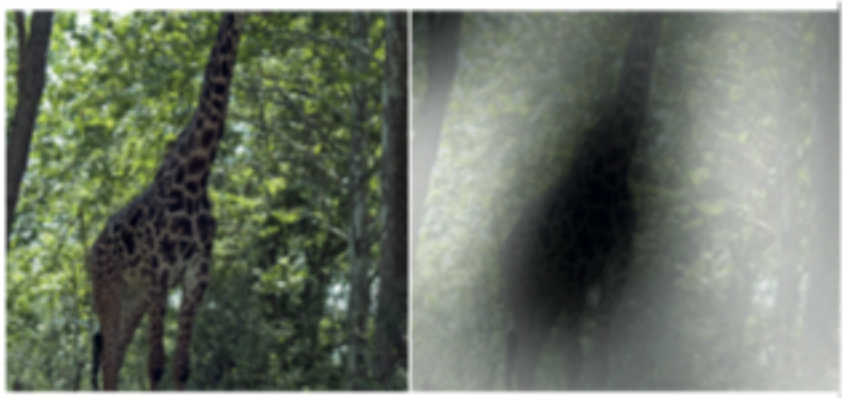

A woman is throwing a **frisbee** in a park.

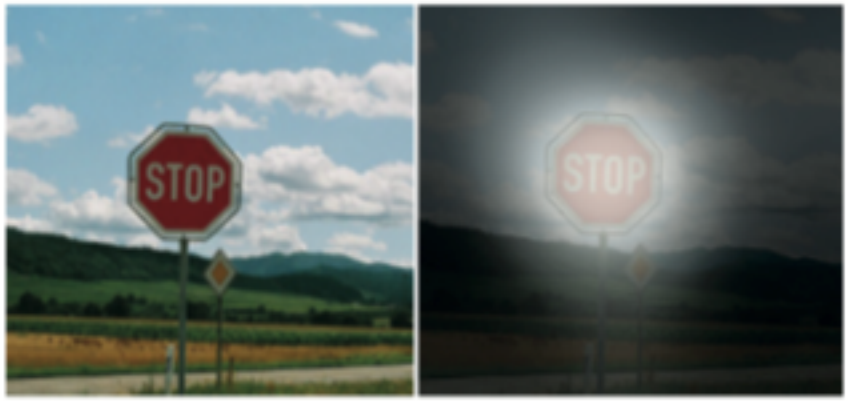

A **dog** is standing on a hardwood floor.

Captions generated by a CNN for object recognition and an RNN for text generation.

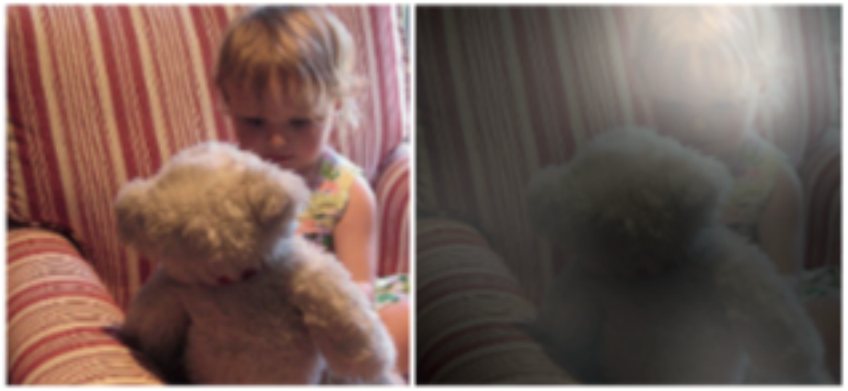
Image understanding with deep CNN


A group of **people** sitting on a boat in the water.


A giraffe standing in a forest with **trees** in the background.



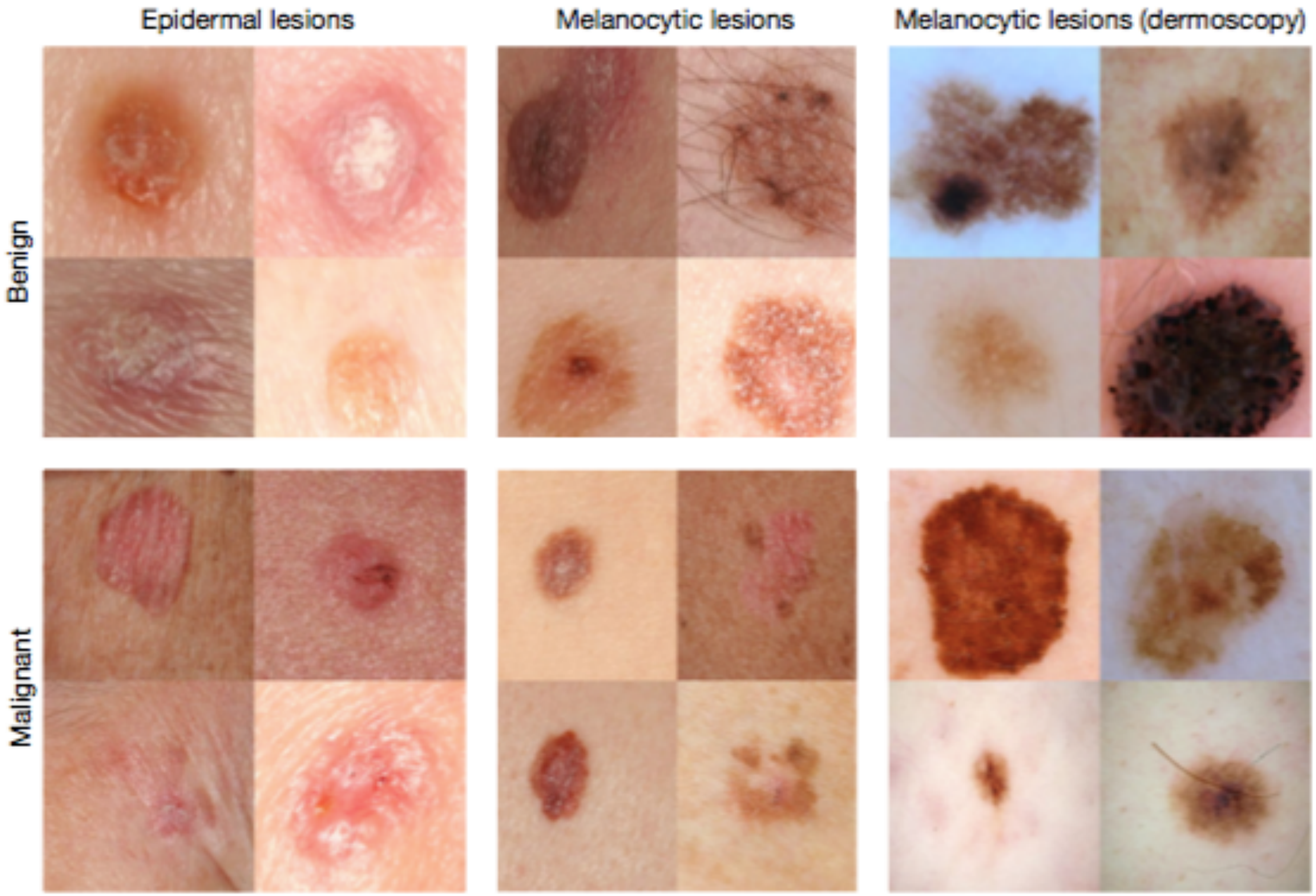
A **stop** sign is on a road with a mountain in the background


A little **girl** sitting on a bed with a teddy bear.

[Vinyals at al., 2014]

Captions generated by CNN for object recognition and RNN for text generation.

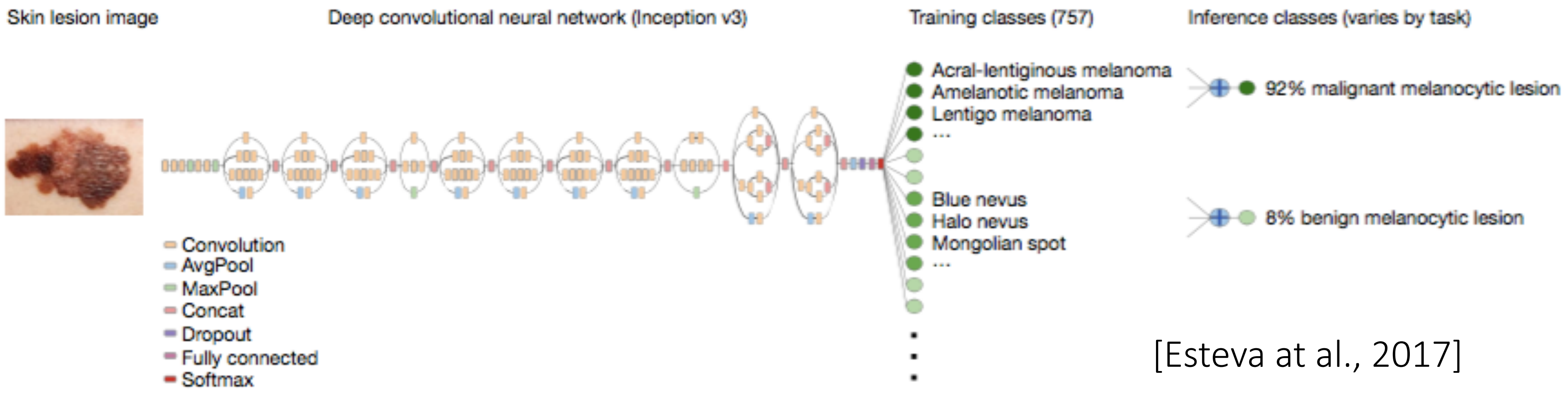
Application to skin cancer classification



Examples of images of benign and malignant lesions that are difficult to distinguish visually

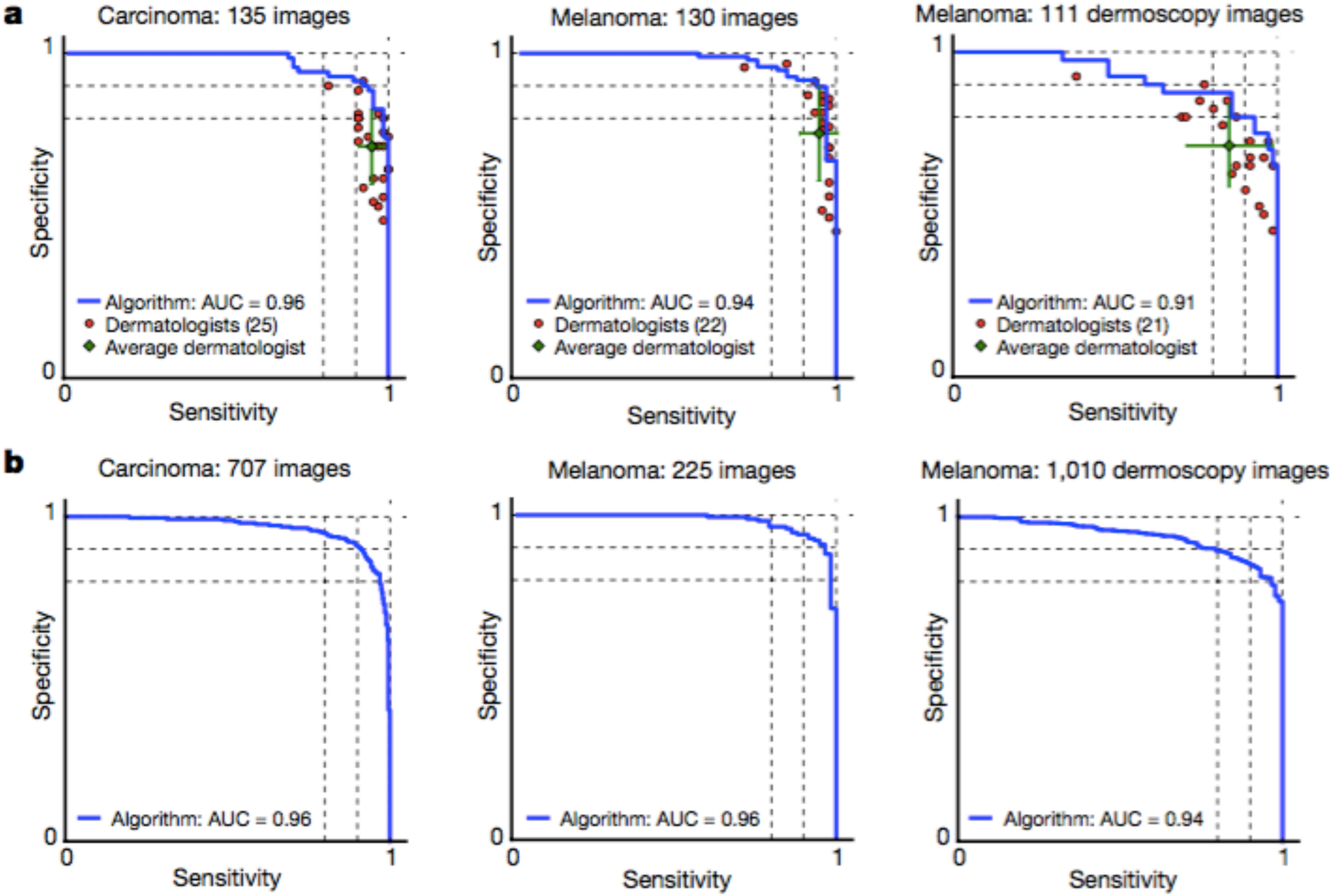
[Esteva at al., 2017]

Application to skin cancer classification

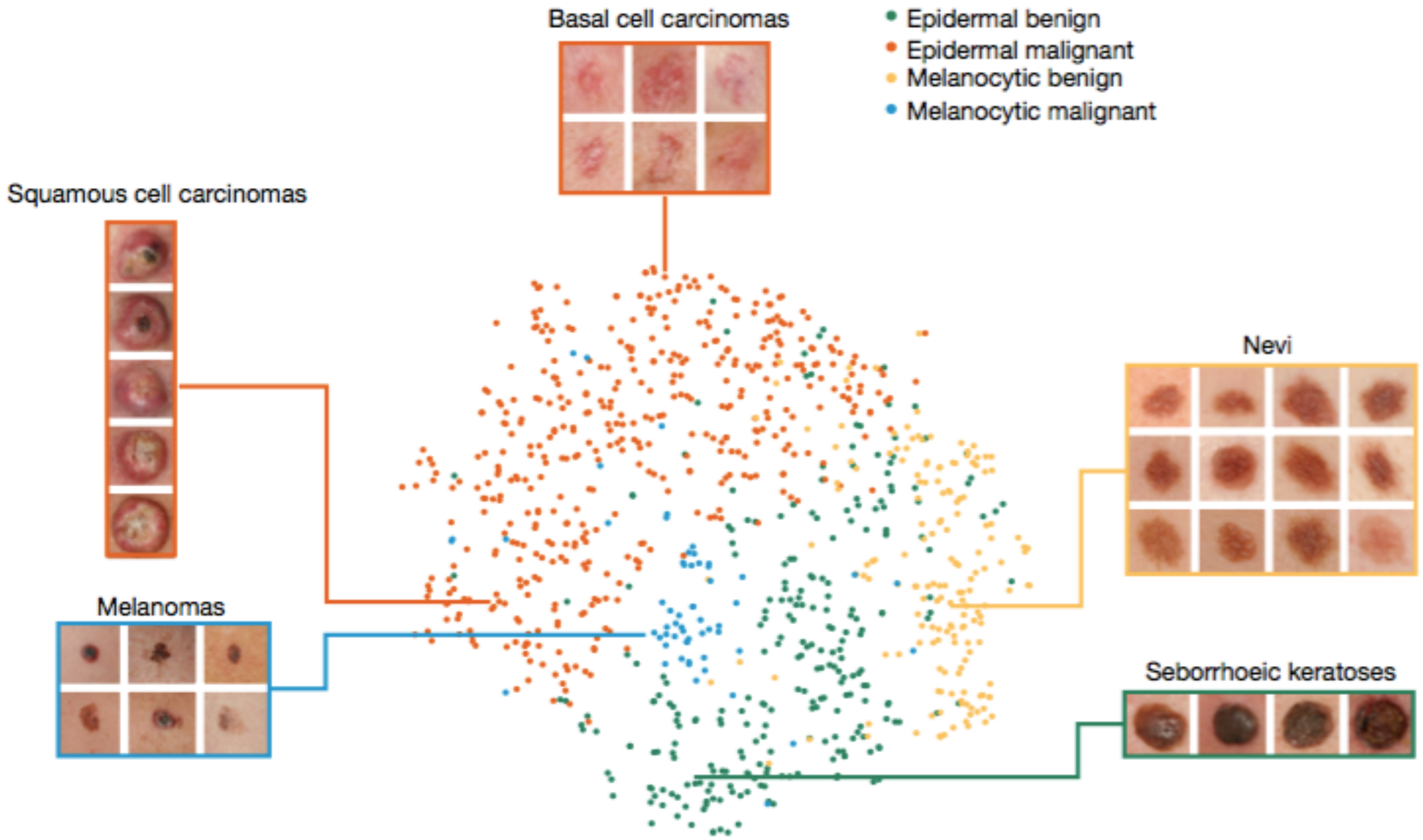


[Esteva at al., 2017]

- Google Inception v3 CNN architecture
- pre-trained on the ImageNet dataset (1.28 million images over 1000 generic object classes) and
- fine-tuned on a dataset of 129450 skin lesions comprising 2032 different diseases grouped in 757 classes.
- 127463 training and validation images and 1942 biopsy-labeled test images



a) Comparison of a CNN with dermatologists.
b) Results of CNN for bigger test sets to validate (a). [Esteva at al., 2017]



(t-SNE) visualization by projecting the 2048-dimensional output of the CNN's last hidden layer into two dimensions

[Esteva at al., 2017]

PS: Application to skin cancer classification

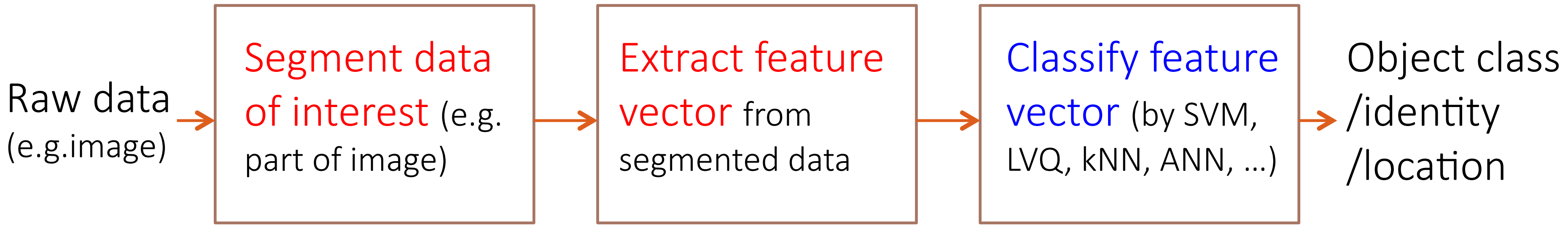
Dear Nicolai,

Just to inform you that A… and I stepped off this idea since the database needs thousands of images, and the network machine learning has nothing to do with dermatological algorithms.

9-3-2017 …… (professor of dermatology)

Which methodological problem does deep learning solve?

Conventional pattern recognition framework


Raw data (e.g.image)
Segment data of interest (e.g. part of image)
Extract feature vector from segmented data
Classify feature vector (by SVM, LVQ, kNN, ANN, …)
Object class /identity /location

Requires (some) recognition.

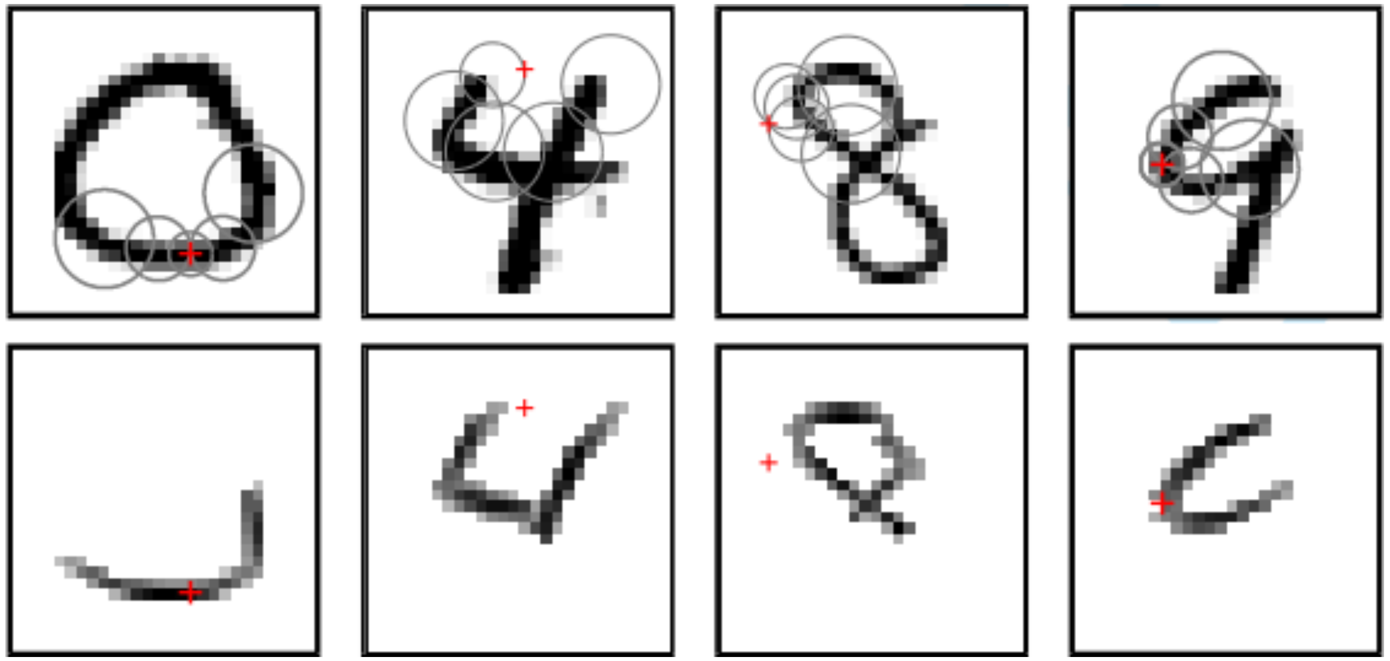
Requires careful engineering and considerable domain expertise.

This at least works fine

Limitations to process natural data in their raw form

Representation learning

A set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification.



Example: trainable COSIFRE filters for handwriting OCR.

Deep learning distinctive aspects

- Multiple layers ANN (5-20), after 2005. Intermediate representations of data, the representations in one layer are used to compute the representations in the following layer.
- Includes representation learning. Automatic design of feature extractors as part of the parameter adjustment. No separation of feature extraction and classification.
- Input is raw data (e.g. image, audio)
- Led to progress in speech/audio and image/video recognition
- Claimed other applications: activity of potential drug molecules, analysing particle accelerator data, reconstructing brain circuits, and predicting the effects of mutations in non-coding DNA on gene expression and disease, natural language understanding, particularly topic classification, sentiment analysis, question answering and language translation
- Huge data sets needed for learning.
- Which is the best architecture for a given problem?

References and further reading

- Y. LeCun, Y. Bengio, G. Hinton: Deep learning, Nature, Vol. 521, 28 May 2015, 436-444, doi:10.1038/nature14539
- A. Esteva, B. Kuprel, R.A. Novoa, J. Ko, S.M. Swetter, H.M. Blau, S. Thrun: Dermatologist-level classification of skin cancer with deep neural networks, Nature, Vol. 542, 2 Feb. 2017, 115-118, doi:10.1038/nature21056
- http://cs231n.github.io/convolutional-networks/