# Convolutional Neural Networks (Ch 14)

Wojtek Kowalczyk

wojtek@liacs.nl

# Why do we need many layers?

- In theory, one hidden layer is sufficient to model any function with arbitrary accuracy; however, the number of required nodes and weights grows exponentially fast
- The deeper the network the less nodes are required to model "complicated" functions
- Consecutive layers "learn" features of the training patterns; from simplest (lower layers) to more complicated (top layers)
- Visual cortex consists of about 10 layers

# Why can't we just add more layers?

- In practice, "classical" multi-layer networks work fine only for a very small number of hidden layers (typically 1 or 2) - this is an empirical fact ...

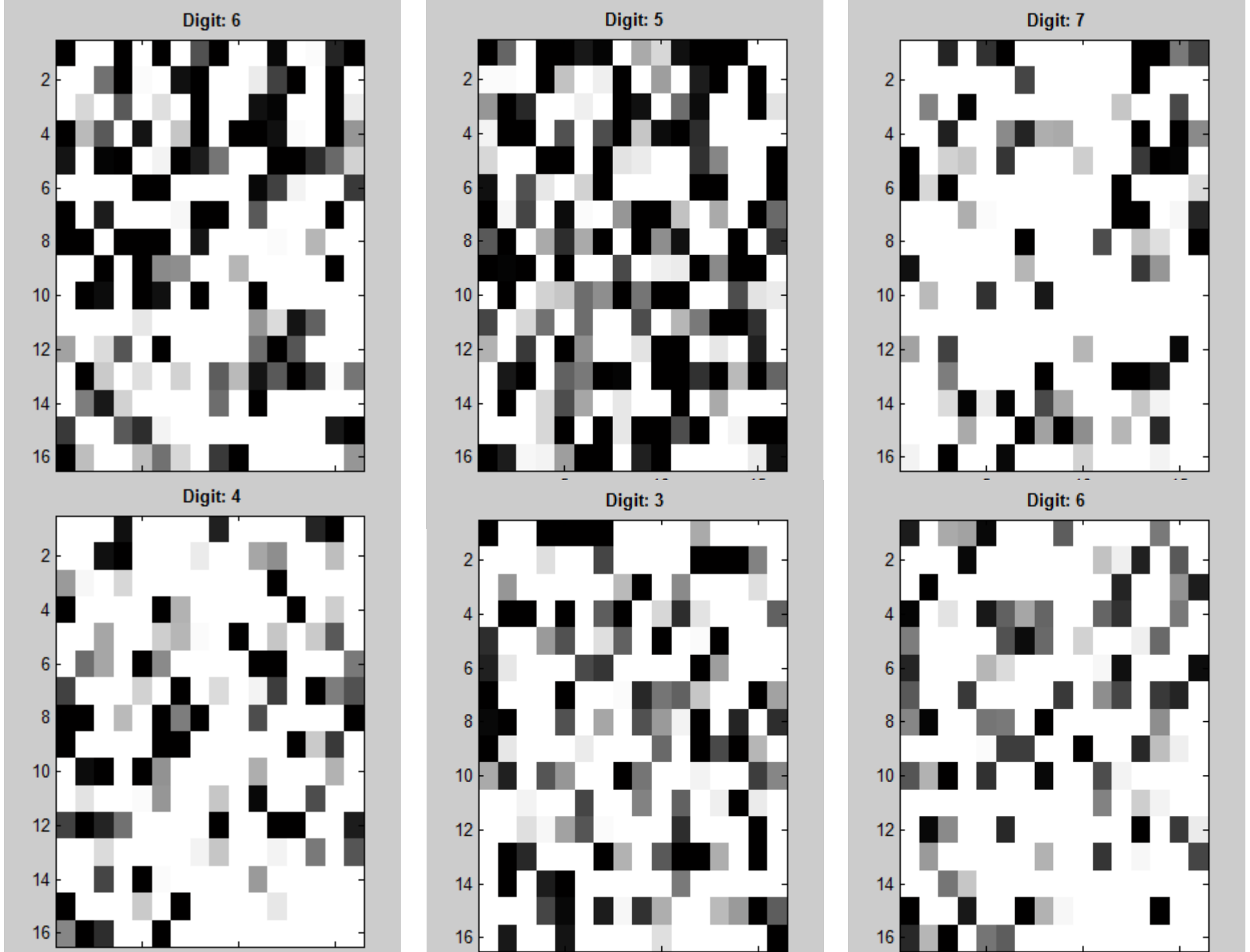
- Adding layers is harmful because:
  - the increased number of weights quickly leads to data overfitting (lack of generalization)
  - huge number of (bad) local minima trap the gradient descent algorithm
  - vanishing or exploding gradients (the update rule involves products of many numbers) cause additional problems

# A disturbing observation

- Consider the *digit recognition problem* (16x16)
- Let us modify the images by *randomly permuting all pixels*: take a random permutation *p* and change every image x[0:16x16] into x[p(0:16x16)]
- What accuracy can be achieved by a single layer perceptron on such a “randomly permuted” data?

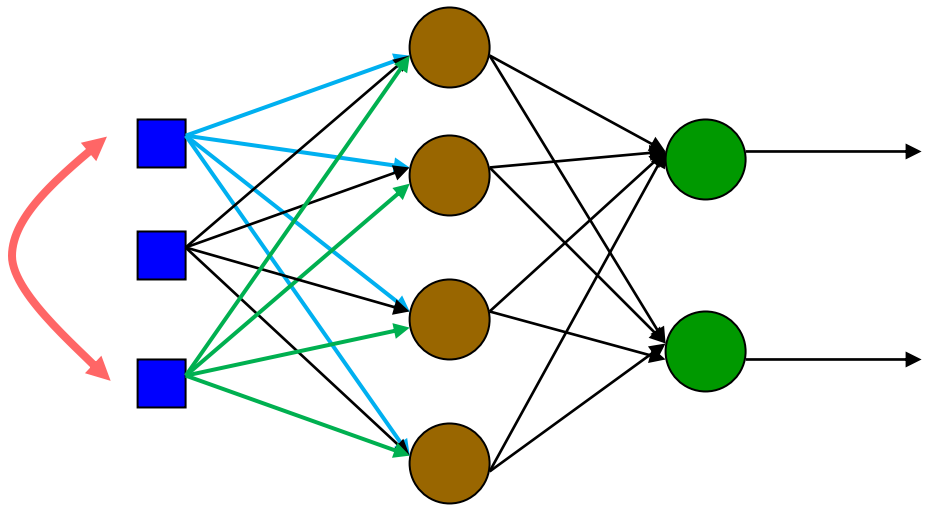
**THE SAME AS ON THE ORIGINAL DATA!**

*(the same holds for a multi-layer perceptron)*


Digit: 6
Digit: 5
Digit: 7
Digit: 4
Digit: 3
Digit: 6
2
4
6
8
10
12
14
16

# A disturbing observation: the proof

- Permute weights from inputs to the hidden nodes in the same way as we permuted input pixels:



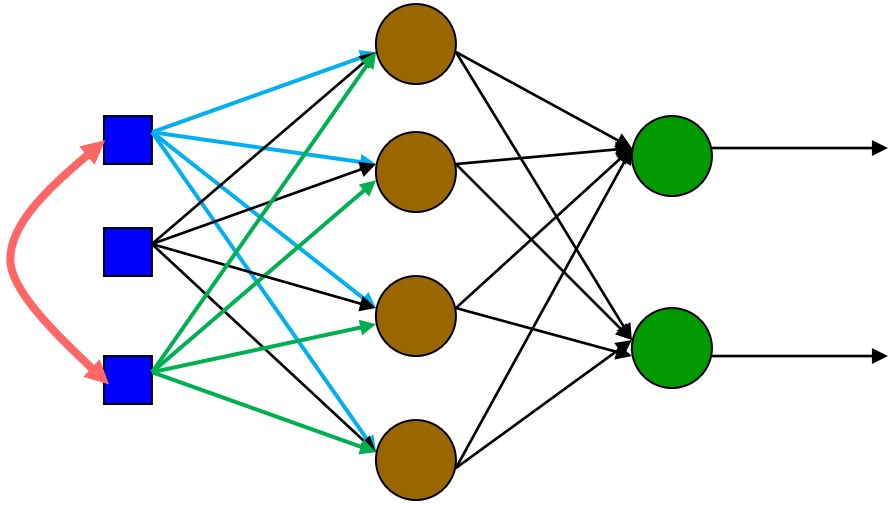
- The “permuted” network behaves in the same way as the original trained network!

# How many local minima?

- Permuting ***n*** hidden nodes (with the corresponding weights) of a given hidden layer doesn't change network behavior
  => ***factorial(n)*** local minima of the error function!

- When a network has $h_1$, $h_2$, .., $h_k$ hidden layers then is has at least $h_1!\ h_2!...h_k!$ local minima (A HUGE NUMBER!)

- Examples:
  - 10!=3.628.800
  - 20!= $2.4*10^{18}$
  - 64!= $1.3*10^{89}$



# How do we “understand” images?

Basic elements: strokes, curves, spots, corners, textures, colors, …

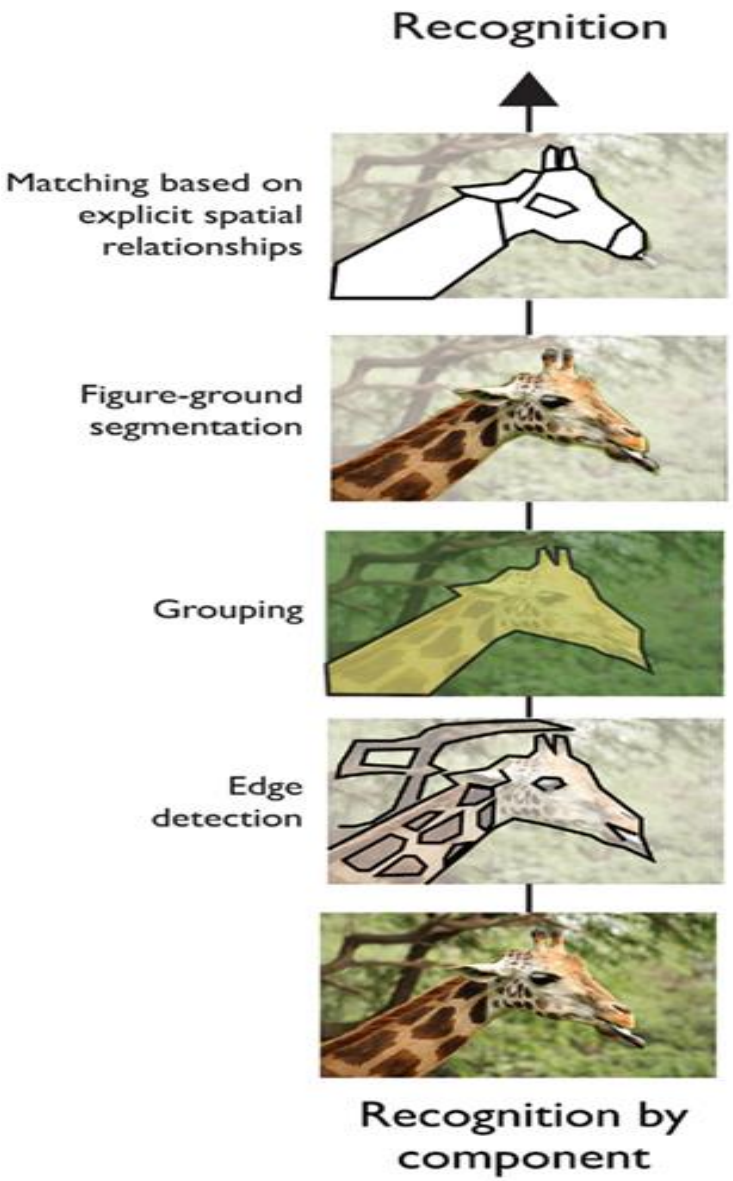
Geometrical relations: distances, angles, up-down, left-right…)

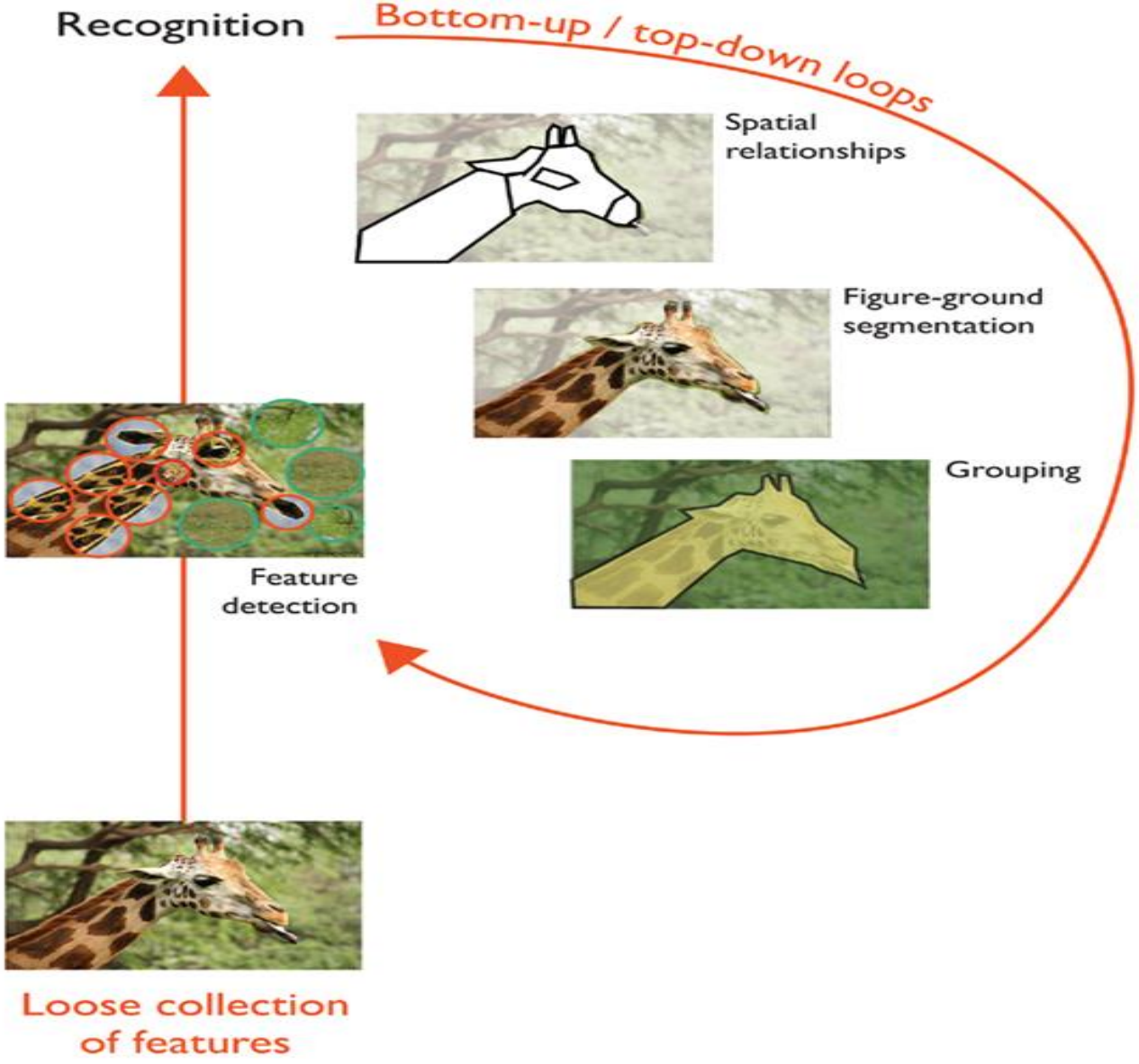
Background

….

=> Visual features

# Visual Features



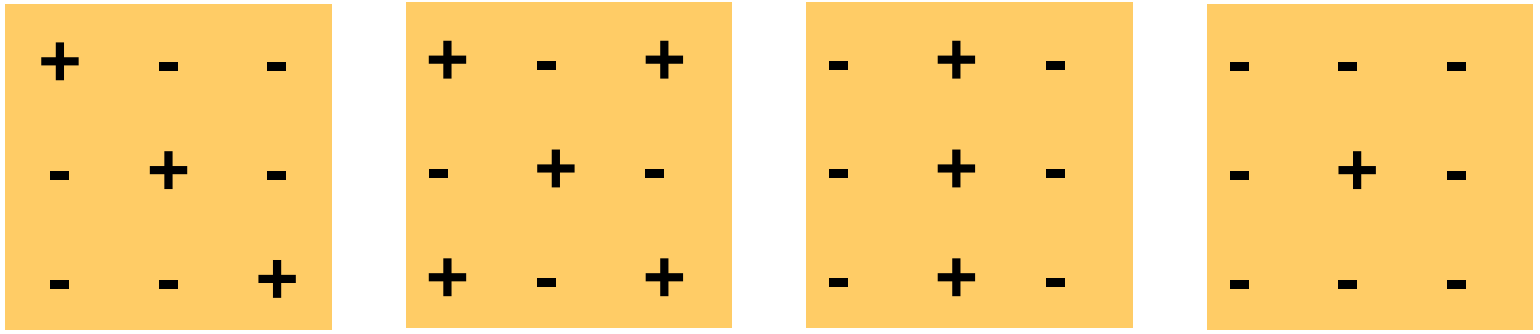



# Convolutional filters

A filter: a “feature detector” – returns high values when the corresponding patch is similar to the filter matrix

Think about all pixels being **-1** (white) or **+1** (black) and filter parameters also restricted to **-1** and **1**

Example: what features are “detected” by:



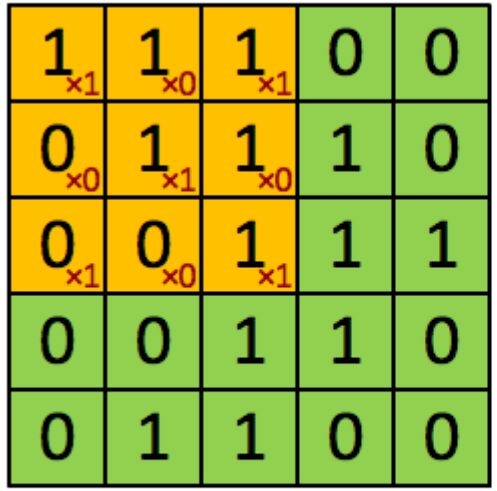
# An example

Let us suppose that in an input image we want to find locations that look like a 3x3 cross. Define a matrix **F** (called a **kernel, receptive field** or **convolutional filter**) and "convolve it" with all possible locations in the image. We will get another (smaller) matrix with "degrees of overlap":

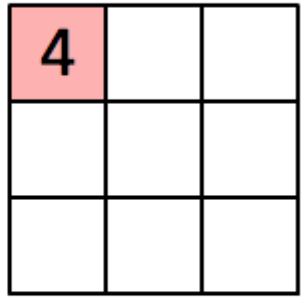
F=

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

*"multiply and add"*




Image




Convolved Feature

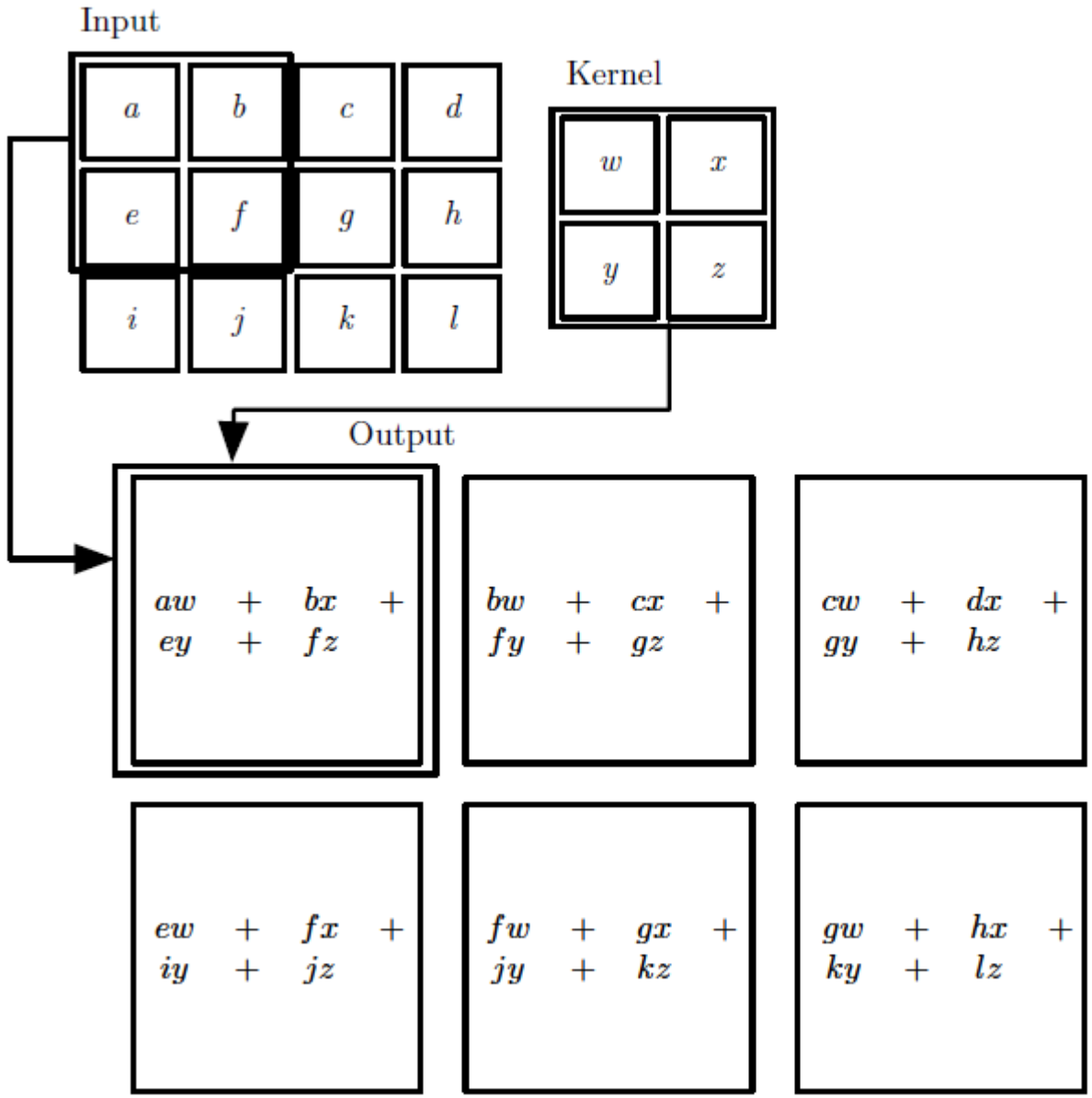
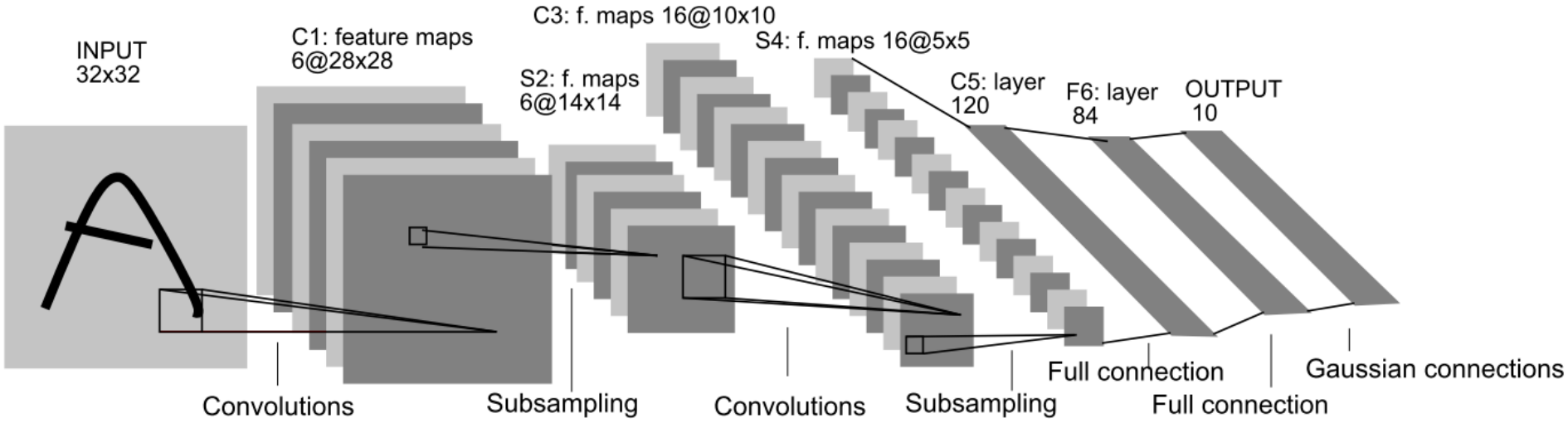



Figure 9.1 (the “Deep Learning” textbook)

# LeNet5 (1989... !?)




- Input: 32x32 pixel image
- Cx: Convolutional layer
- Sx: Subsample layer
  (reduces image size by averaging 2x2 patches)
- Fx: Fully connected layer

http://yann.lecun.com/exdb/lenet/

# The key idea

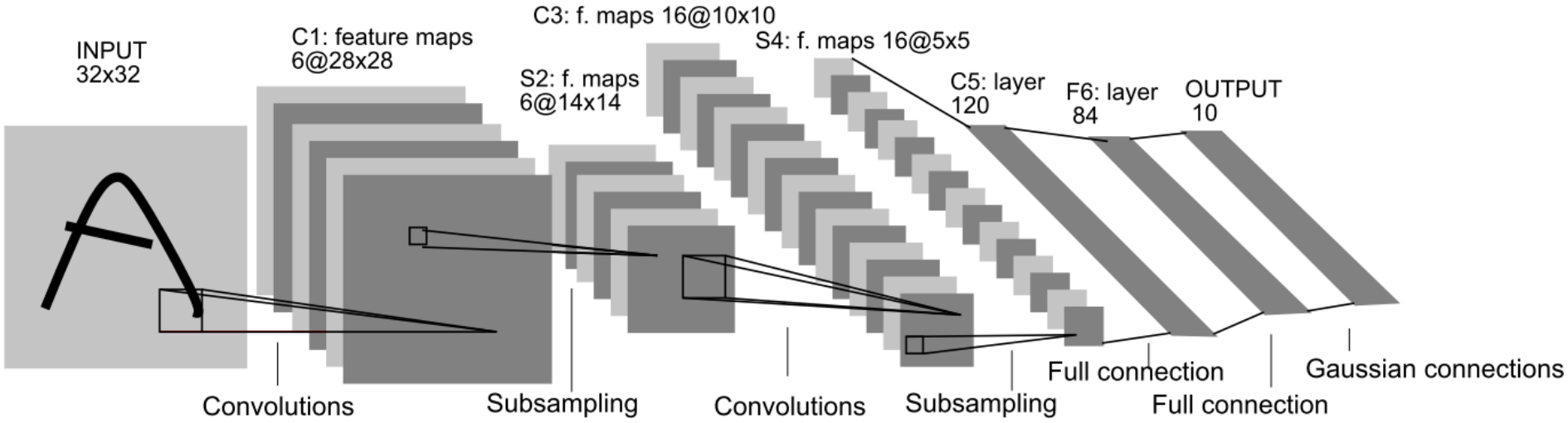
A filter: a “feature detector” – returns high values when the corresponding patch is similar to the filter matrix

How do we know which filters do we need to recognize digits (people, birds, …)?

*Instead of “hand-crafting” relevant filters specify each filter with help of (very few) parameters and find values of these parameters by backpropagation!*

*Why “very few”? A 3x3 filter requires just 9 (or 10? bias) parameters, no matter the size of the input images!*

# LeNet5 (1989... !?)




- Input: 32x32 pixel image
- Cx: Convolutional layer
- Sx: Subsample layer
  (reduces image size by averaging 2x2 patches)
- Fx: Fully connected layer

http://yann.lecun.com/exdb/lenet/

# MNIST data set

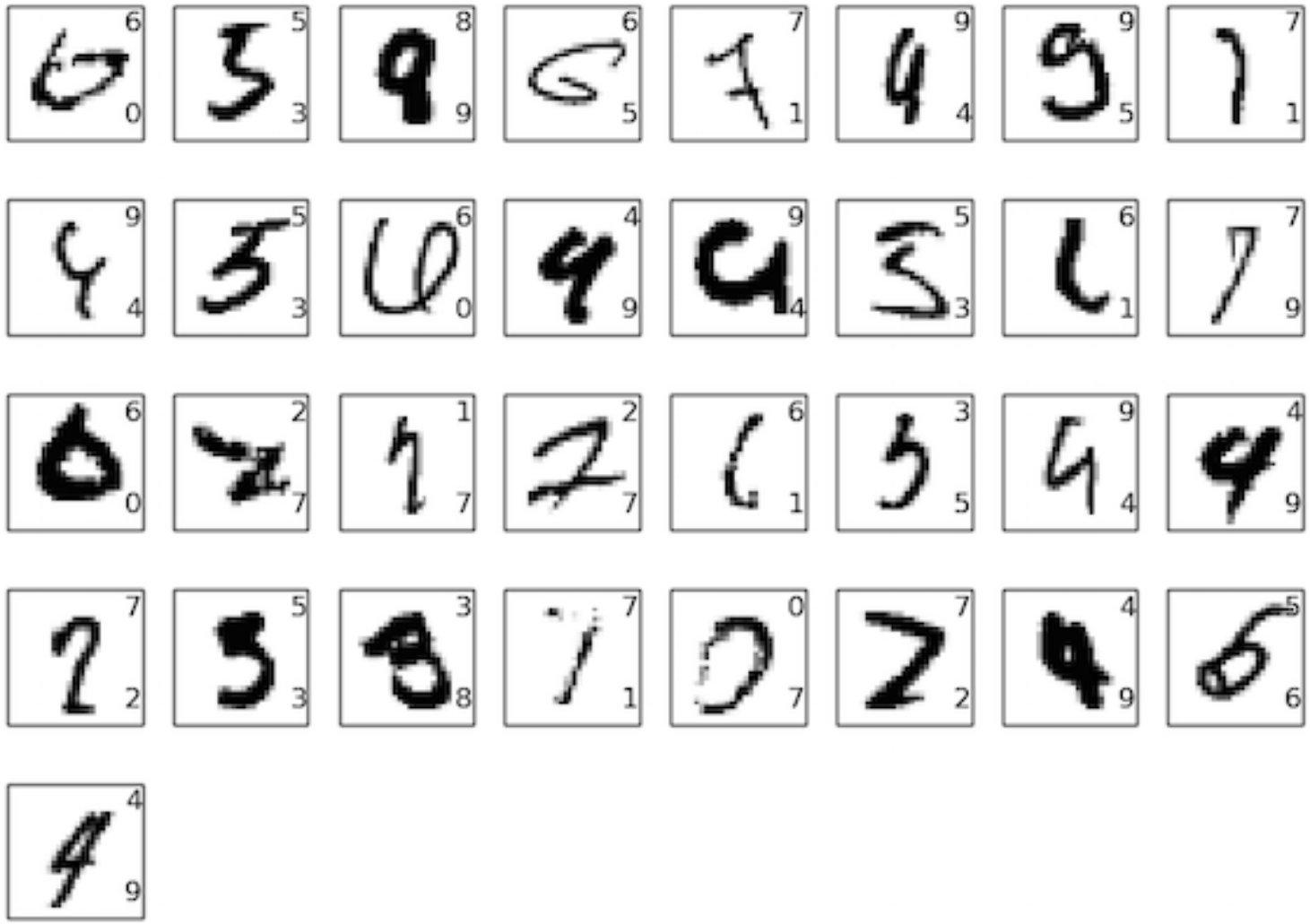
training:
60.000 images

testing:
10.000 images

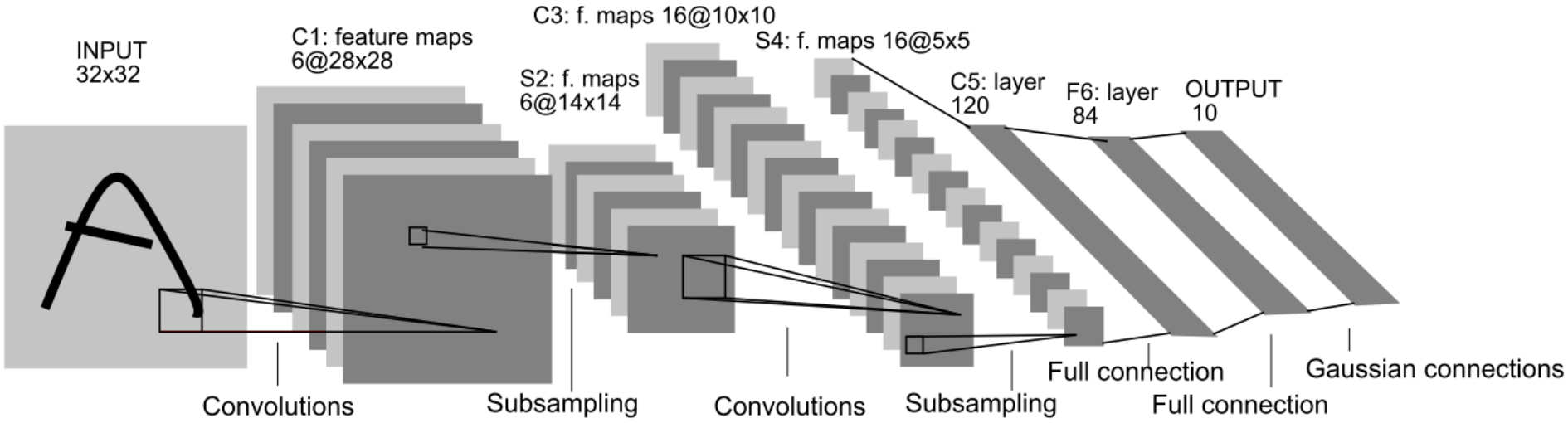
each image:
32x32 pixels

accuracy: 99.7%
*(on the test set)*

# All 33 misclassified digits

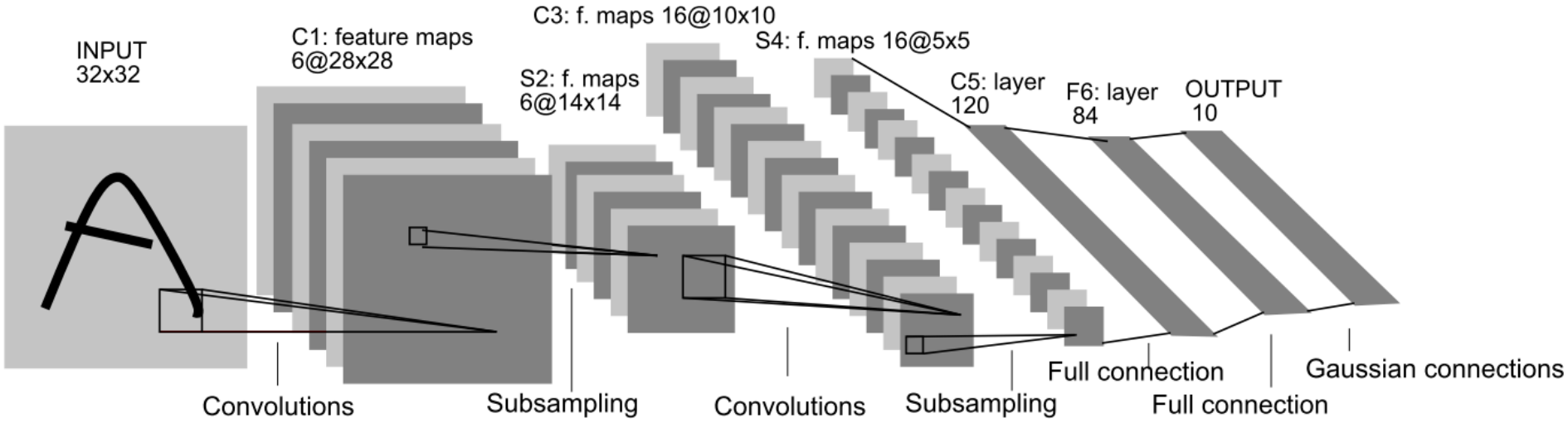


# LeNet5: A MLP!



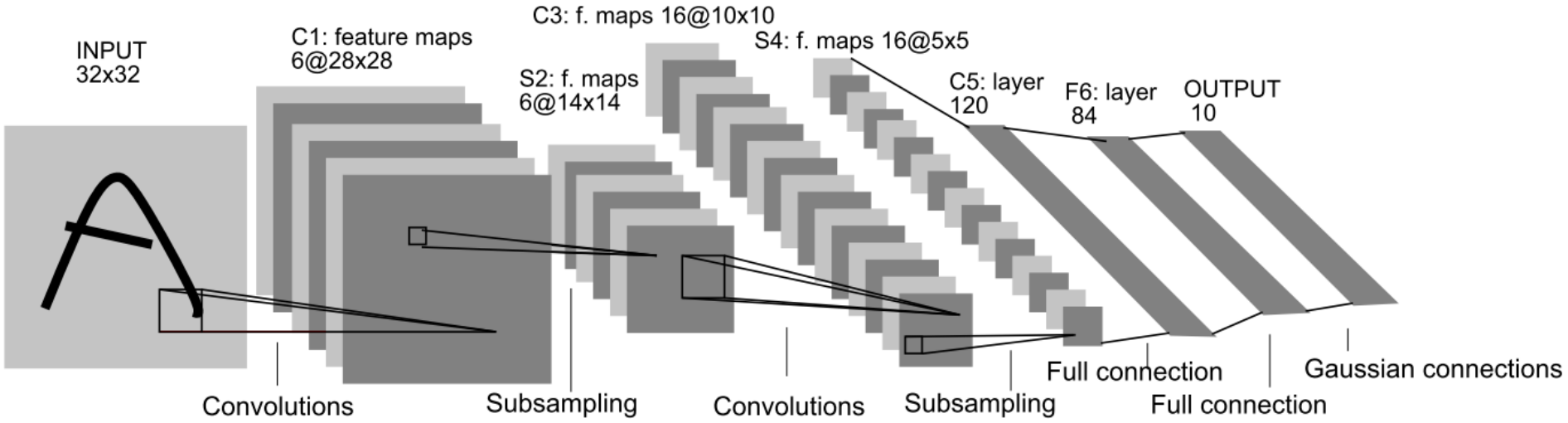

- Input: 32x32 pixel image
- Cx: Convolutional layer
- Sx: Subsample layer
  (reduces image size by averaging 2x2 patches)
- Fx: Fully connected layer

# LeNet 5: Layer C1



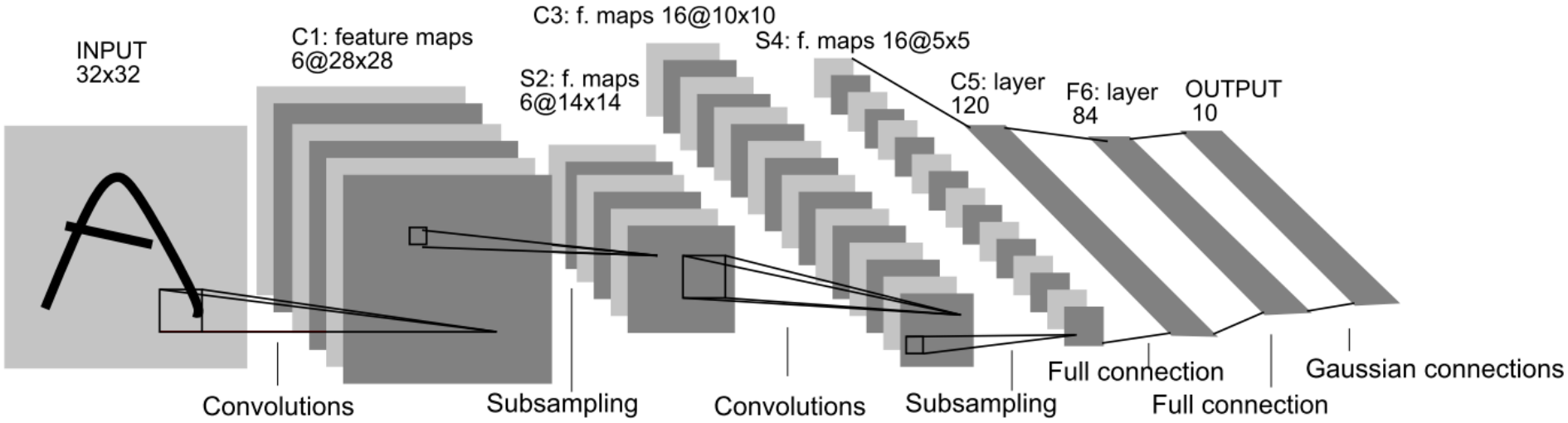

- C1: Convolutional layer with 6 feature maps of size 28x28.
- Each unit of C1 has a 5x5 receptive field in the input layer.
- Shared weights (5*5+1)*6=**156 parameters to learn**
  Connections: 28*28*(5*5+1)*6=122304
- If it was fully connected we had:
  (32*32+1)*(28*28)*6 = **4.821.600 parameters**

# LeNet 5: Layer S2




- S2: Subsampling layer with 6 feature maps of size 14x14 2x2 nonoverlapping receptive fields in C1
- Layer S2: 6*2=**12 trainable parameters**.
- Connections: 14*14*(2*2+1)*6=5880

# ... and so on ...

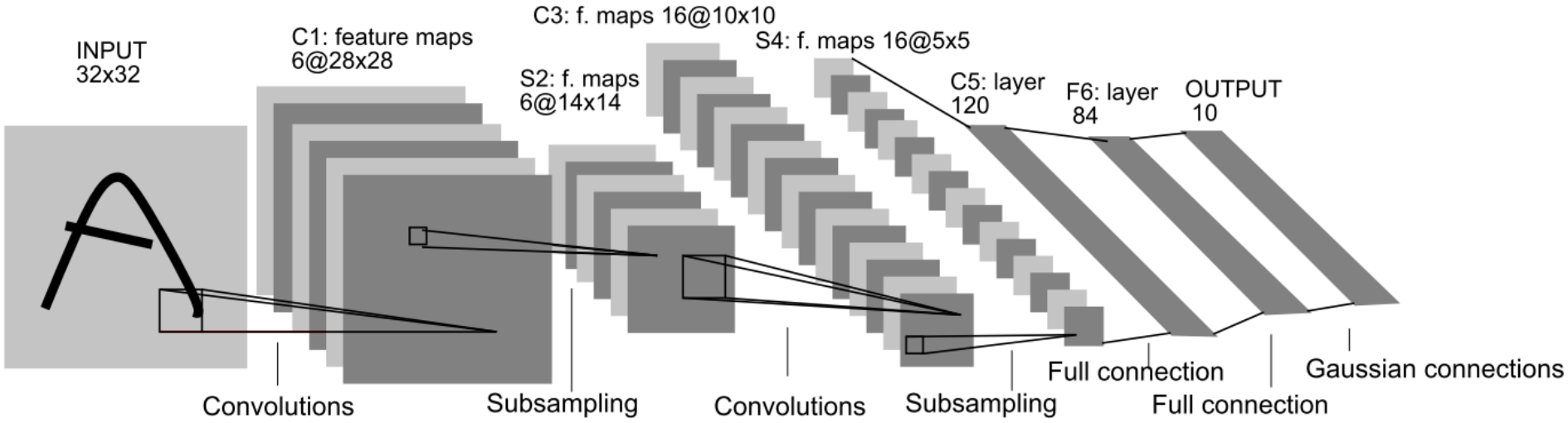



Study slides 11-27 of DeepLearning.pdf

Check that C3 has 1516 trainable parameters (slide 17)

Read the original paper lecun-01a.pdf

# LeNet 5: totals




- The whole network has:
  - 1256 nodes
  - 64.660 connections
  - **9.760 trainable parameters (and not millions!)**
  - **trained with the Backpropagation algorithm!**

# Misclassified cases

# From LeNet5 to ImageNet (2010/2012)

## ImageNet

- 15M images
- 22K categories
- Images collected from Web
- RGB Images
- Variable-resolution
- Human labelers (Amazon's Mechanical Turk crowd-sourcing)

## ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)

- 1K categories
- 1.2M training images (~1000 per category)
- 50,000 validation images
- 150,000 testing images

# ImageNet (study slides 28-40)

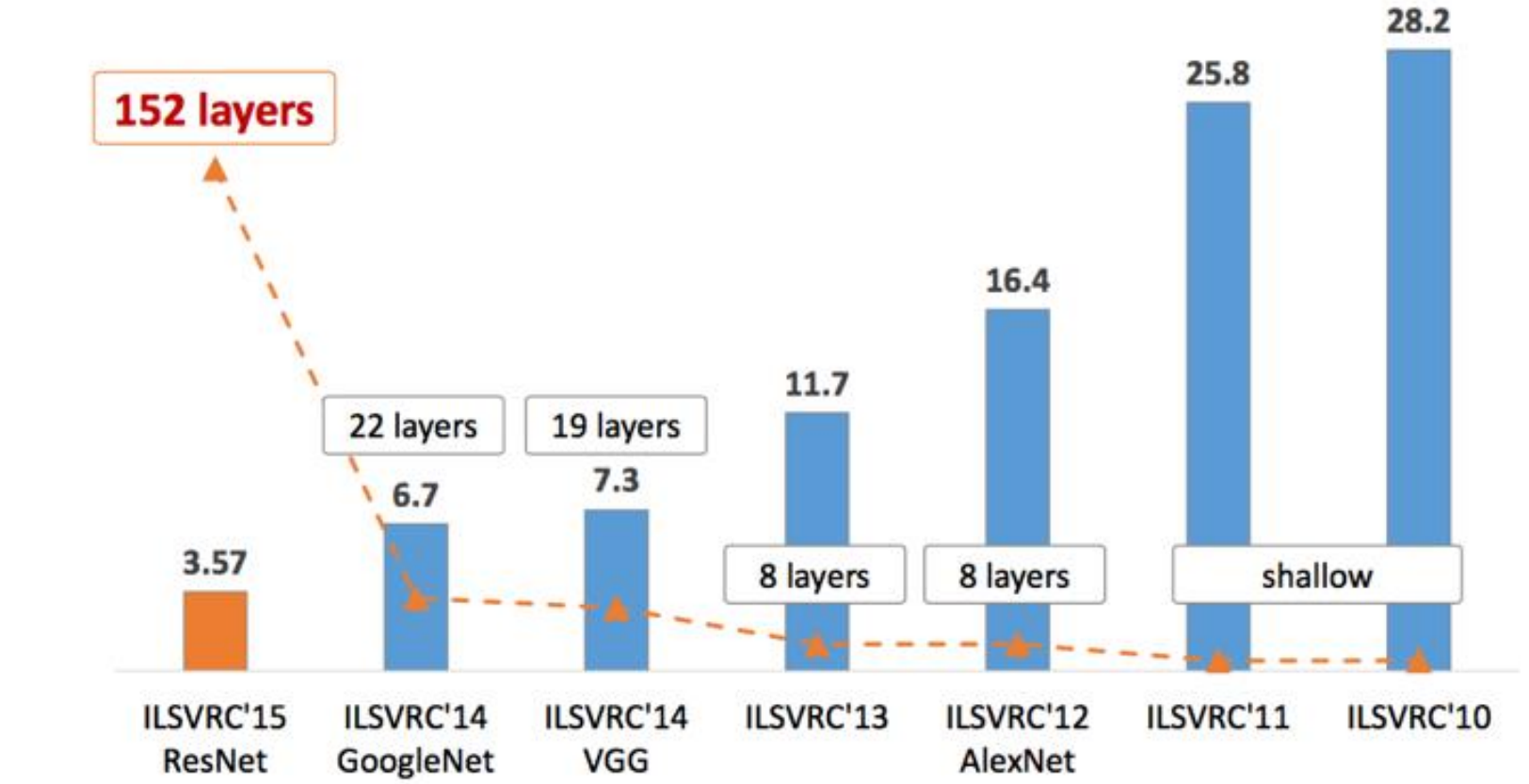
- ILSVRC-2010 test set

| Model | Top-1 | Top-5 |
|---|---|---|
| *Sparse coding [2]* | *47.1%* | *28.2%* |
| *SIFT + FVs [24]* | *45.7%* | *25.7%* |
| CNN | **37.5%** | **17.0%** |

- ILSVRC-2012 test set

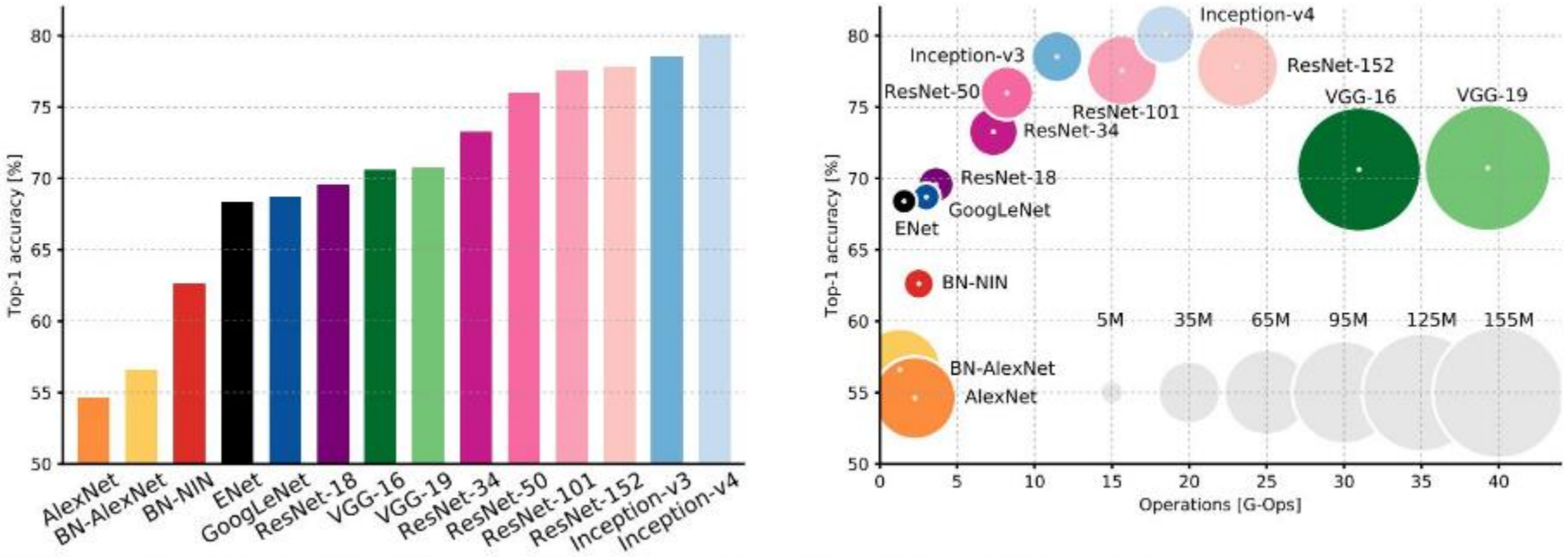
| Model | Top-1 (val) | Top-5 (val) | Top-5 (test) |
|---|---|---|---|
| *SIFT + FVs [7]* | — | — | *26.2%* |
| 1 CNN | 40.7% | 18.2% | — |
| 5 CNNs | 38.1% | 16.4% | **16.4%** |
| 1 CNN* | 39.0% | 16.6% | — |
| 7 CNNs* | 36.7% | 15.4% | **15.3%** |

# CNNs: progress




https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

# CNNs: progress




An Analysis of Deep Neural Network Models for Practical Applications, 2017.

https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

# CNNs: progress

| Year | CNN | Developed by | Place | Top-5 error rate | No. of parameters |
|---|---|---|---|---|---|
| 1998 | LeNet(8) | Yann LeCun et al | | | 60 thousand |
| 2012 | AlexNet(7) | Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever | 1st | 15.3% | 60 million |
| 2013 | ZFNet() | Matthew Zeiler and Rob Fergus | 1st | 14.8% | |
| 2014 | GoogLeNet(19) | Google | 1st | 6.67% | 4 million |
| 2014 | VGG Net(16) | Simonyan, Zisserman | 2nd | 7.3% | 138 million |
| 2015 | ResNet(152) | Kaiming He | 1st | 3.6% | |

https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

# Key Points

- convolutions, feature maps, kernels, ...
- subsampling/pooling
- weights sharing
- ReLU (Rectified Linear Unit)
- Data Augmentation
- Dropout