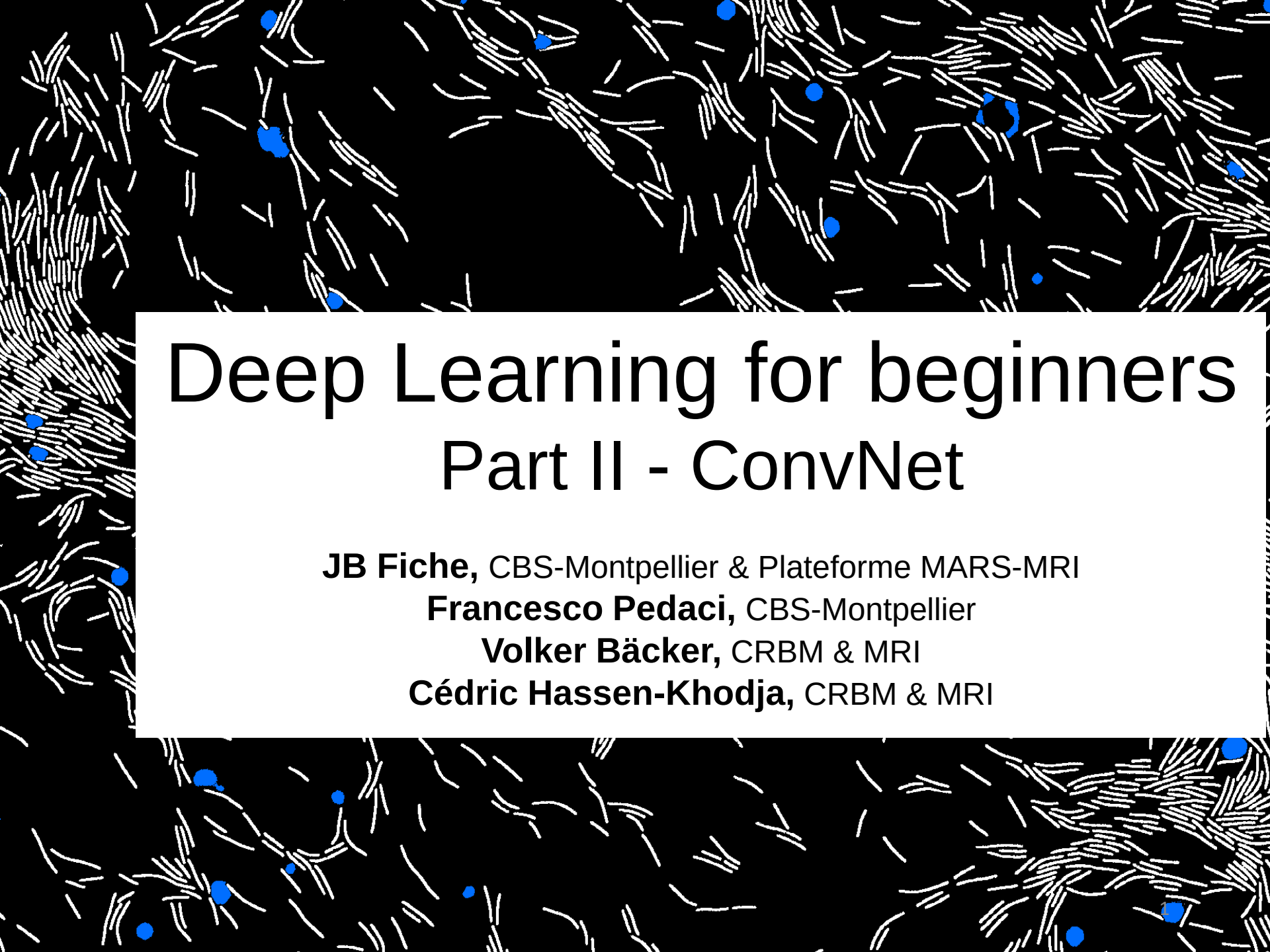


Deep Learning for beginners

Part II - ConvNet

JB Fiche, CBS-Montpellier & Plateforme MARS-MRI
Francesco Pedaci, CBS-Montpellier
Volker Bäcker, CRBM & MRI
Cédric Hassen-Khodja, CRBM & MRI

Outline

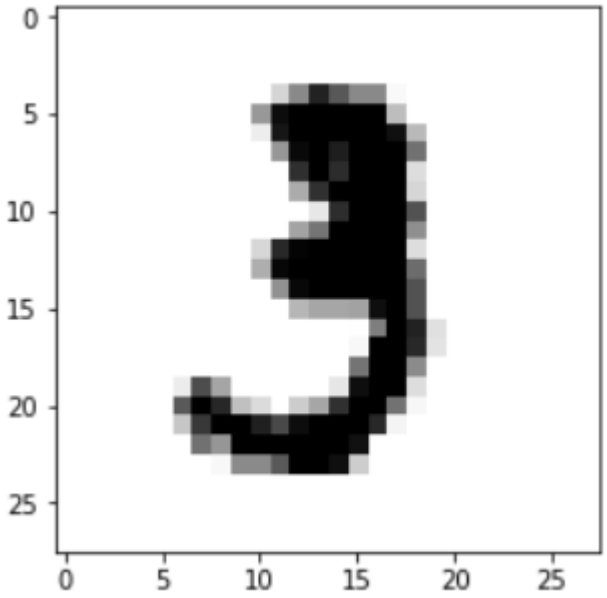
I. Introduction

II. Starting with Deep Learning and Artificial Neuron Network

III. Deep Learning for image analysis

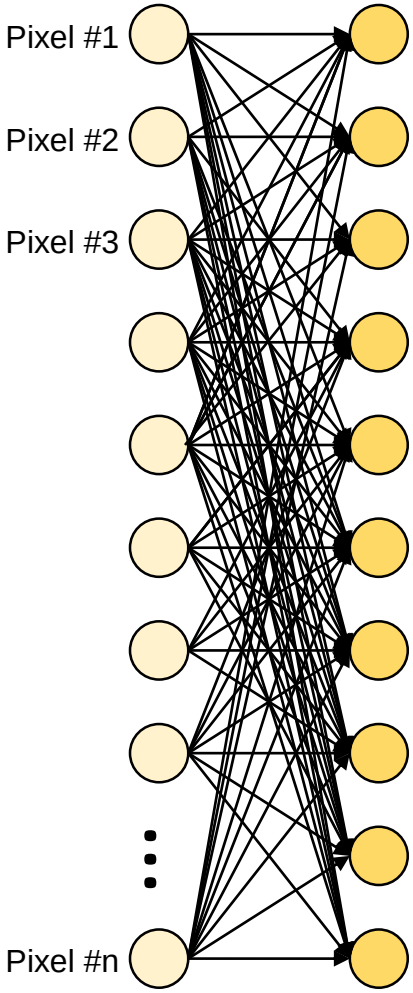
i. **Introduction to convolutional network**
ii. Application : Digit recognition
iii. Practical case 1 : image classification
iv. Practical case 2 : 2D regression of bead positions
v. Practical case 3 : Semantic segmentation of bacteria

Analyze images with densely connected networks



28x28 pixels image

784 inputs

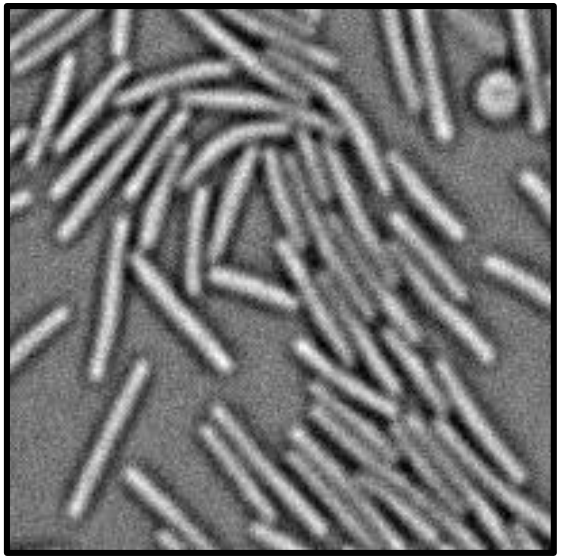


First layer 10 neurons

Number of parameters:

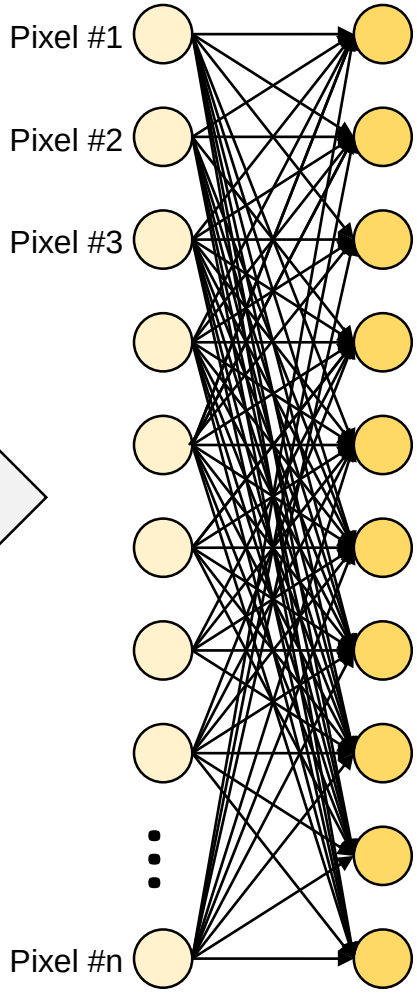
784*10+10 = 7850

Analyze images with densely connected networks



200x200 pixels image

40,000 inputs



First layer 10 neurons

Number of parameters:

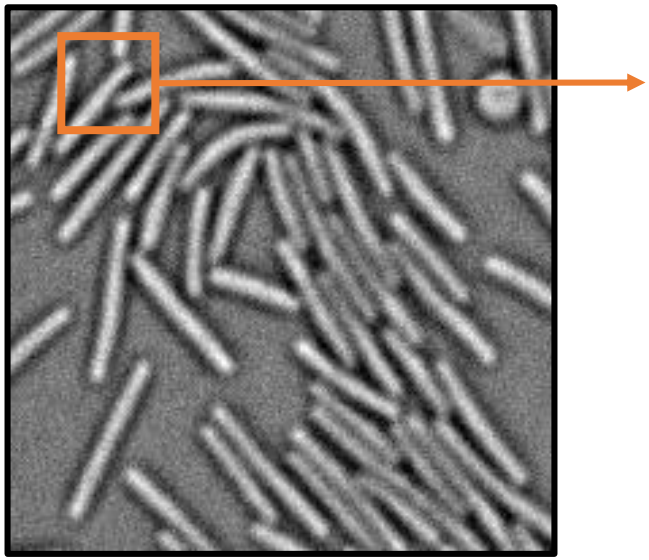
40,000*10+10 = 400,010 (!!!)

Analyze images with densely connected networks

Densely connected networks are **not suited** for image analysis:

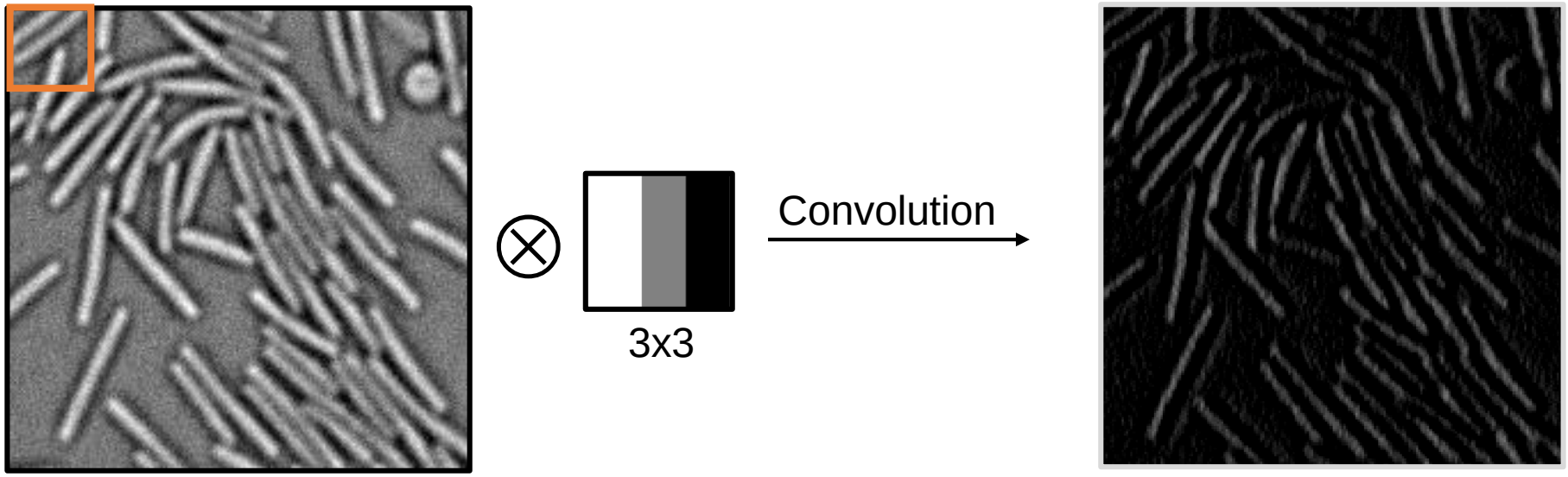
- Too many parameters, even for small networks
- Loses the local information around each pixel

Go local

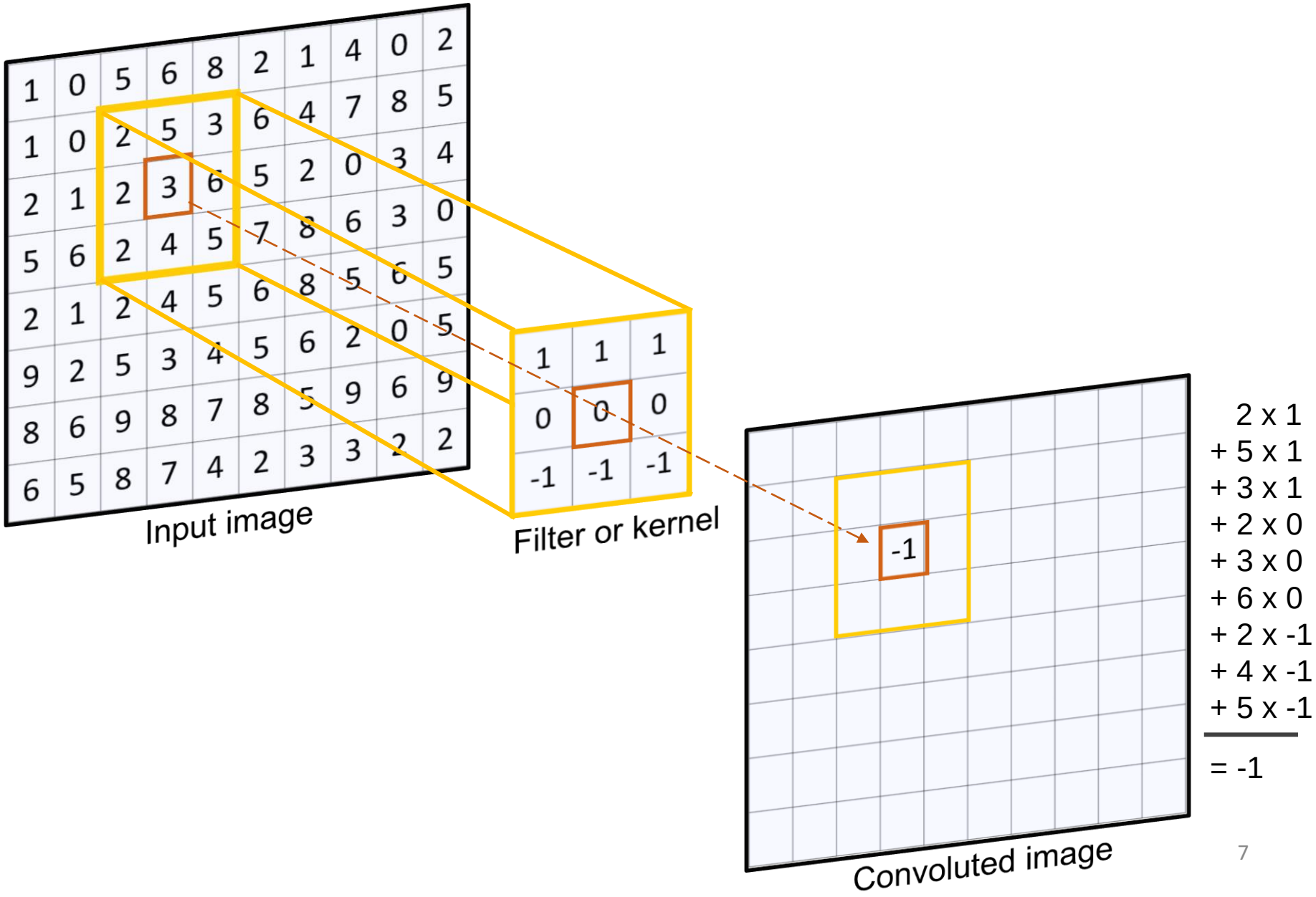


Learn local representations with **convolutional** network.

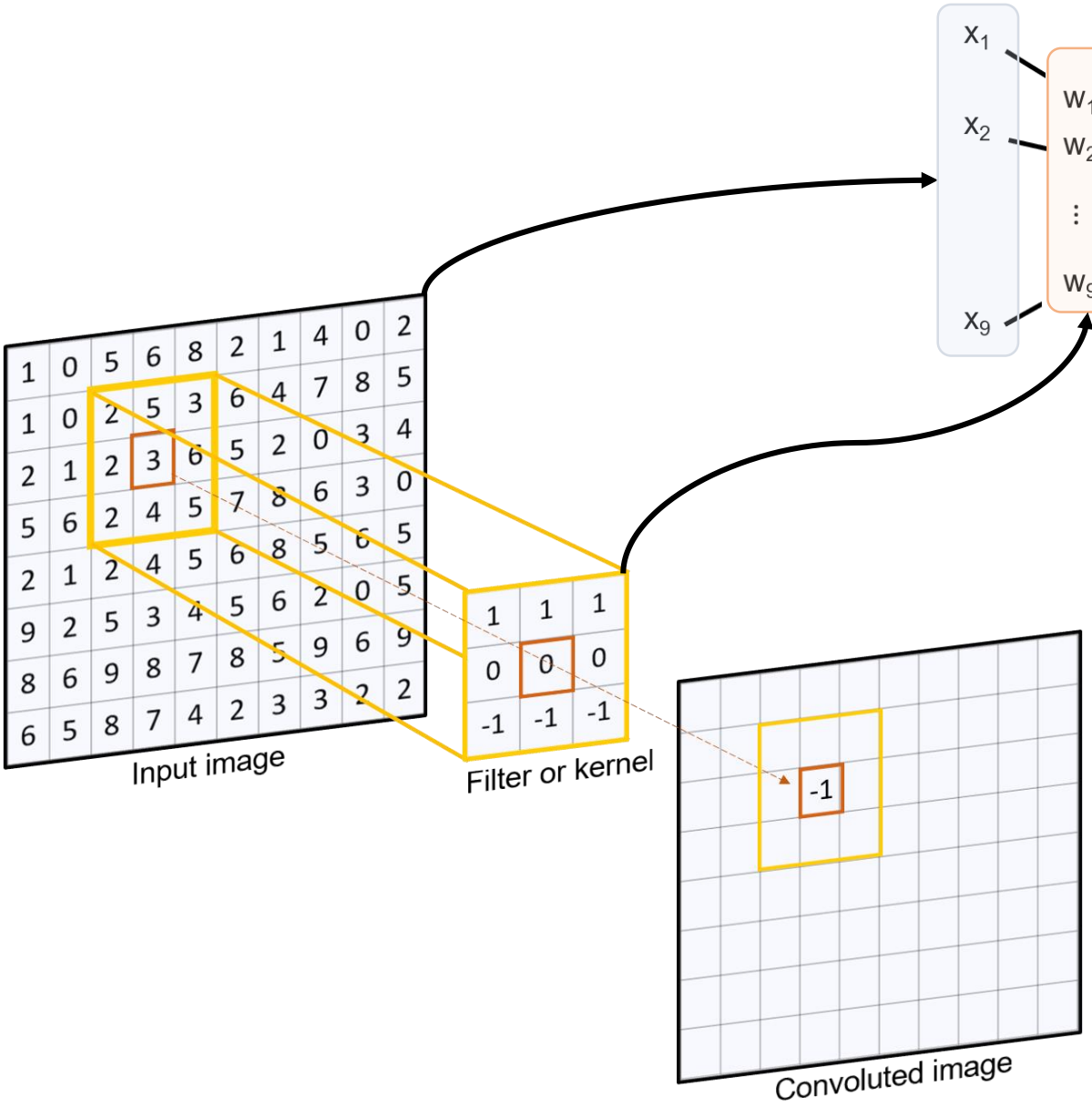
Convolutional neural network



Convolutional operation …

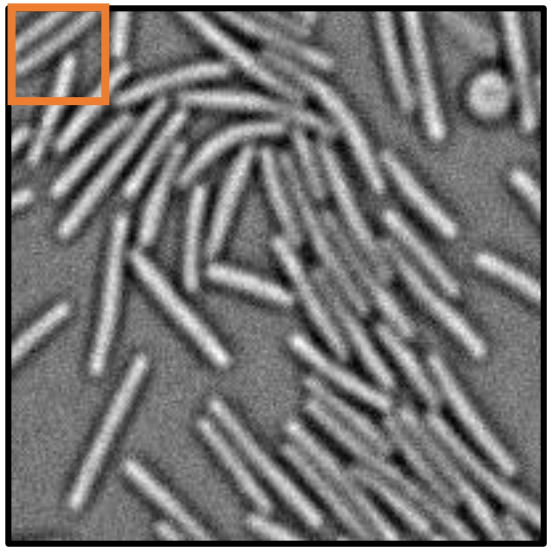


Convolutional operation and single neuron operation

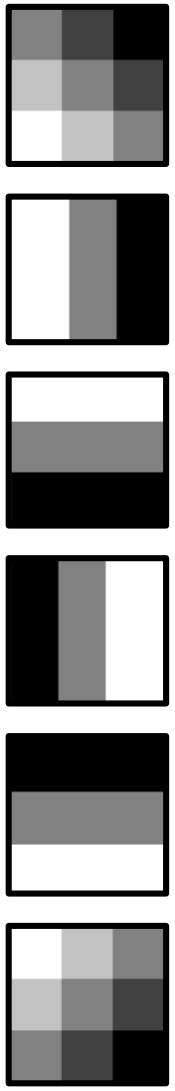


In a convolution neural network, each set of weights associated to a neuron is **equivalent to a filter**.

Convolutional neural network



$\otimes$



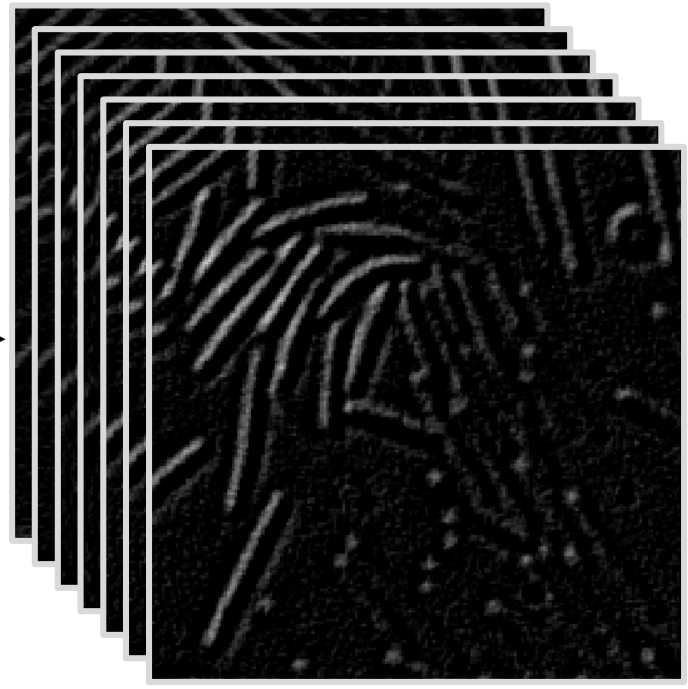
$\varphi(XW^1)$

$\varphi(XW^2)$

$\varphi(XW^n)$

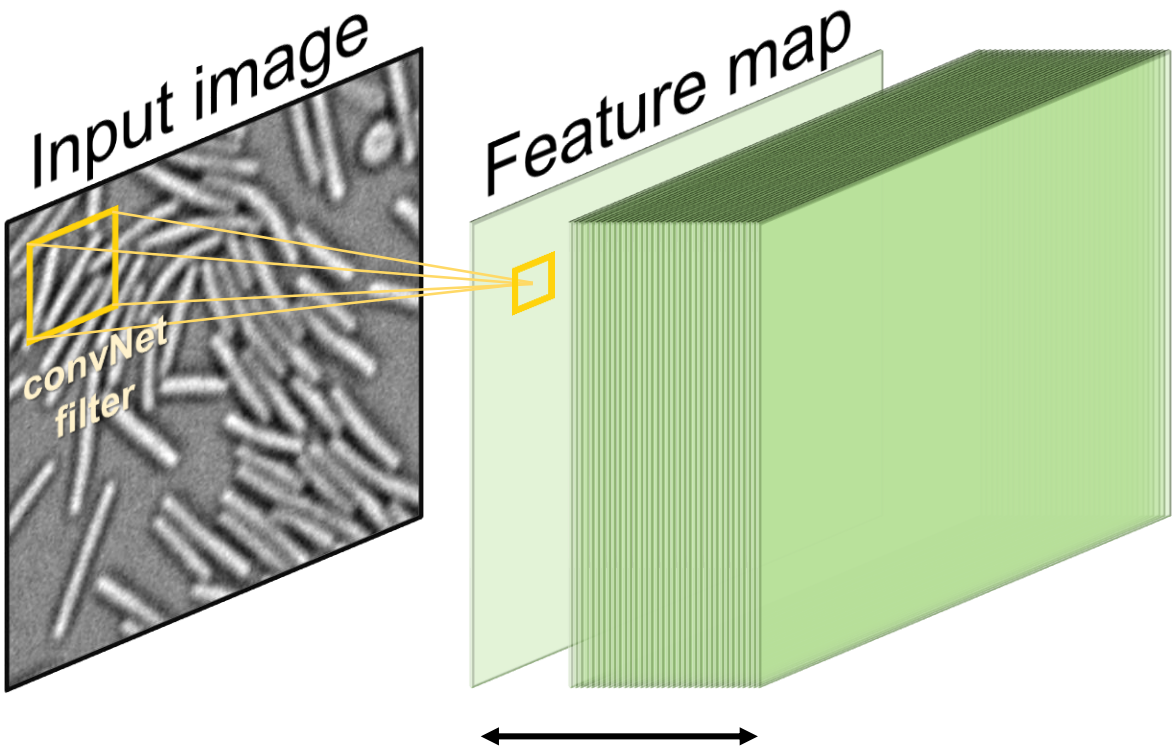
Convolution
& ReLu

Each filters will help **highlights specific & useful features** of the image (edge orientation, poles, texture, etc.)



Features map

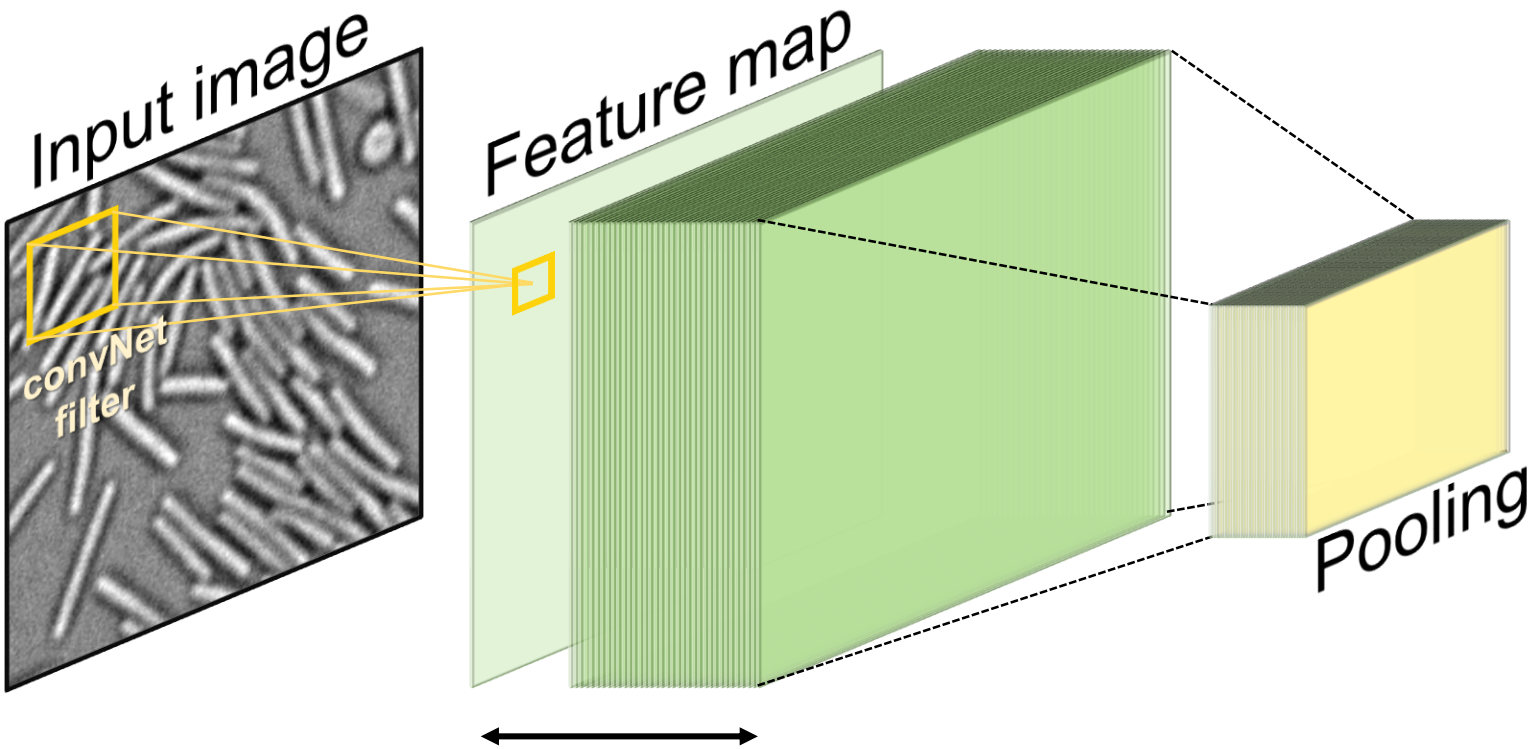
Convolutional neural network



Feature maps

There will be as **many maps as neurons in the convolution layer**

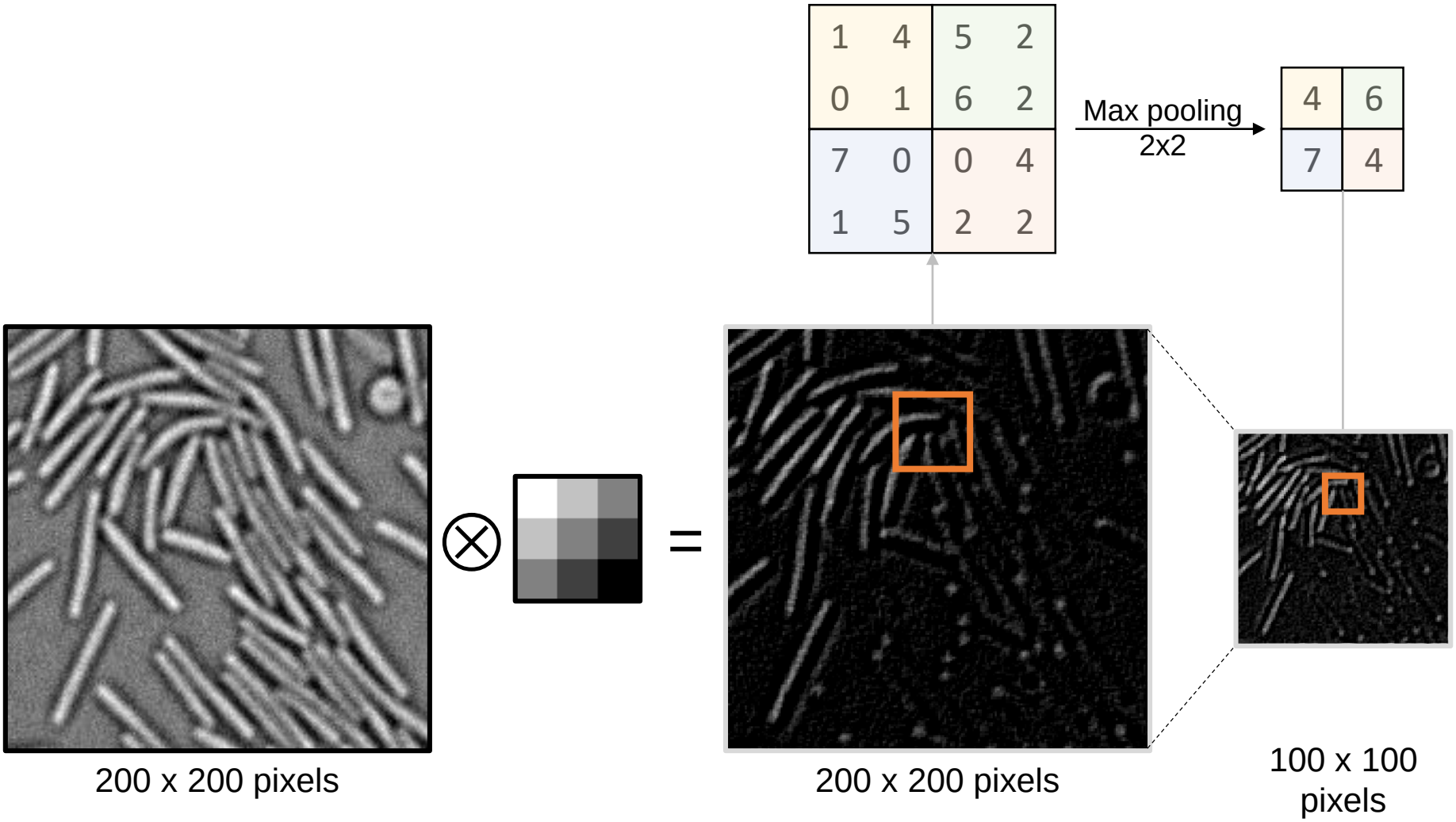
Convolutional neural network



Feature maps

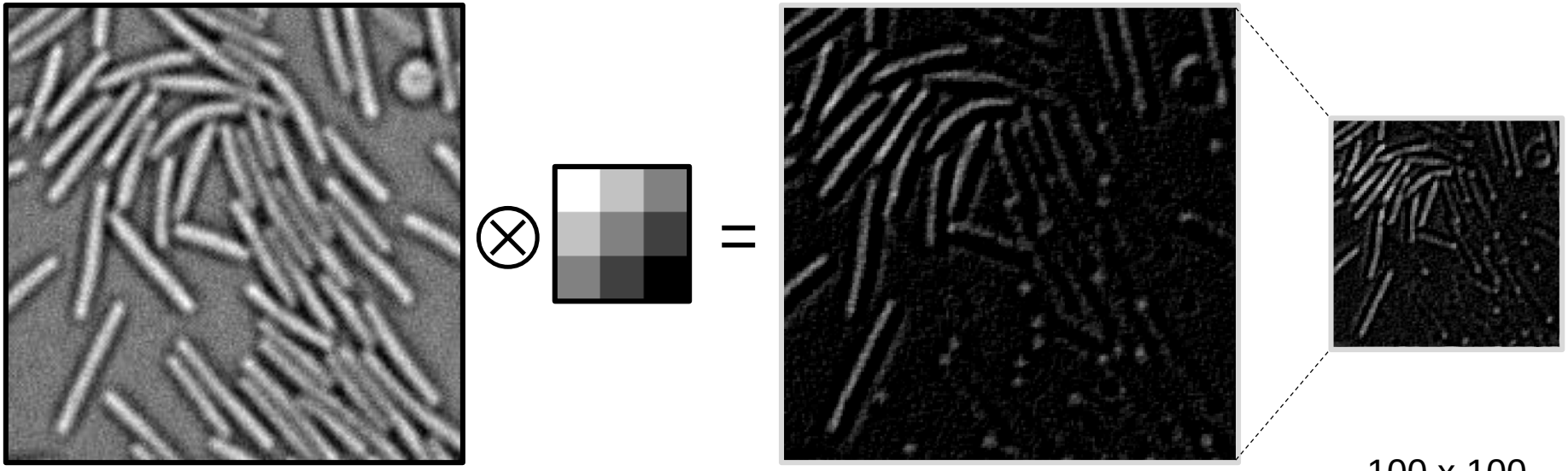
There will be as **many maps as neurons in the convolution layer**

Goal of the max-pooling layer ?



Goal of the max-pooling layer ?

1. **Reduce the spatial resolution** of the feature maps while keeping only the most relevant information
2. **Lowering memory and computing requirements**
3. Create **translation invariance**



200 x 200 pixels

200 x 200 pixels

100 x 100 pixels

Convolutional network architecture

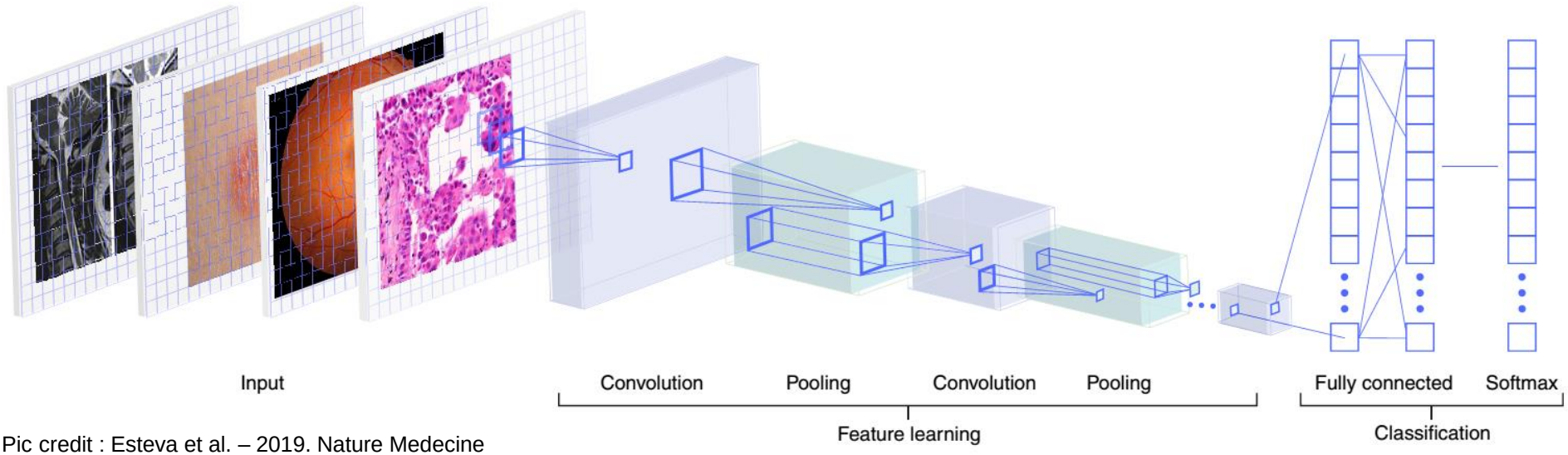
Classifier based on convNet are always built the same way:

1. A first part of the network is **using convolution layers to learn the features**

    - **Convolution layers → features extraction**
    - **Pooling → downsampling**

2. A second part is then classifying those features using **densely connected layers** in order to predict the right output.

    - Lots or parameters → **heavy on the memory**
    - Image input size is fixed → **not flexible**



Pic credit : Esteva et al. – 2019. Nature Medecine

Outline

I. Introduction
II. Starting with Deep Learning and Artificial Neuron Network
III. Deep Learning for image analysis

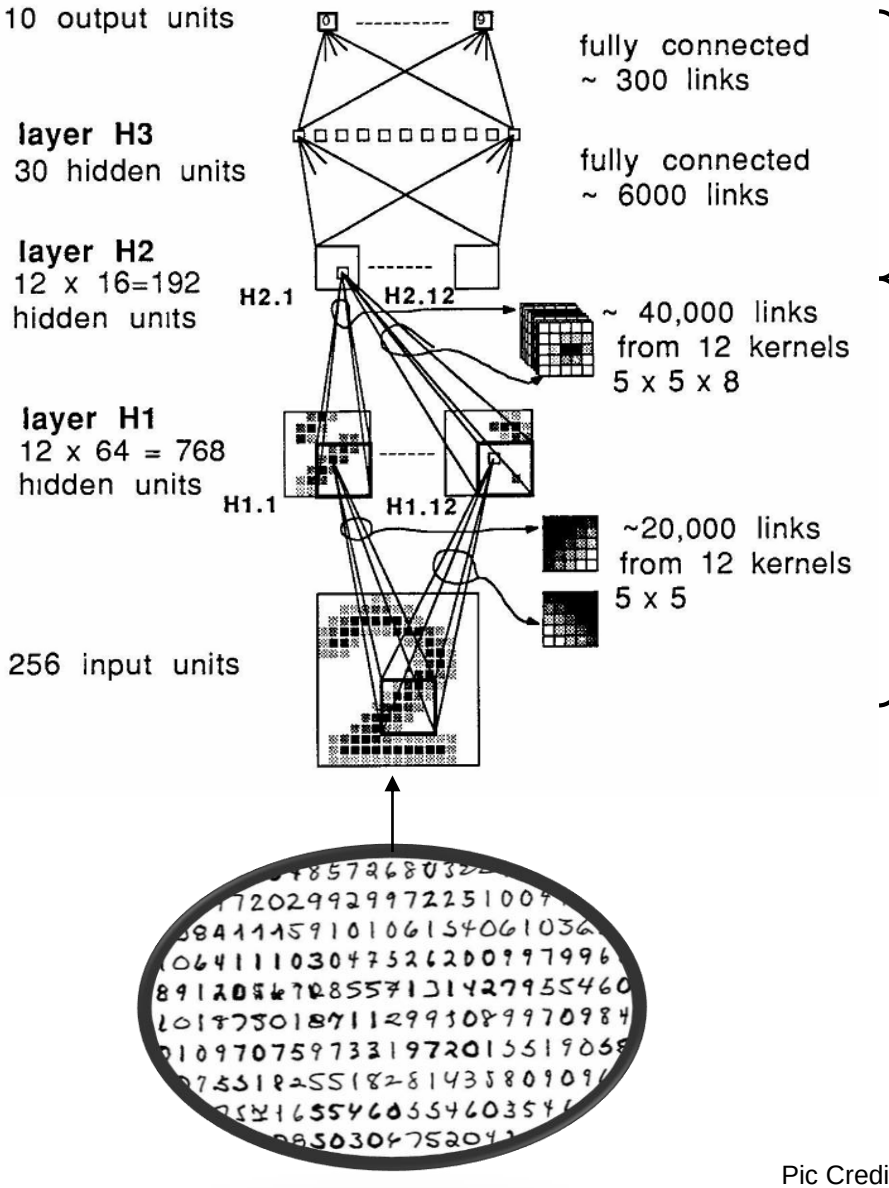
i. Introduction to convolutional network
ii. Application : Digit recognition
iii. Practical case 1 : image classification
iv. Practical case 2 : 2D regression of bead positions
v. Practical case 3 : Semantic segmentation of bacteria

Classify hand-written digits

Example n°1 : MNIST_convNet

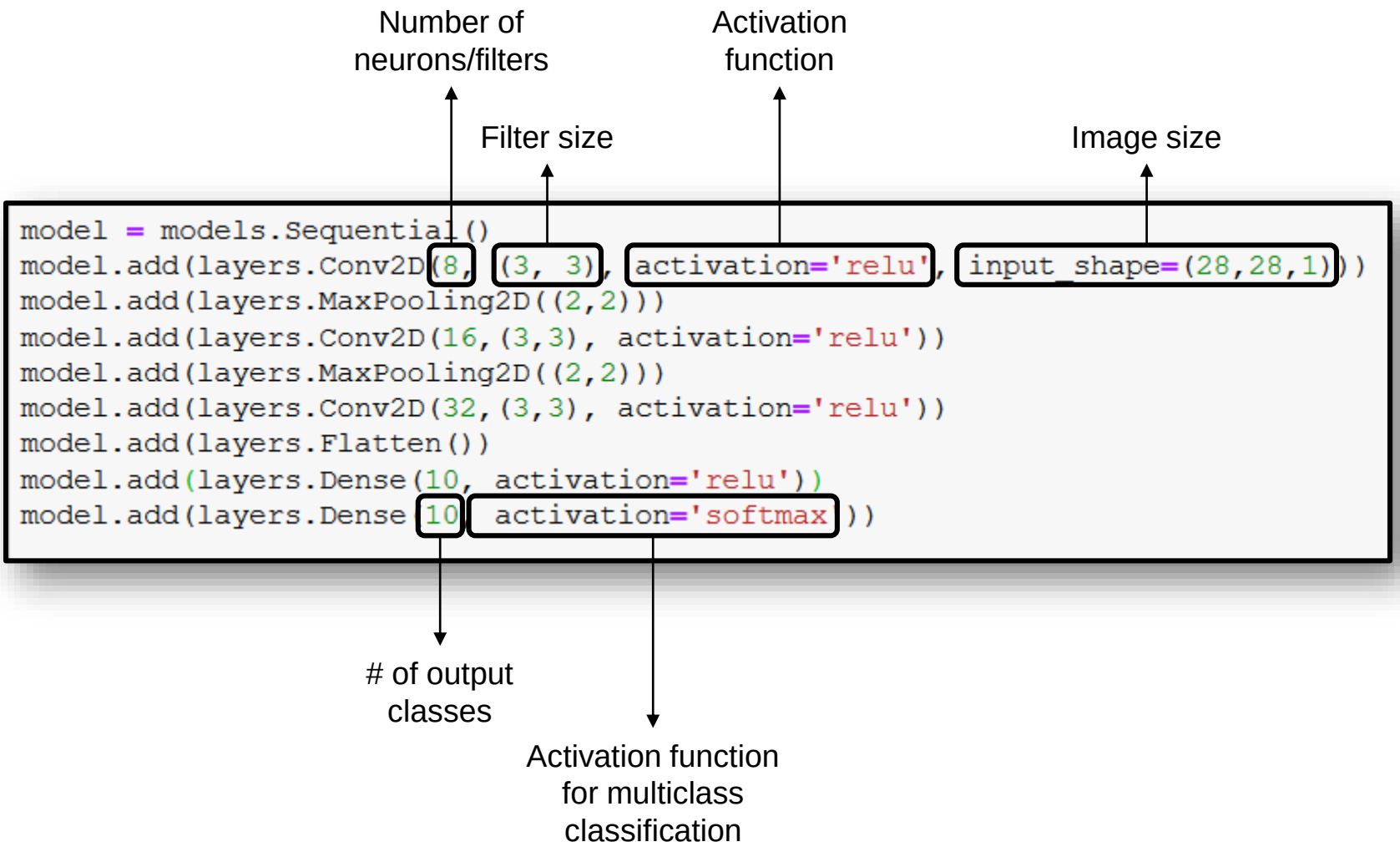
1. Understand and build your own ConvNet
2. Test this architecture for image classification

Hand-written zipcode classification - 1989



Pic Credit: LeCun et al. 1989. Neural Computation

Hand-written digit classification with a ConvNet



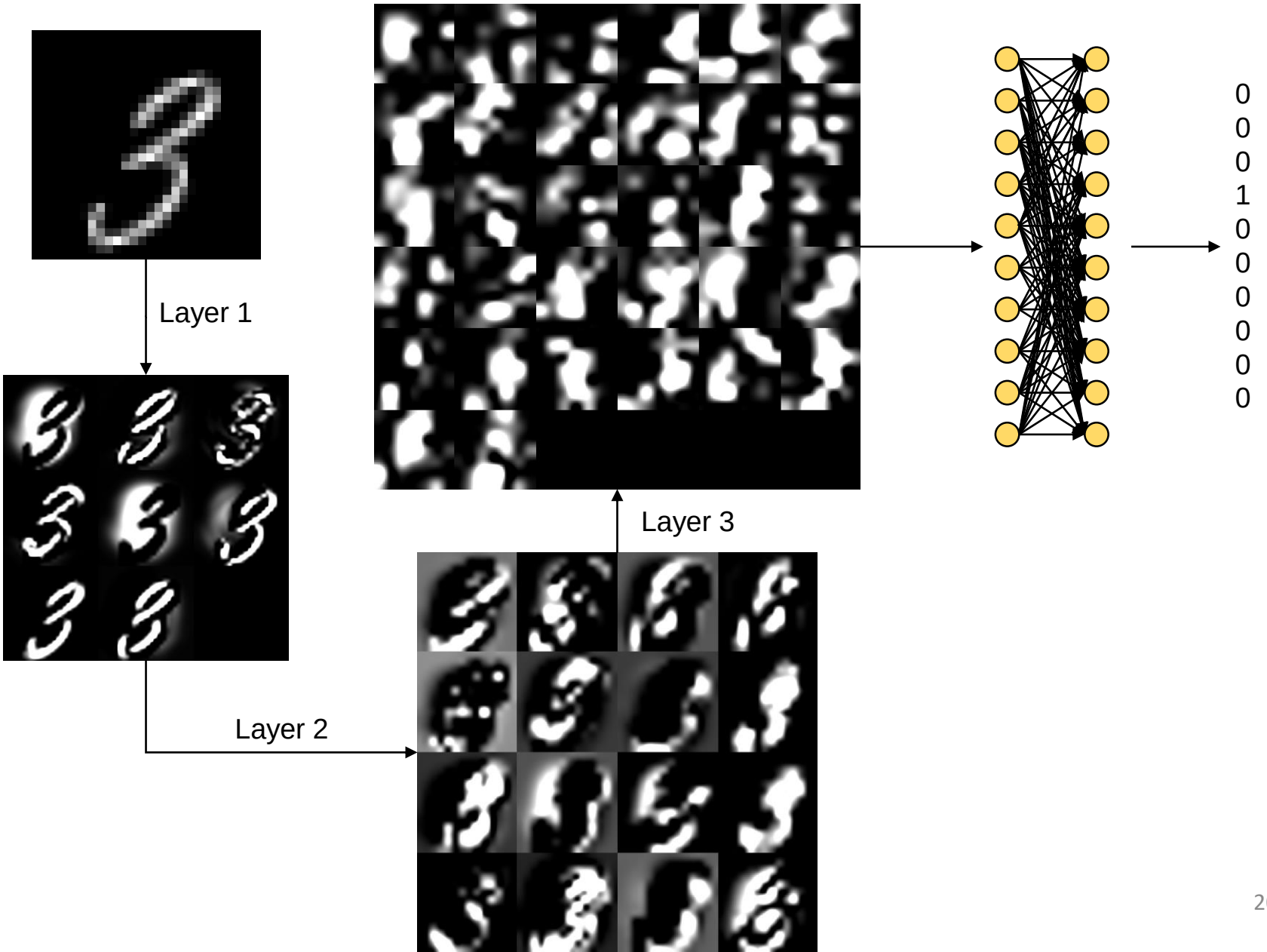
Hand-written digit classification with a ConvNet

```
model = models.Sequential()
model.add(layers.Conv2D(8, (3, 3), activation='relu', input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(16,(3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(32,(3,3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 26, 26, 8)         80
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 13, 13, 8)         0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 11, 11, 16)        1168
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 3, 3, 32)          4640
_________________________________________________________________
flatten_5 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                2890
_________________________________________________________________
dense_9 (Dense)              (None, 10)                110
=================================================================
Total params: 8,888
Trainable params: 8,888
Non-trainable params: 0
_________________________________________________________________
```

>30% of the parameters

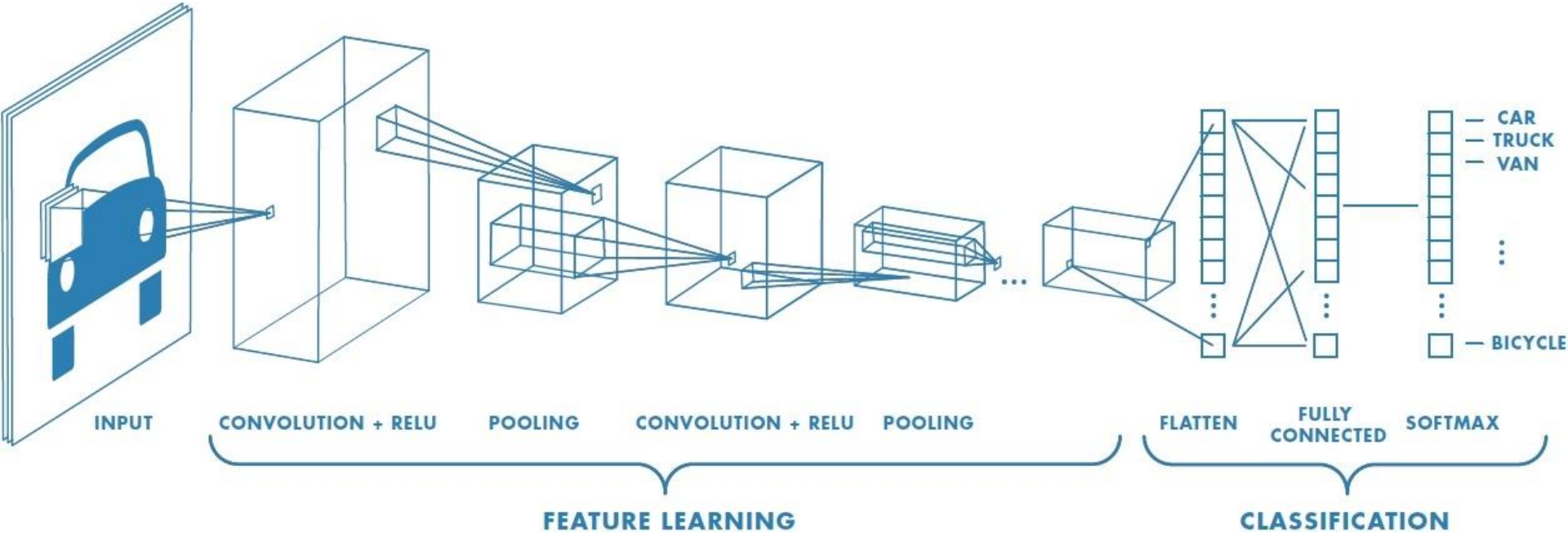
What are the features?



What did we learn?

How to build & train a simple classifier based on a convNet

```
model.add(layers.Conv2D(8, (3, 3), activation='relu', input_shape=(28,28,1)))

model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Flatten())

model.add(layers.Dense(10, activation='relu'))
```



Outline

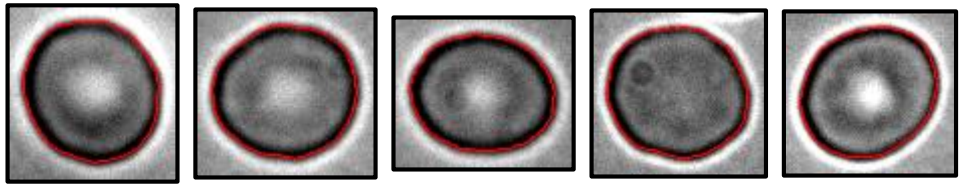
I. Introduction
II. Starting with Deep Learning and Artificial Neuron Network
III. Deep Learning for image analysis

i. Introduction to convolutional network
ii. Application : Digit recognition
iii. Practical case 1 : image classification
iv. Practical case 2 : 2D regression of bead positions
v. Practical case 3 : Semantic segmentation of bacteria

Deep Learning applied to **image classification**

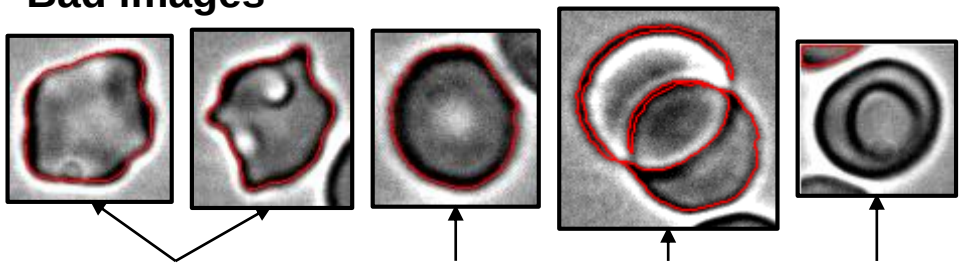
Viviana Claveria Manouk Abkarian

Good images



In-focus image of RBC with a properly defined contour (red)

Bad images

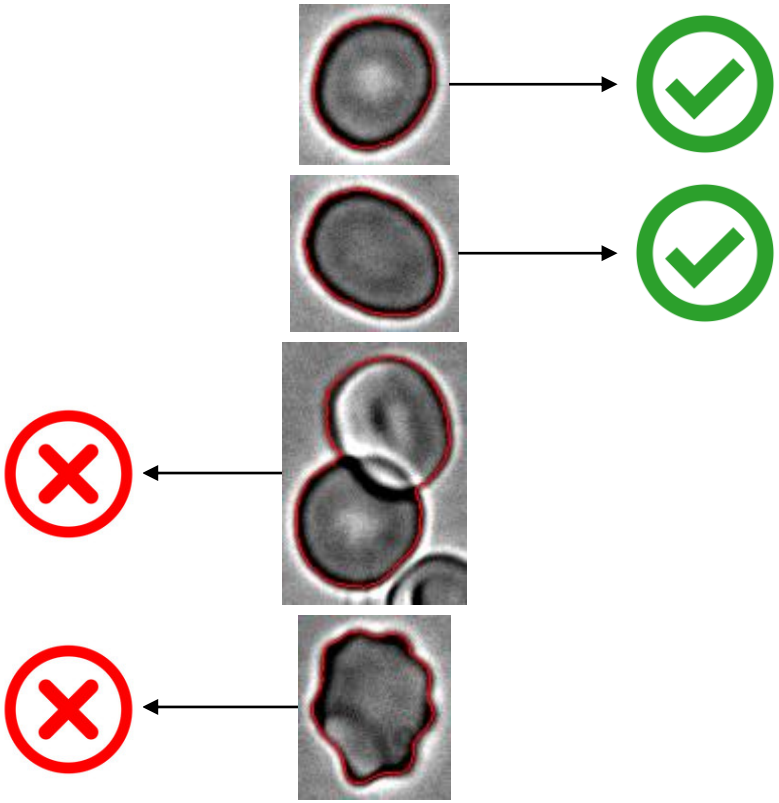


Poor quality images that need to be discarded from the analysis

Classify red blood cell (RBC) images

Example n°2 : RBC_image_classification_no_image_augmentation

1. Build your own ConvNet classifier to solve a biological problem
2. Understand how to detect overfitting and which strategies to avoid it
3. How already-trained network can be used for classification



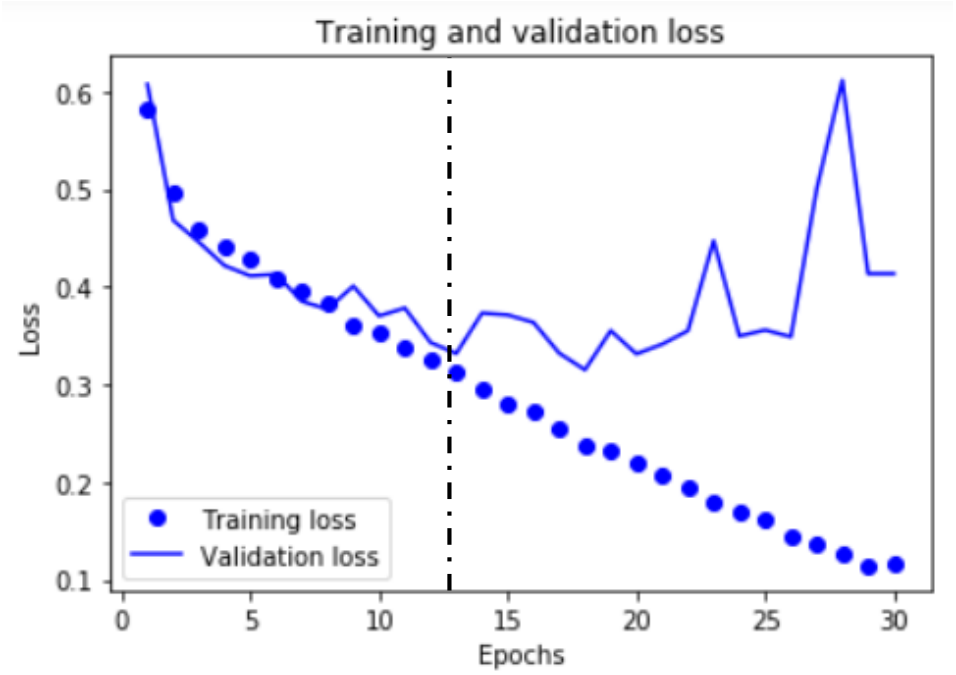
One possible solution …

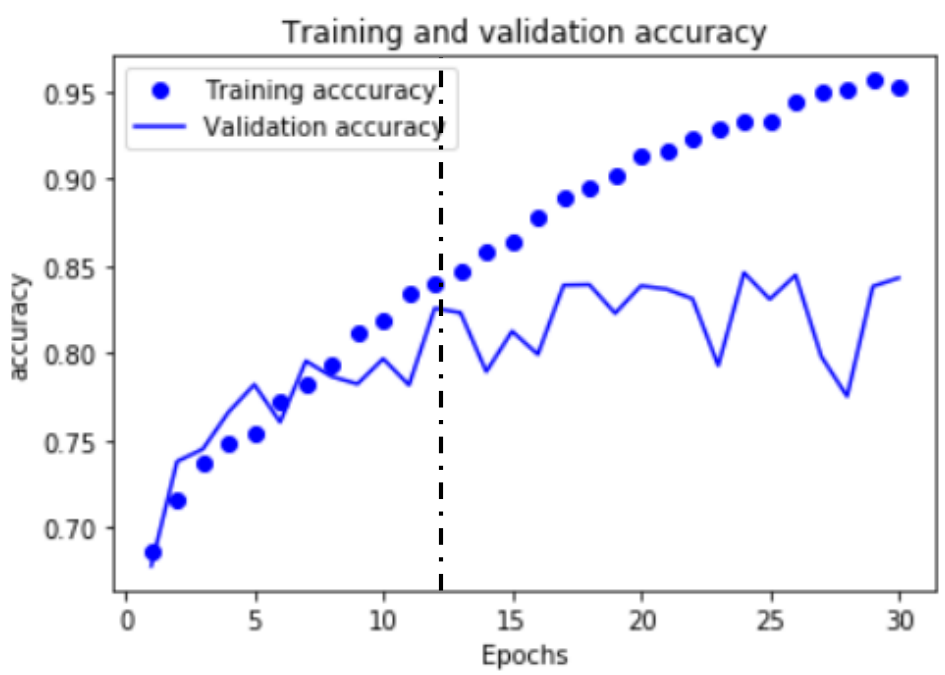
Type of data : png RGB images resize to 50x50 and **normalized**
Size of the training set : 956 good images / 2000 bad images
Size of the validation/testing sets : 319 good images / 667 bad images

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(50,50,3)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer = optimizers.RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Number of trainable parameters : 405,185

… but overfitting





Using the validation set, we observe that the training and validation loss & accuracy are diverging after epoch #12. **The network is no longer learning new useful features**.

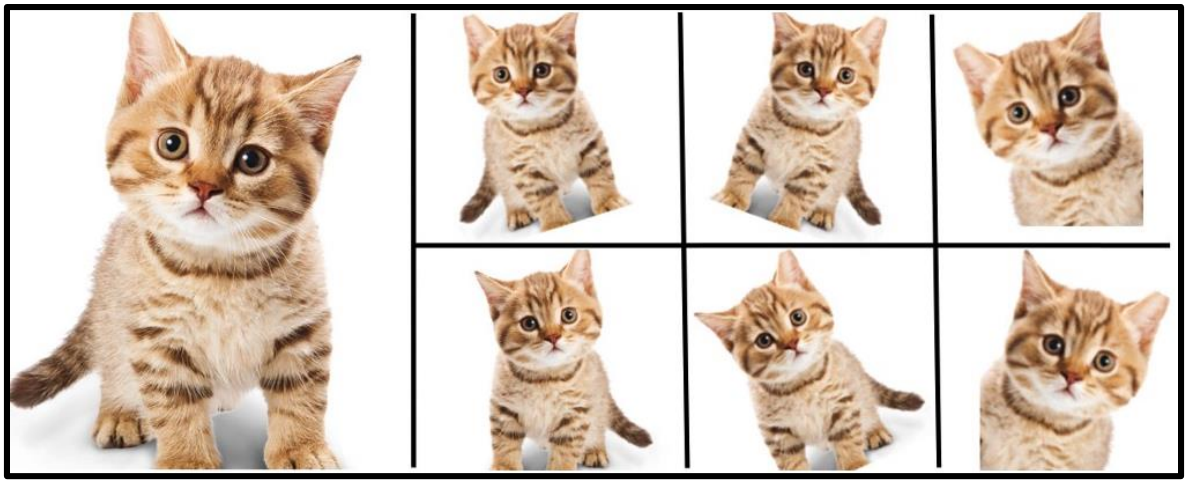
- Overfitting
- Validation loss & accuracy are noisy
- The global accuracy of the network is ~84%

How to avoid overfitting?

- **Reduce the size of the network** … but no magical formula to determine how many layers we need

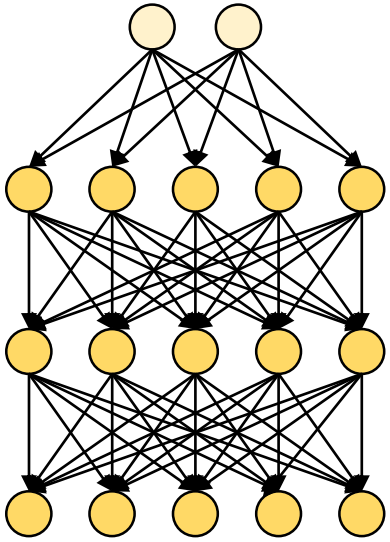
- Increase the size of the training set:

    - Actually **increase the number of images**
    - Use **image augmentation** strategy

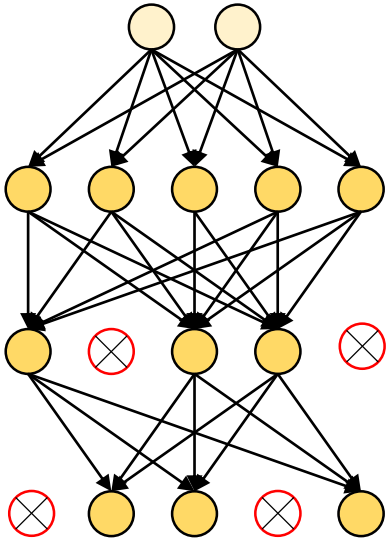


How to avoid overfitting?

- **Reduce the size of the network** … but no magical formula to determine how many layers we need
- Increase the size of the training set:
    - Actually **increase the number of images**
    - Use **image augmentation** strategy
- **Dropout**, randomly "turning-off" neurons of the network



No dropout



Dropout with 40% probability

Dropout is used to avoid co-adaptation of neurons → enforce the fact that neurons should **learn and work independently**.

How to avoid overfitting?

- **Reduce the size of the network** … but no magical formula to determine how many layers we need

- Increase the size of the training set:

    - Actually **increase the number of images**
    - Use **image augmentation** strategy

- **Dropout**, randomly "turning-off" neurons of the network

- **Weight regularization L_1 & L_2**, a strategy to force the weights to take only small values during the training:

    - L_1 – the cost added to the loss is proportional to the absolute weight value
    - L_2 – the cost is proportional to the square value of the weight

Effect of image augmentation & dropout

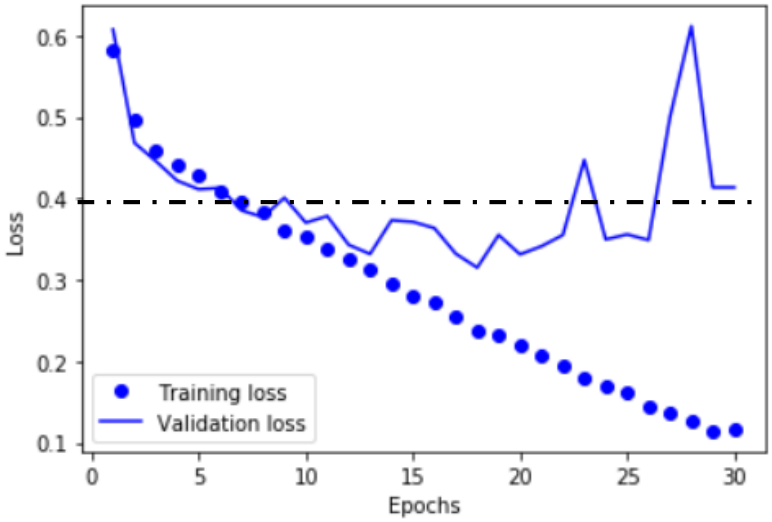
Type of data : png RGB images resize to 85x85 and **normalized**
Size of the training set : 956 good images / 2000 bad images
Size of the validation/testing sets : 319 good images / 667 bad images

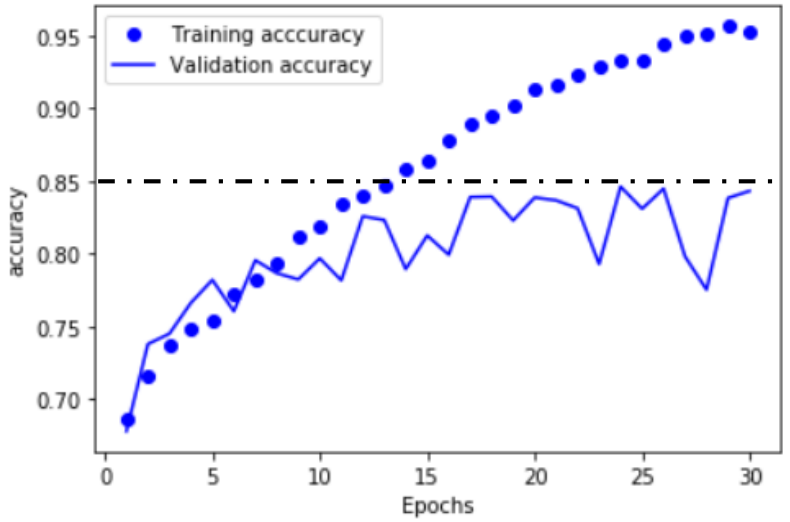
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(85,85,3)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer = optimizers.RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Number of trainable parameters : 1,453,761

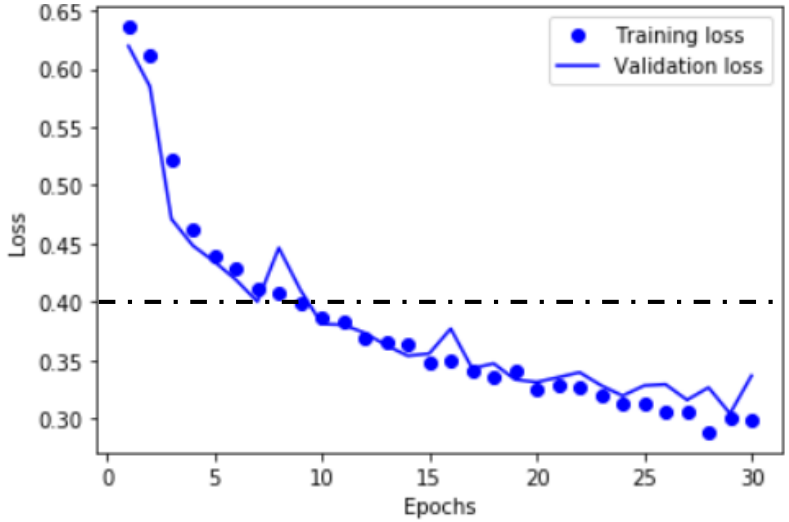
Better but the accuracy is still <90%

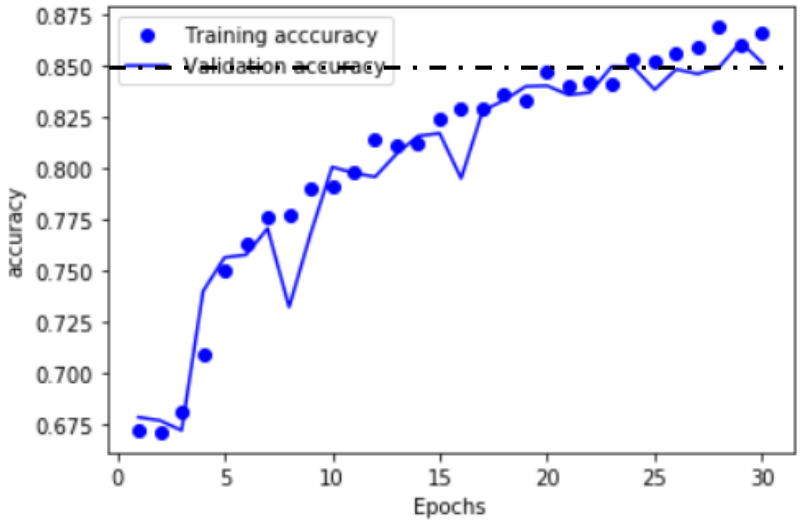
Basic convNet





ConvNet with image augmentation & dropout

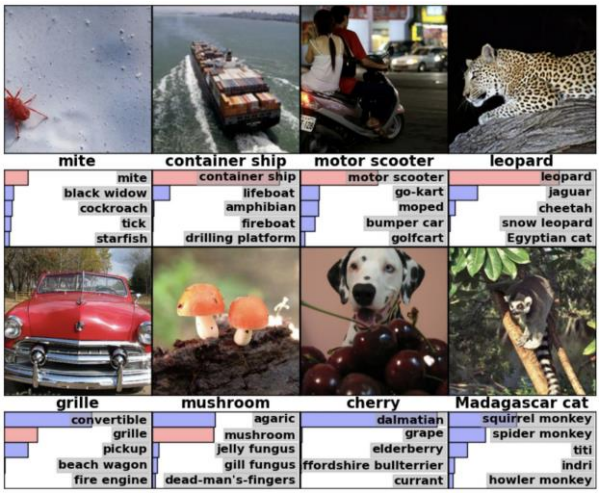


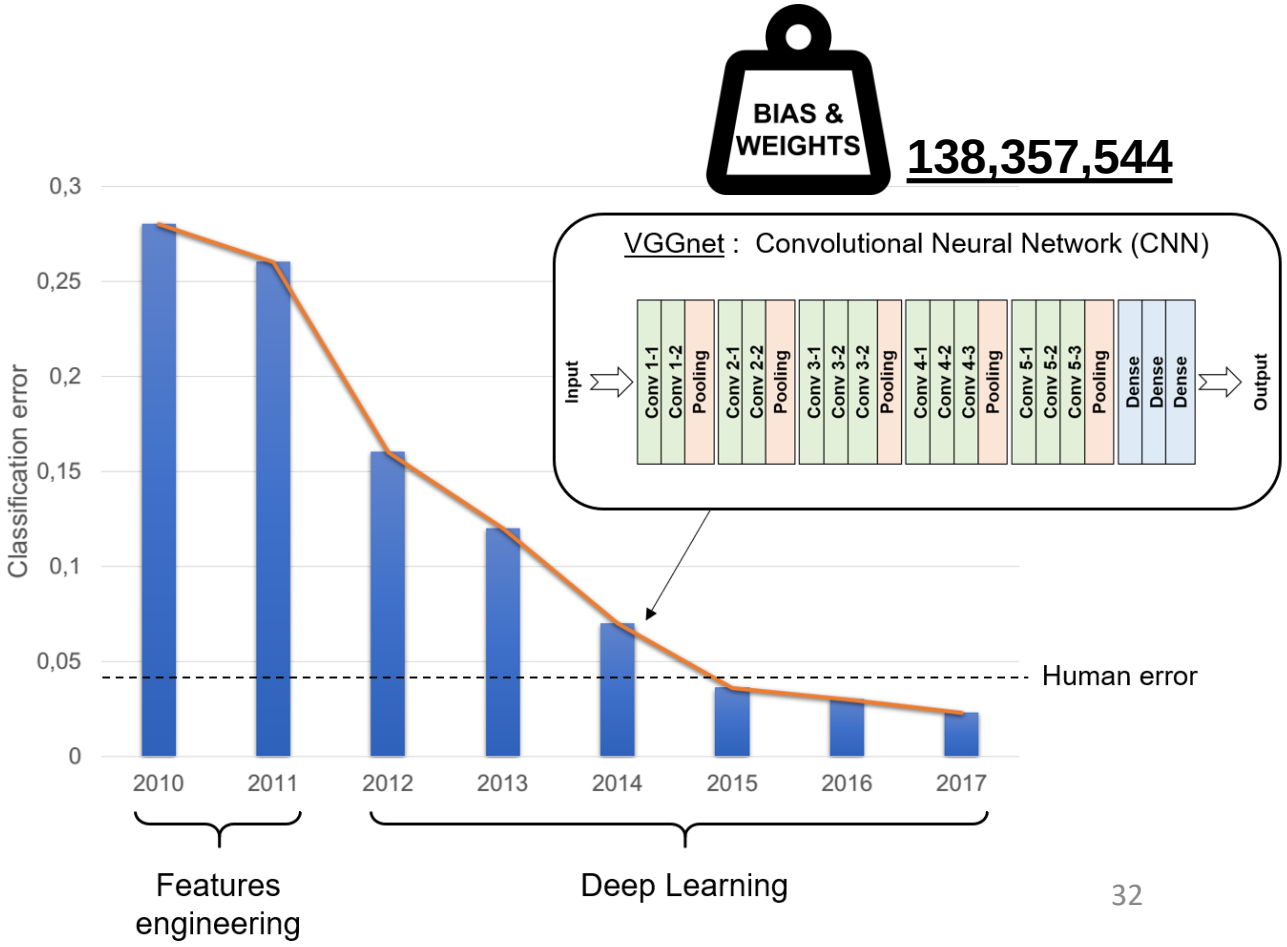


Used a pre-trained convNet

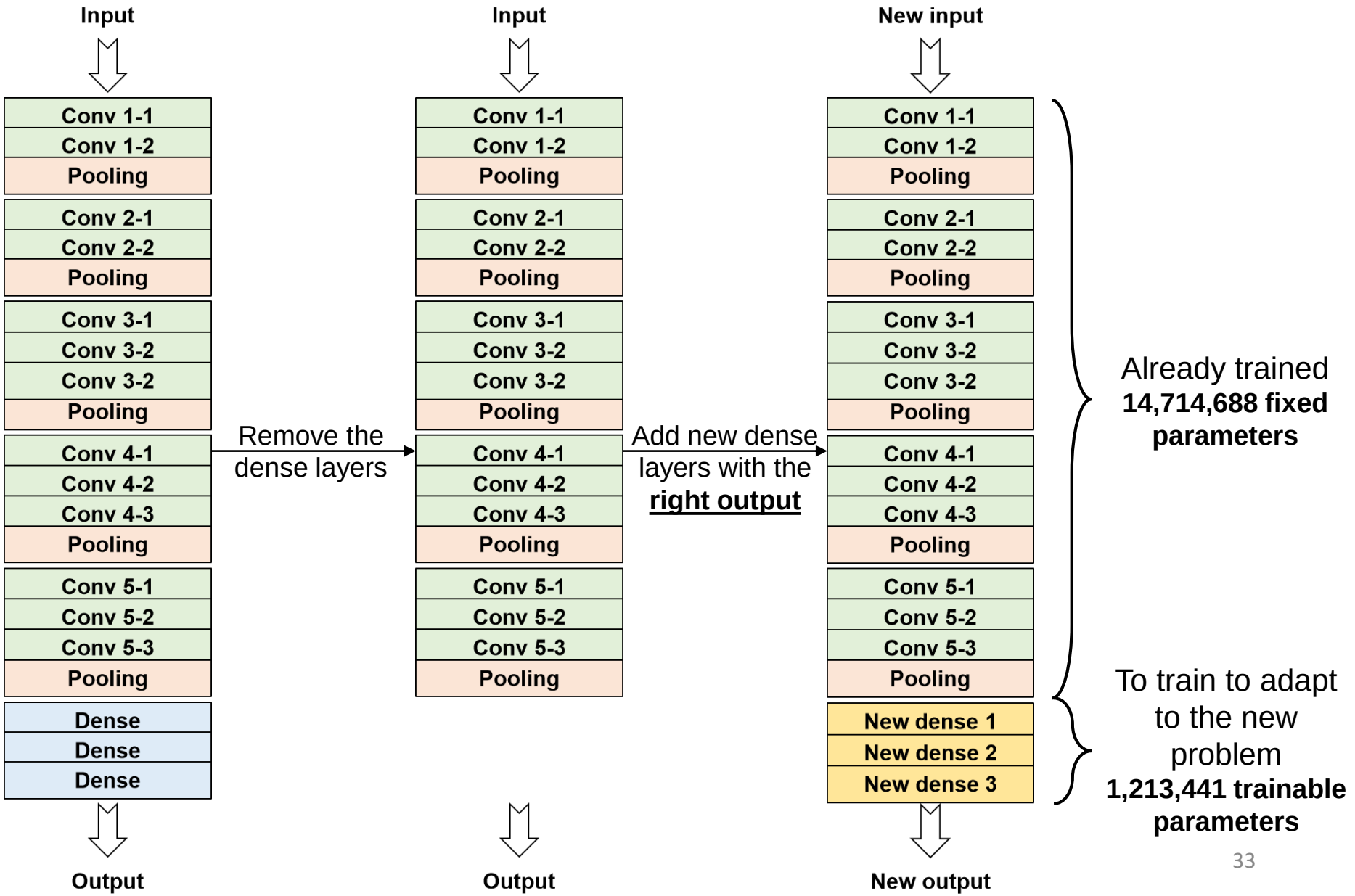
Many networks used for the **Image Large Scale Visual Recognition Challenge** are available and already trained on million of images.

https://keras.io/applications/





Used a pre-trained convNet : VGG16



Used a pre-trained convNet : VGG16

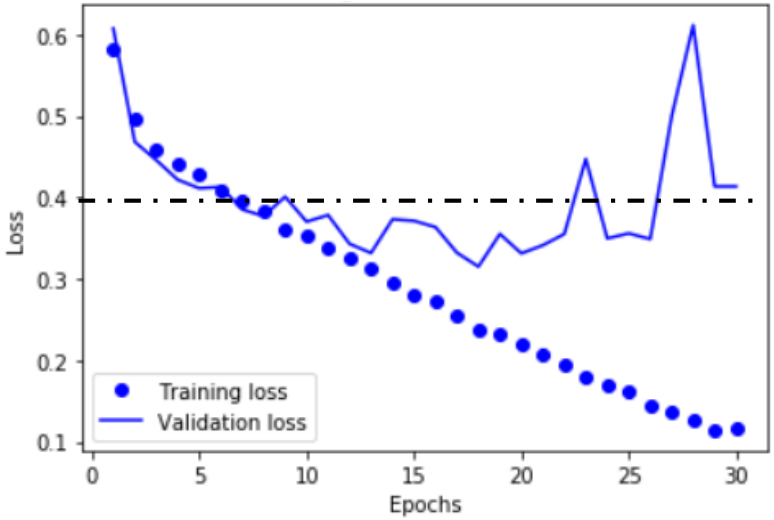
```
from keras.applications import VGG16
conv_base = VGG16(weights='imagenet',include_top = False,input_shape=(85,85,3))

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

conv_base.trainable = False
```

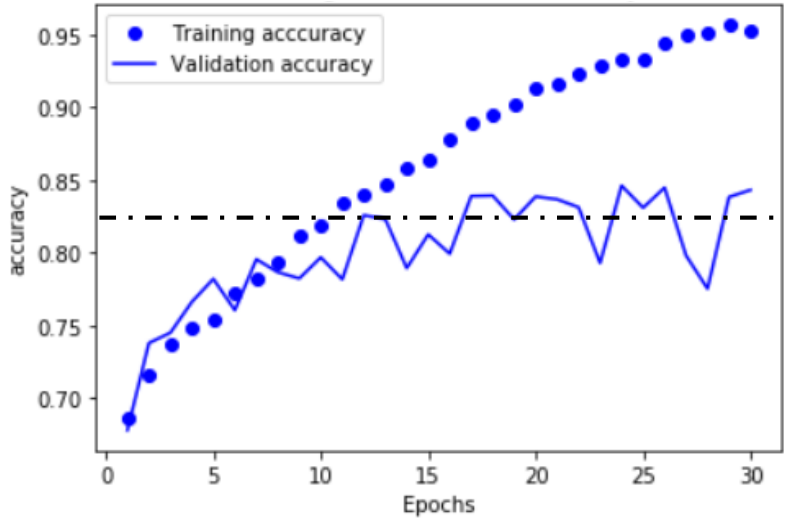
Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 2, 2, 512)	14714688
flatten_6 (Flatten)	(None, 2048)	0
dropout_16 (Dropout)	(None, 2048)	0
dense_21 (Dense)	(None, 512)	1049088
dropout_17 (Dropout)	(None, 512)	0
dense_22 (Dense)	(None, 256)	131328
dropout_18 (Dropout)	(None, 256)	0
dense_23 (Dense)	(None, 128)	32896
dense_24 (Dense)	(None, 1)	129

```
Total params: 15,928,129
Trainable params: 1,213,441
Non-trainable params: 14,714,688
```

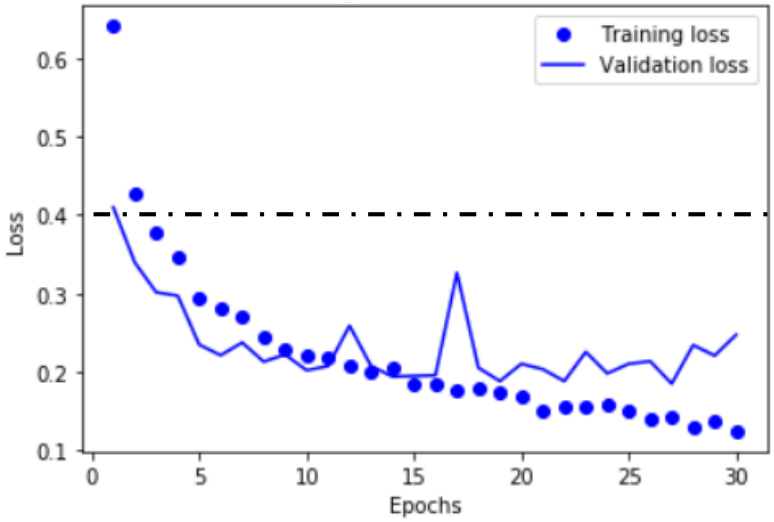
Fine tuning pre-trained network : efficient strategy!

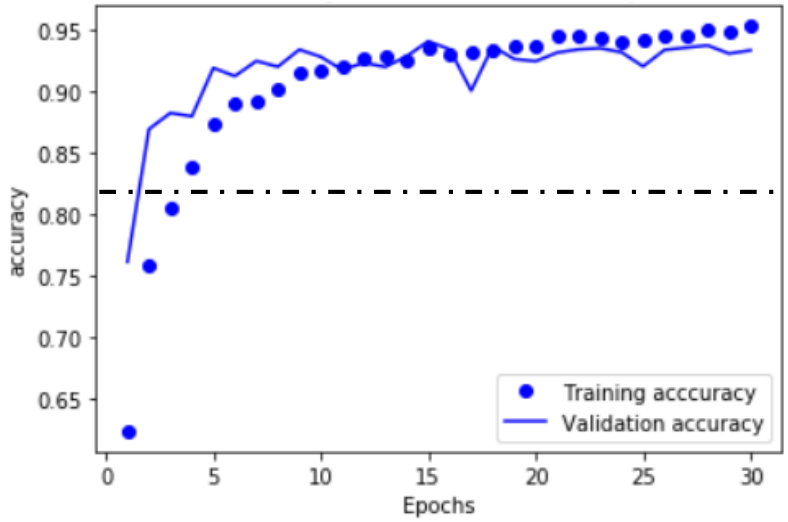
Basic convNet



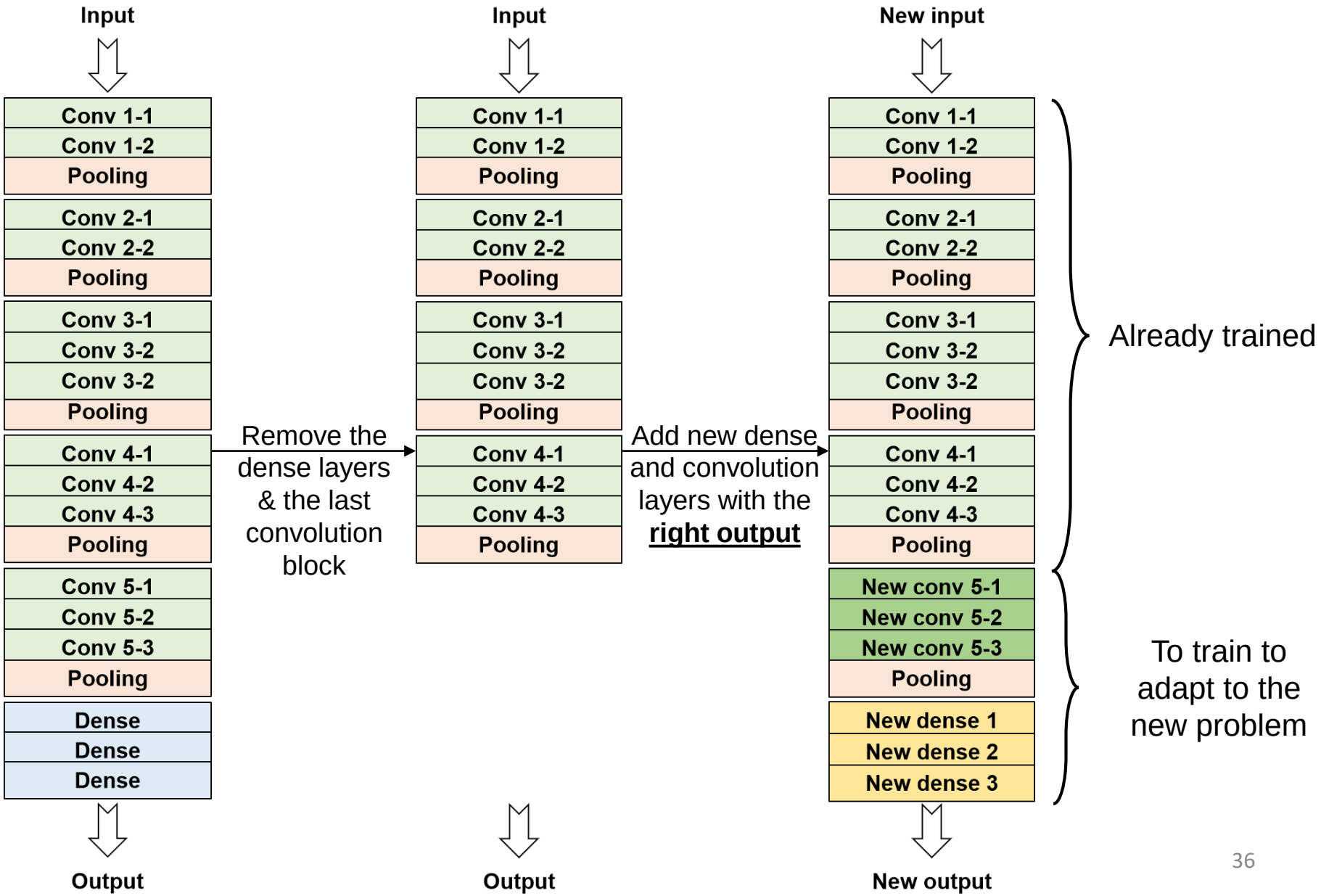


Fine-tune a pre-trained network



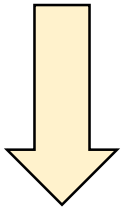


Used a pre-trained convNet : VGG16

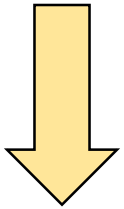


What did we learn?

Step 1 : Clarify your problem and **assemble a dataset** for the training



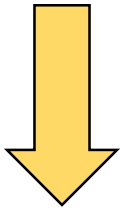
Step 2 : Try a **simple architecture** (for example MNIST) and test it



Your problem can be solved using DL?

- Training data format OK
- Measure of success
- etc.

Step 3 : Scale up the model until you **observe overfitting**



Step 4 : **Improve** your model performances

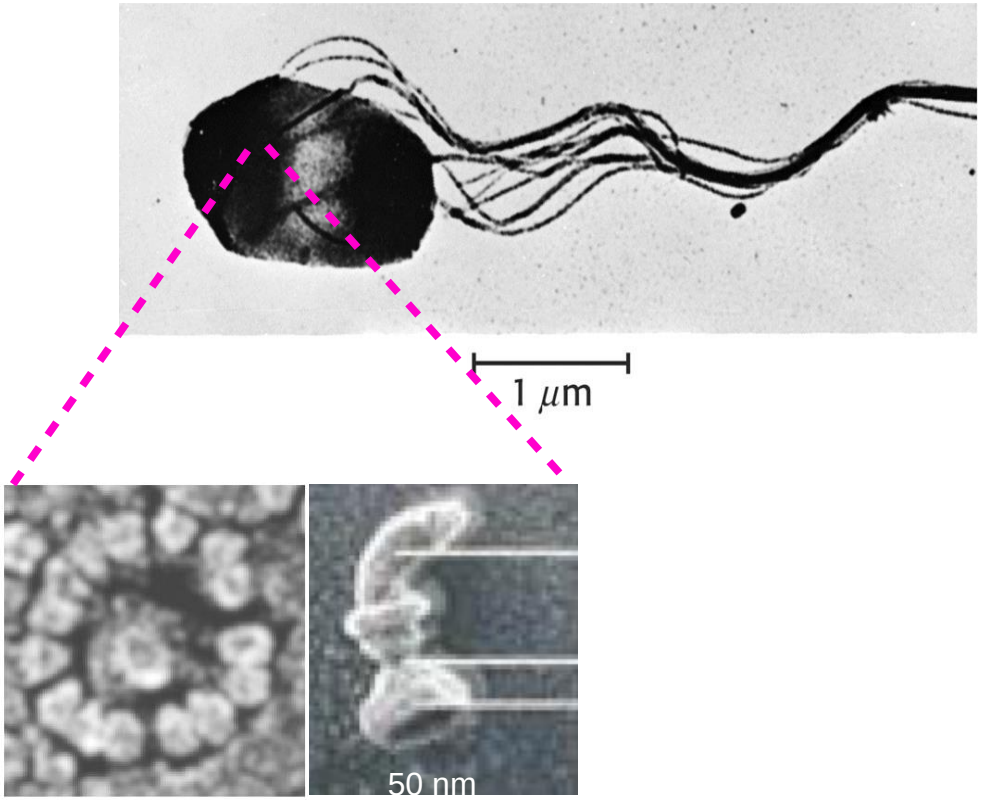
Outline

I. Introduction
II. Starting with Deep Learning and Artificial Neuron Network
III. Deep Learning for image analysis

i. Introduction to convolutional network
ii. Application : Digit recognition
iii. Practical case 1 : image classification
iv. Practical case 2 : 2D regression of bead positions
v. Practical case 3 : Semantic segmentation of bacteria

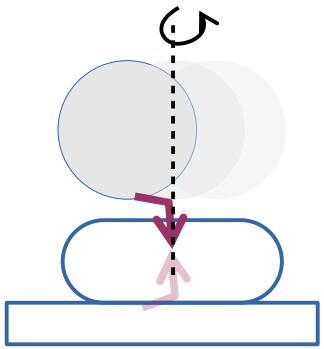
A practical application of Deep Learning for **regression**

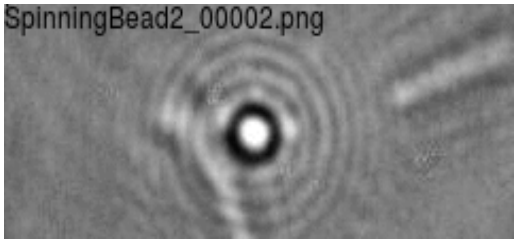
Tracking microscopic beads rotated by the flagellar motor of a bacteria



Francesco Pedaci

Ashley Nord





Outline

I. Introduction
II. Starting with Deep Learning and Artificial Neuron Network
III. Deep Learning for image analysis

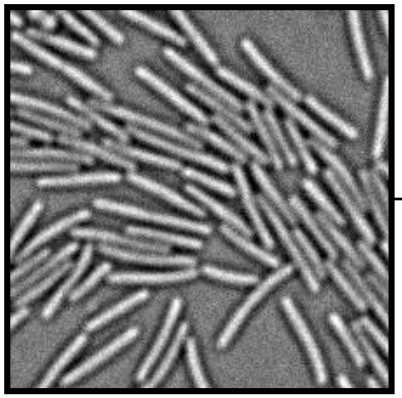
i. Introduction to convolutional network
ii. Application : Digit recognition
iii. Practical case 1 : image classification
iv. Practical case 2 : 2D regression of bead positions
v. Practical case 3 : Semantic segmentation of bacteria

Deep Learning applied to **semantic segmentation**

Sara Rombouts

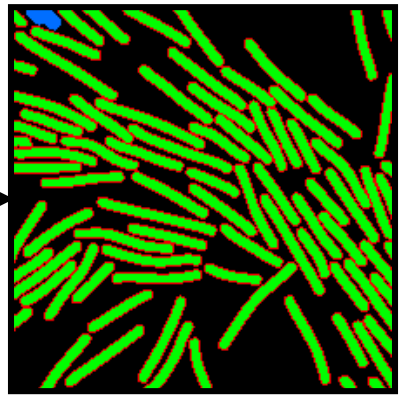
Marcelo Nöllmann

JB Fiche

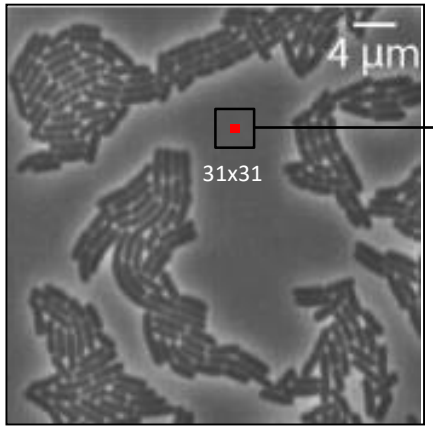


Bright field image of Myxococcus xanthus cells on an agard pad

Segmentation

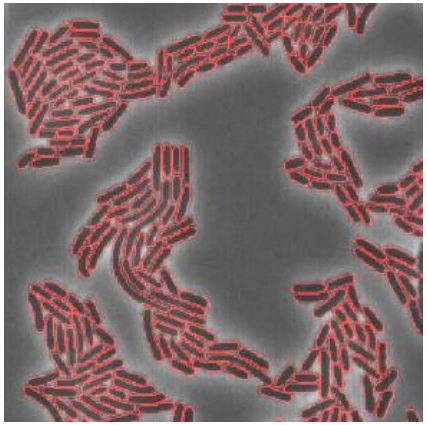


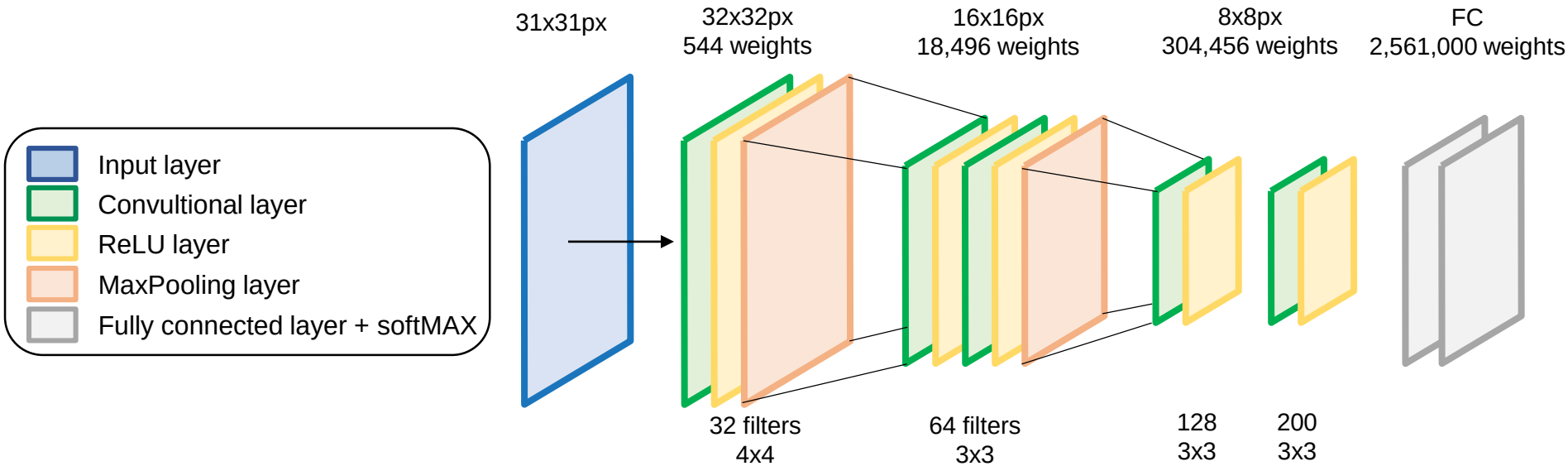
ConvNet applied to cell segmentation



convNet

- Cell boundary
- Cell interior
- Background

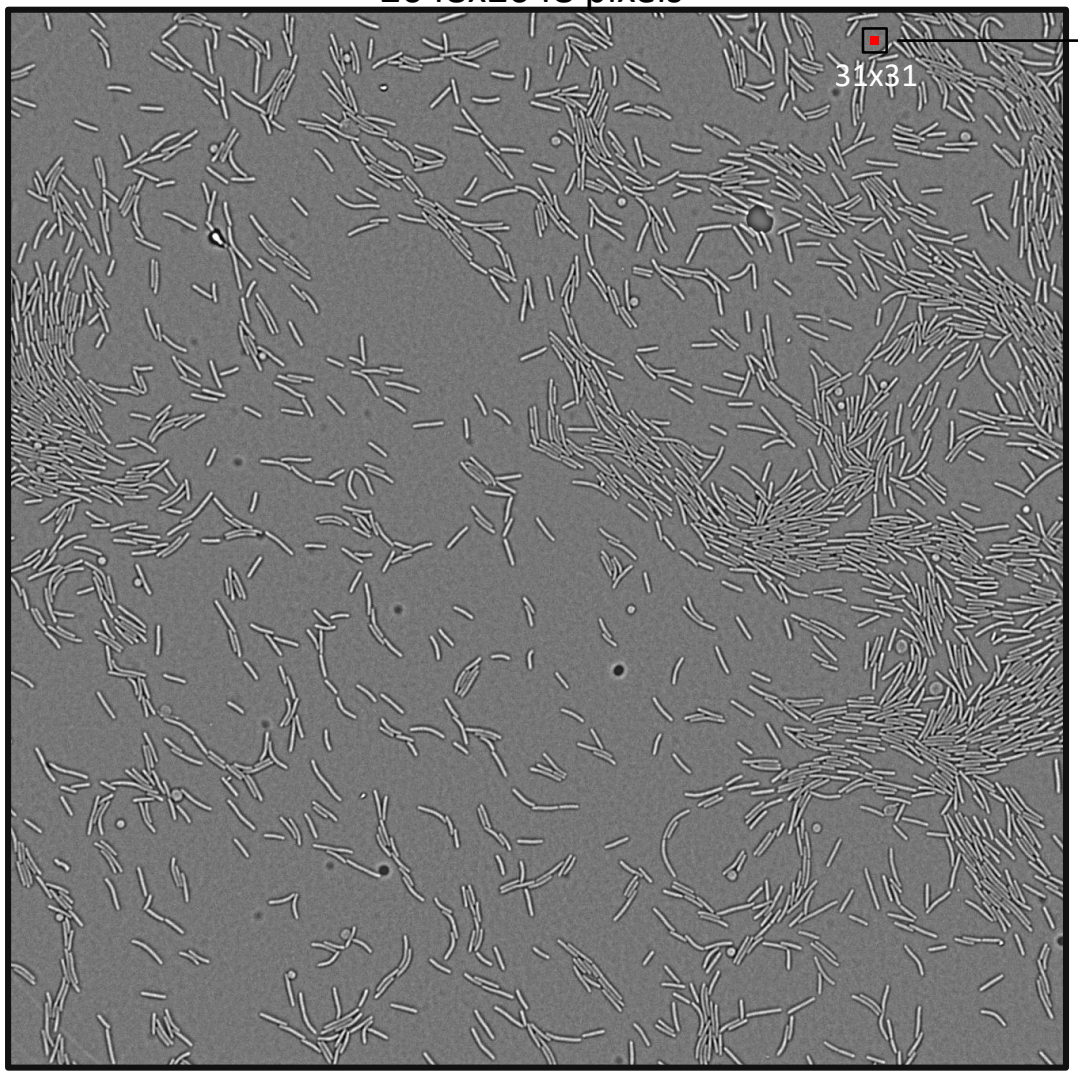




Van Valen et al. 2016. Plos Computational Biology

Main drawback of convNet : the speed

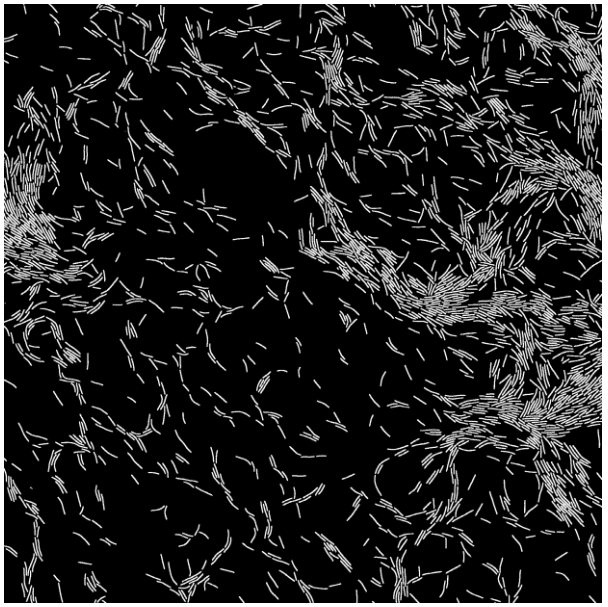
2048x2048 pixels



convNet → classification

- Cell boundary
- Cell interior
- Background

>4M

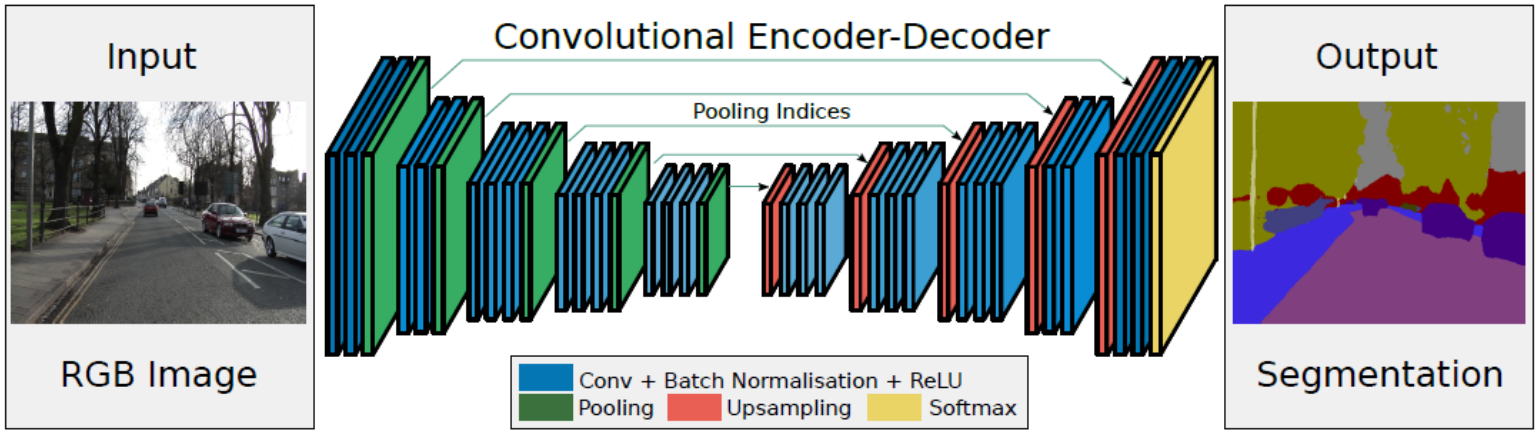


~15 minutes to segment a single image

>500

~5 days to analyze an experiment…

Fully convolutional neural network : FCNN - SegNet

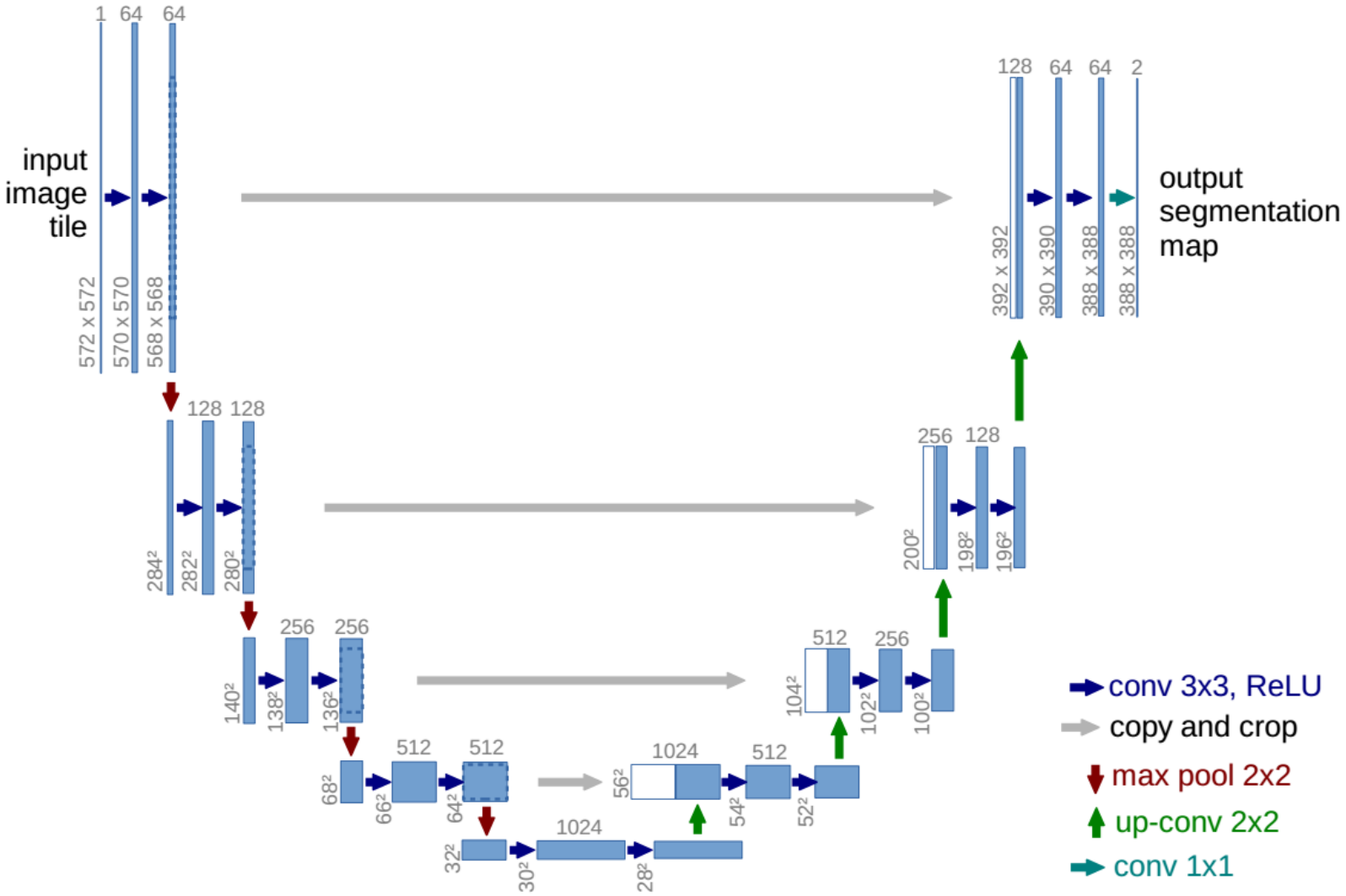


This type of neural network architecture has many advantages:

- Simple to implement
- **Faster**
- **Less heavy** on memory since no fully connected layers
- **No restriction** on the size of the input images
- The **output image is already segmented**

Badrinarayanan et al. IEEE 2016 – Noh et al. ArXiv 2015 – Ronnerberger et al. ArXiv 2015

Fully convolutional neural network : FCNN - UNet

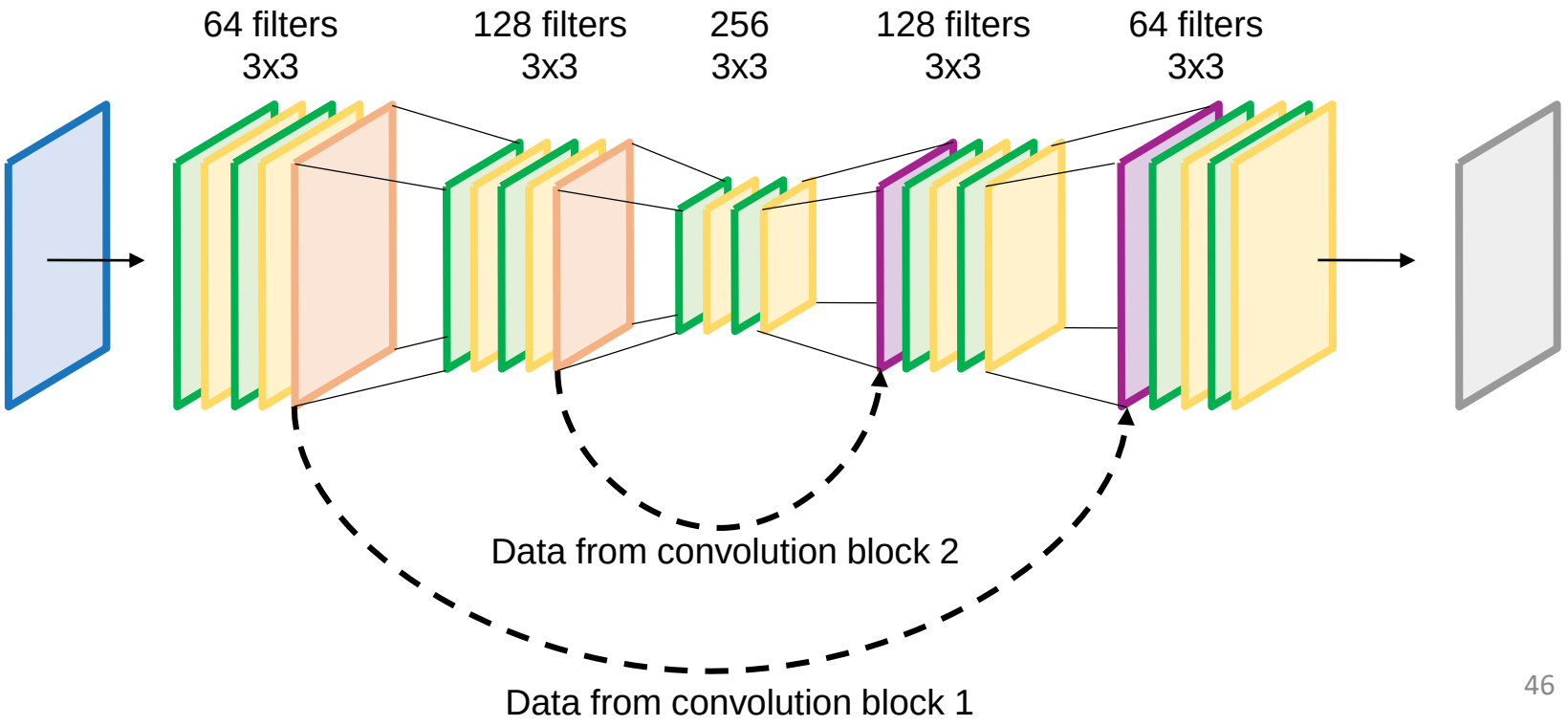


Badrinarayanan et al. IEEE 2016 – Noh et al. ArXiv 2015 – Ronnerberger et al. ArXiv 2015

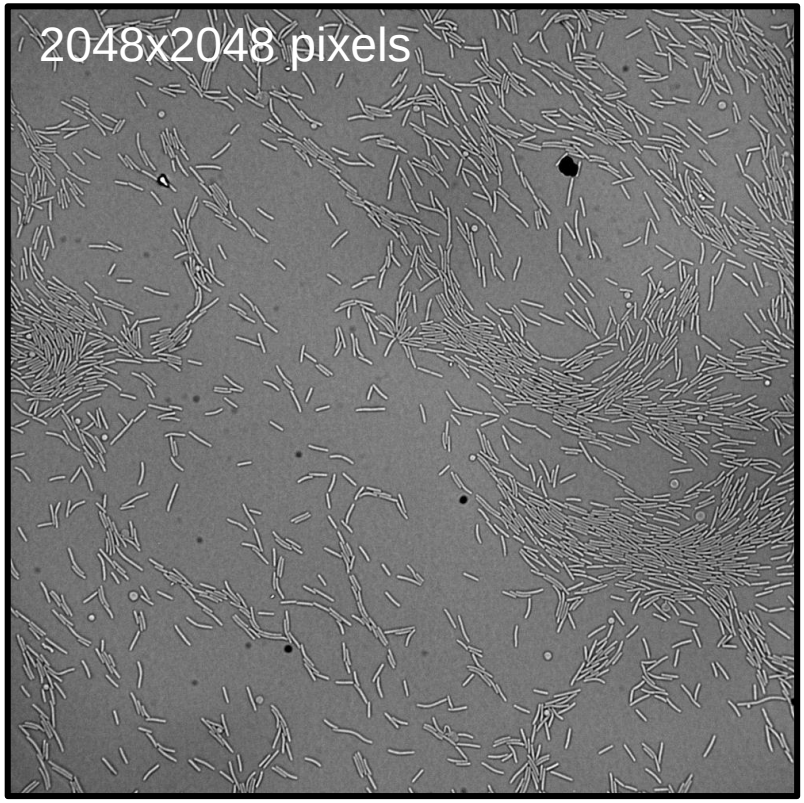
Petit UNet …

- Input raw image(s)
- Convultional layer
- ReLU layer + batch normalization
- MaxPooling layer
- Deconvolution layer
- Output segmented image

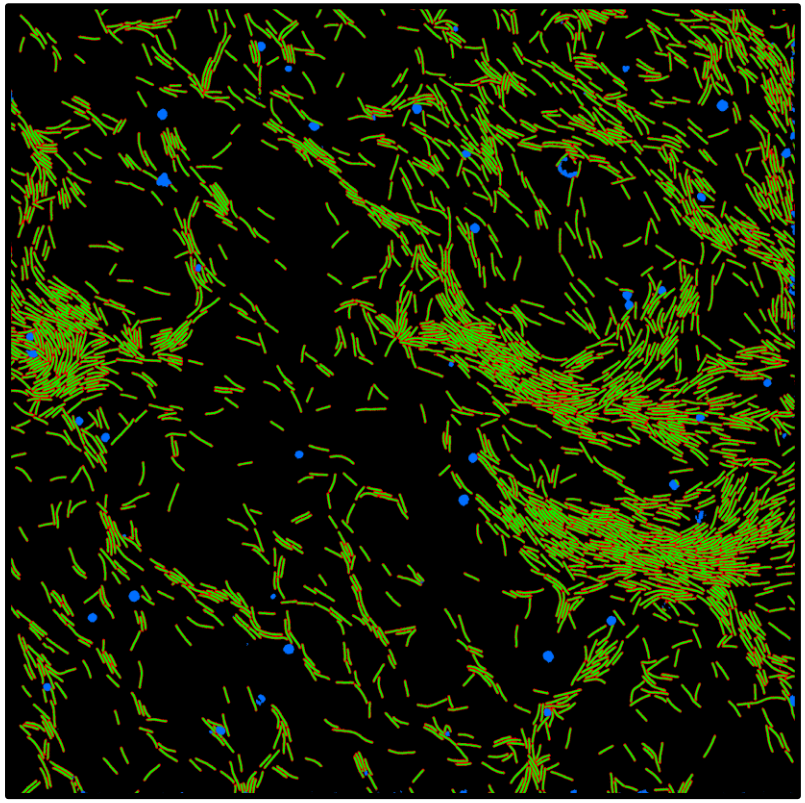
Total number of learnable : **1,864,969**



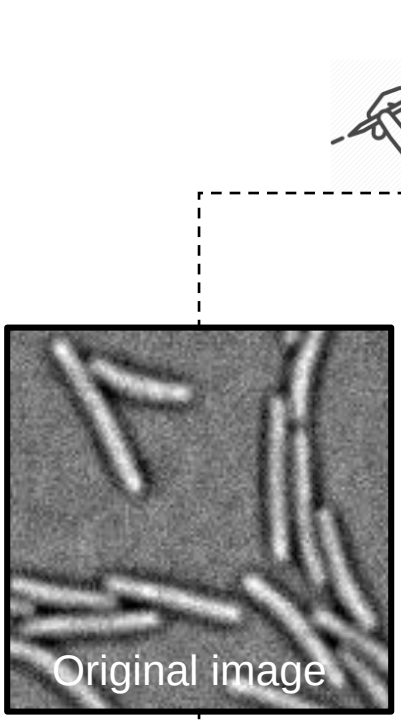
… mais costaud !

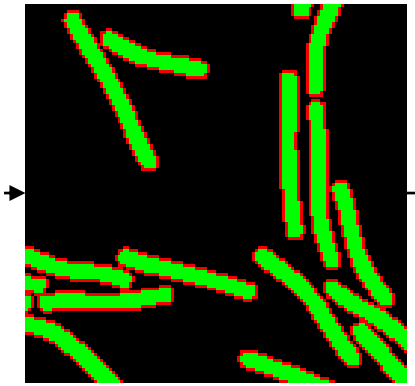


UNet
~4s

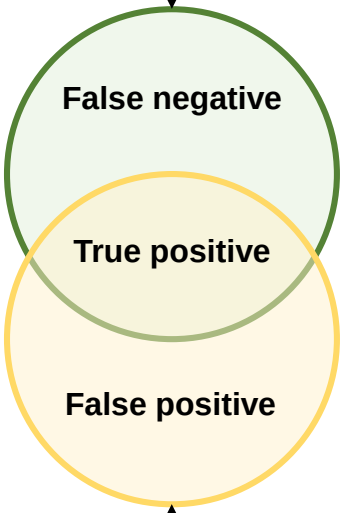


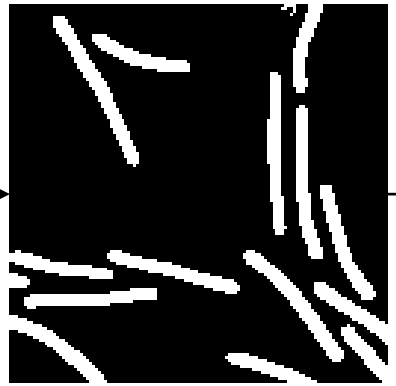
How to evaluate the quality of the prediction?





Ground truth





Model prediction

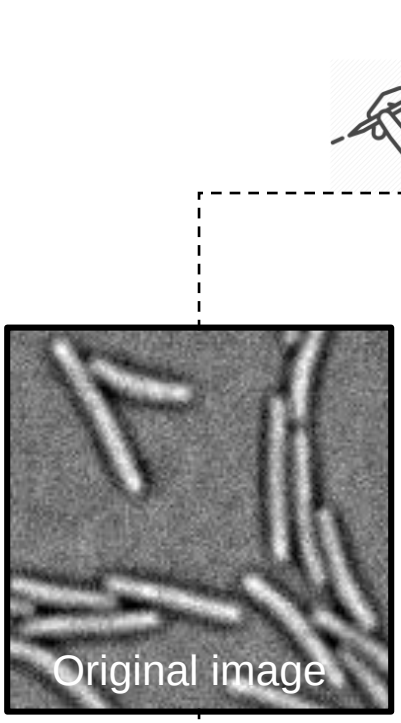
Accuracy for each class:

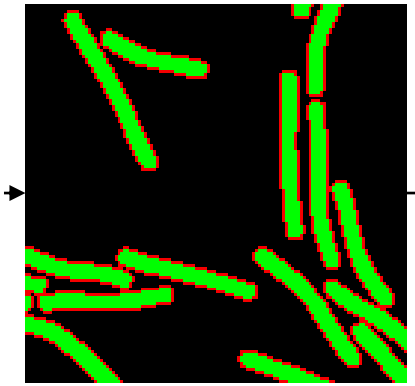
$$\frac{TP}{TP + FN}$$

Cell : 87%
Contour : 92%

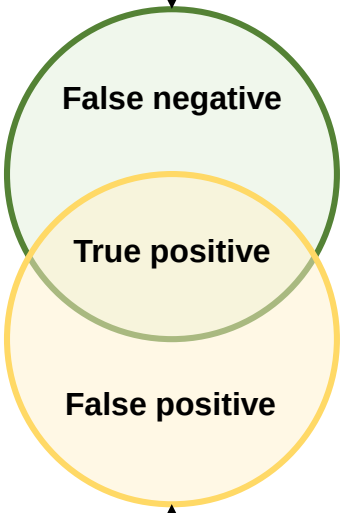
Good results **even if the accuracy is <90%**

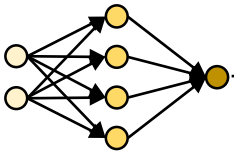
How to evaluate the quality of the prediction?

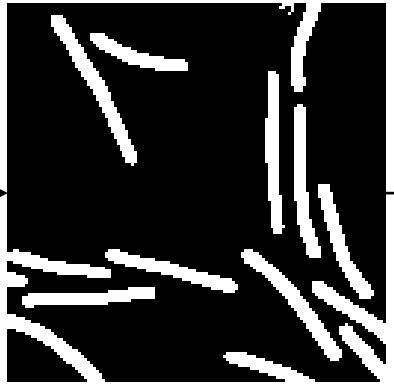




Ground truth







Model prediction

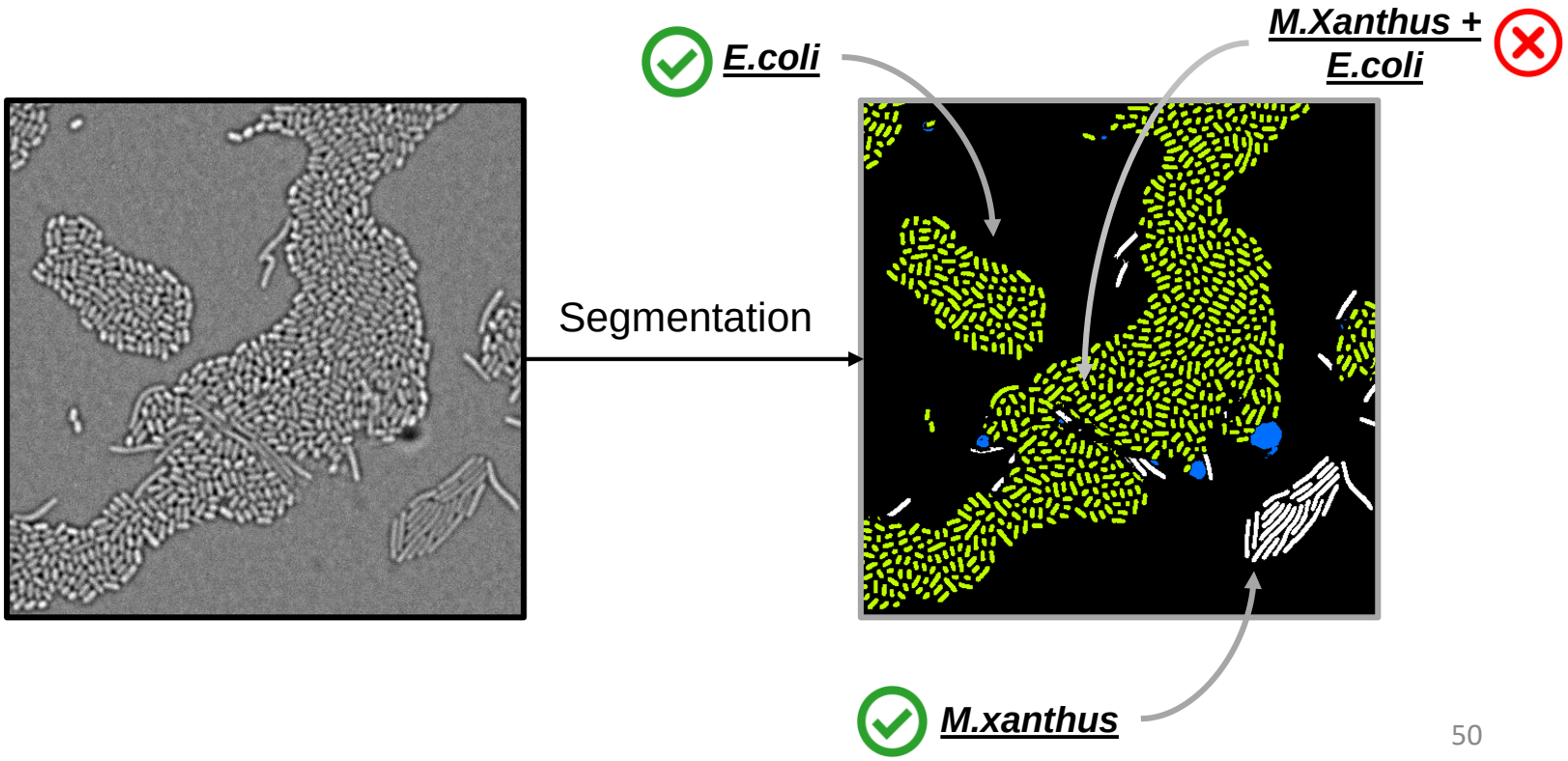
IoU or Jaccard similarity:

$$\frac{TP}{TP + FN + FP}$$

Cell : 85%
Contour : 90%

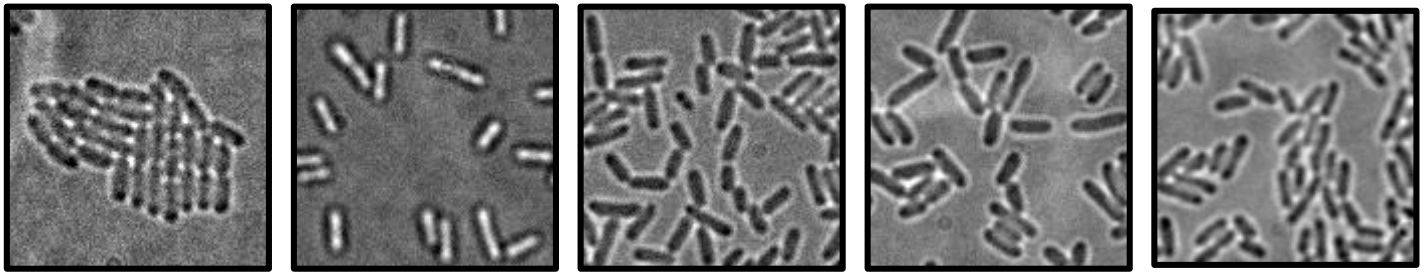
One last VERY important point

1. Your images should be normalized:
   - Average intensity equals to 0
   - Standard deviation equals to 1
2. Your set of images should be **as representative and diverse as possible**



One last VERY important point

1. Your images should be normalized:
   - Average intensity equals to 0
   - Standard deviation equals to 1
2. Your set of images should be **as representative and diverse as possible**



Variation in :

- Density of cells
- Different focal planes
- Light intensity
- etc.

→ **Good generalization!**

Outline

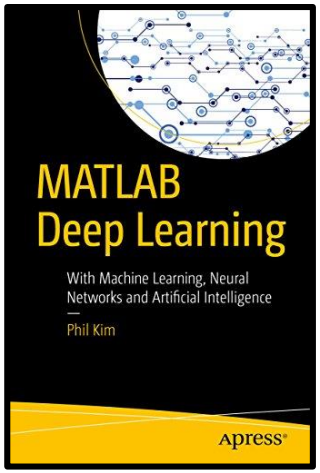
I. Introduction
II. Starting with Deep Learning and Artificial Neuron Network
III. Deep Learning for image analysis

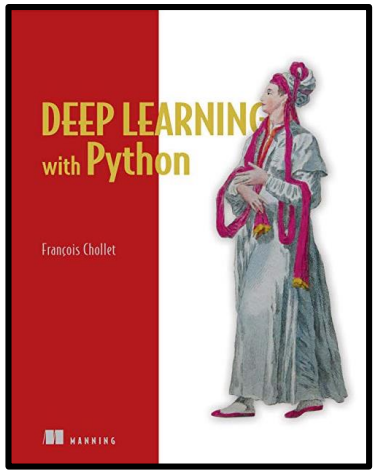
IV. Conclusion…

Activation Function
generalization
overfitting
gradient descent
ground truth
hyperparameter
matplotlib
neuron
one-hot encoding
learning rate
Backpropagation
Categorical Cross-Entropy
CNN, ConvNet
Dropout
Keras
Max-Pooling
MNIST
Momentum
Nonlinearity
ReLU
SGD
Softmax
TensorFlow
Vanishing Gradient Problem
VGG16
data augmentation
transfer learning
epoch
weights
bias
hidden layer
batch size
classification
regression
convolution
loss function
testing set
validation set
training set
deep
optimizer
fully connected layer

Bibliography & useful links

- F. Chollet – 2018 – Deep Learning with Python
- P. Kim – 2017 – Matlab Deep Learning
- I. Goodfellow, Y. Bengio, A. Courville – 2016 – Deep Learning





- Stanford lectures on Deep Learning and Machine Learning : https://cs230.stanford.edu/lecture/

- Inaugural lecture of Y. Lecun at collège de France : https://www.college-de-france.fr/site/yann-lecun/inaugural-lecture-2016-02-04-18h00.htm

- Awesome website for math : https://www.3blue1brown.com/