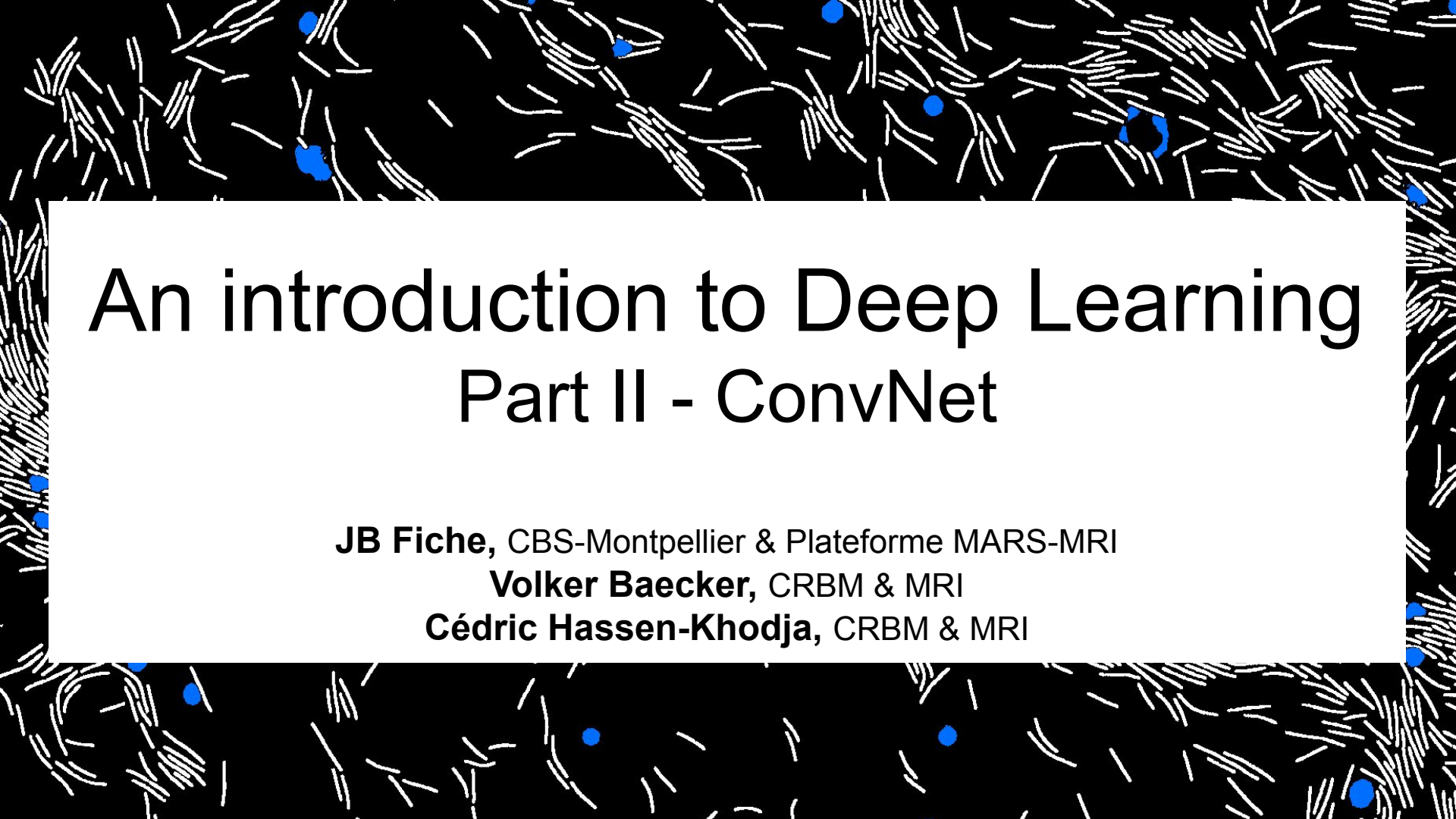


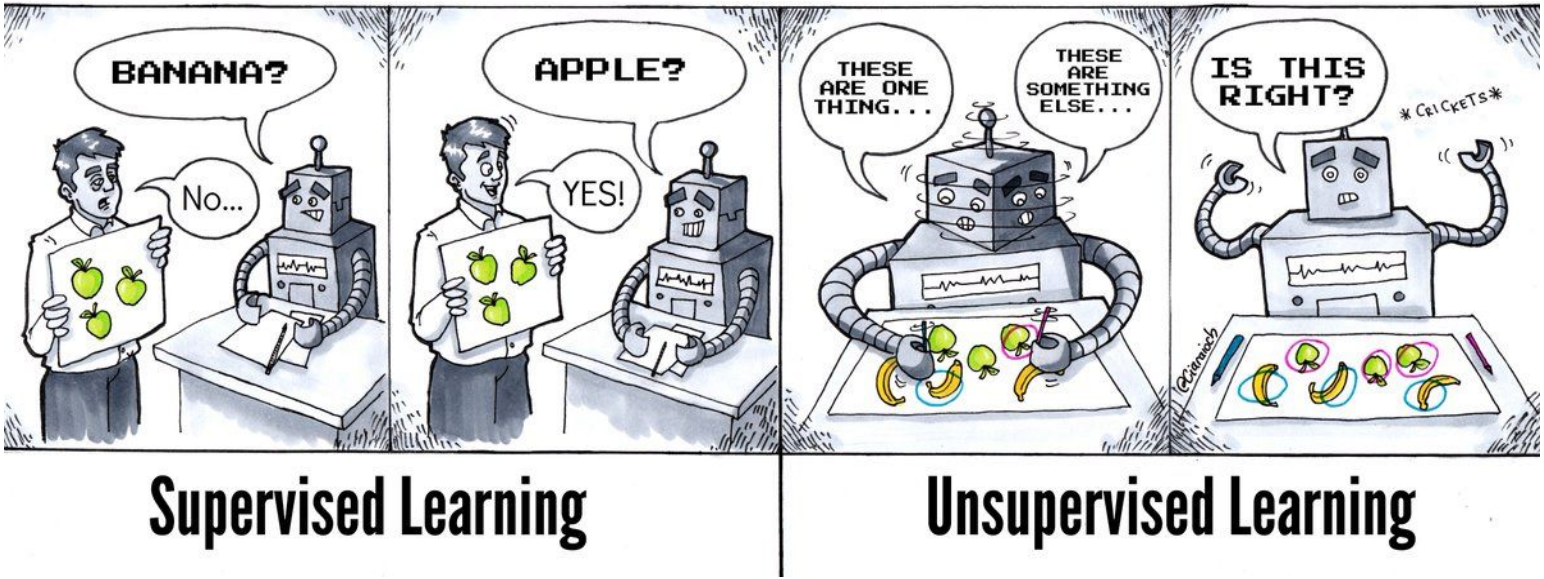
An introduction to Deep Learning

Part II - ConvNet

JB Fiche, CBS-Montpellier & Plateforme MARS-MRI
Volker Baecker, CRBM & MRI
Cédric Hassen-Khodja, CRBM & MRI

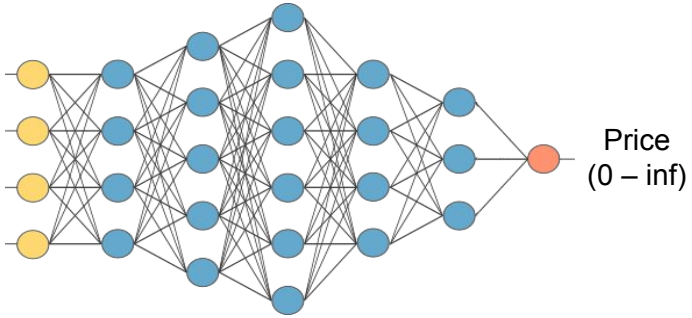
Quick reminder of the key concepts

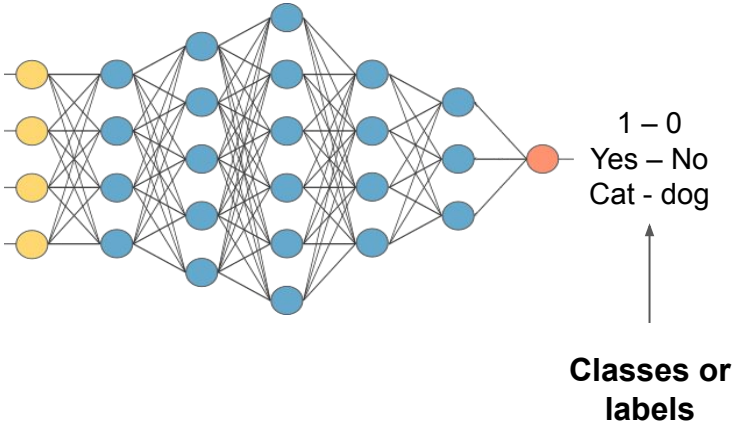
- **Supervised learning** = the data are coming with already labelled data (**ground truth**)



Quick reminder of the key concepts

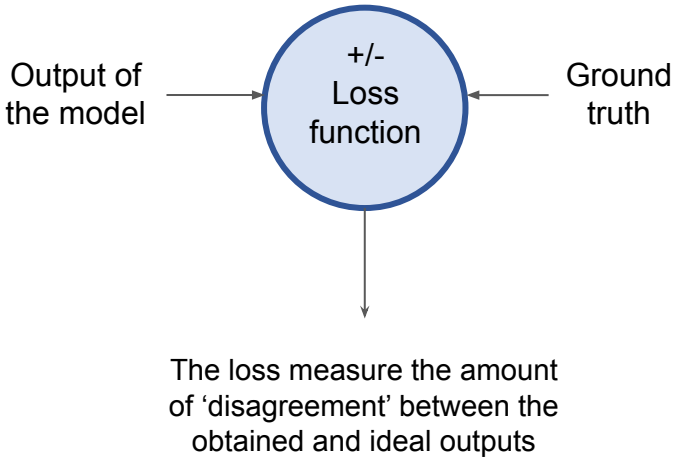
- Supervised learning
- **Classification and regression**





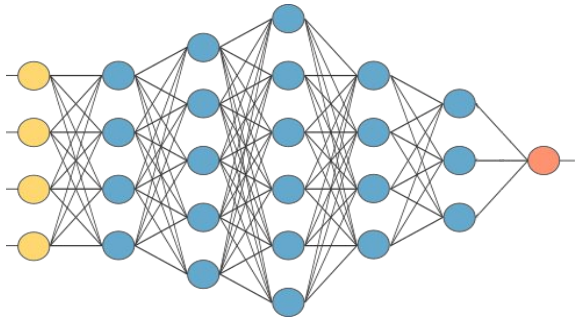
Quick reminder of the key concepts

- Supervised learning
- Classification and regression
- **Loss function**



Quick reminder of the key concepts

- Supervised learning
- Classification and regression
- Loss function
- Parameters vs. hyperparameters

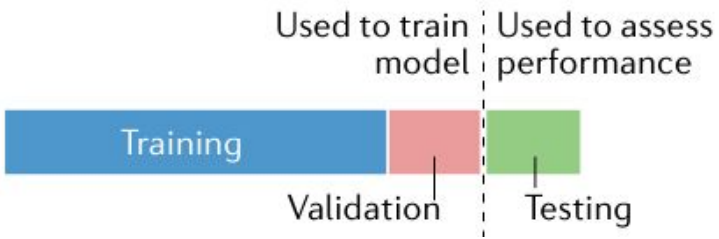


parameters : weight and bias of a trained model

hyperparameters : all the parameters that could influence the learning process (number of hidden layers, loss function, optimizer, number of epochs, batch size, etc.)

Quick reminder of the key concepts

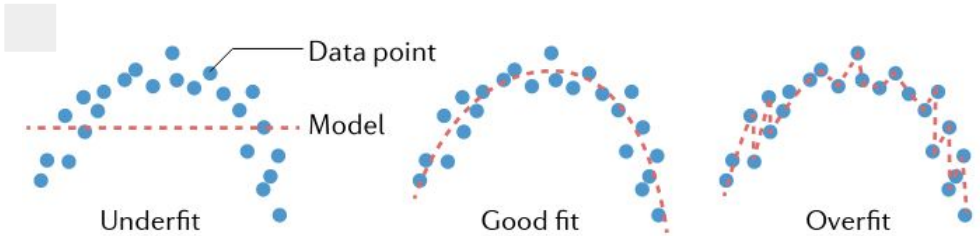
- Supervised learning
- Classification and regression
- Loss function
- Parameters vs. hyperparameters
- Training, testing and validation

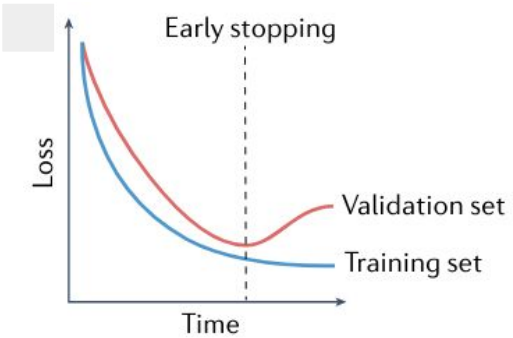


Greener et al. - 2021 A guide to machine learning for biologists

Quick reminder of the key concepts

- Supervised learning
- Classification and regression
- Loss function
- Parameters vs. hyperparameters
- Training, testing and validation
- Under / overfitting

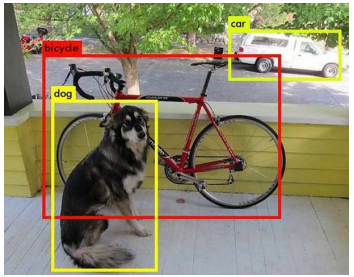




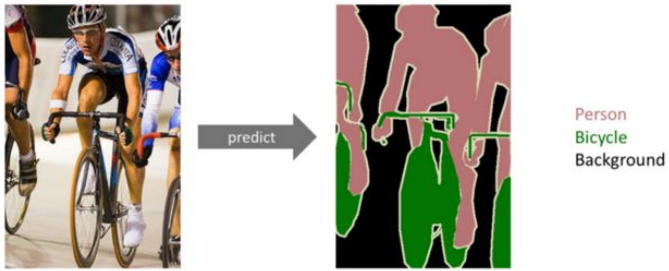
Greener et al. - 2021 A guide to machine learning for biologists

Computer Vision - possible tasks

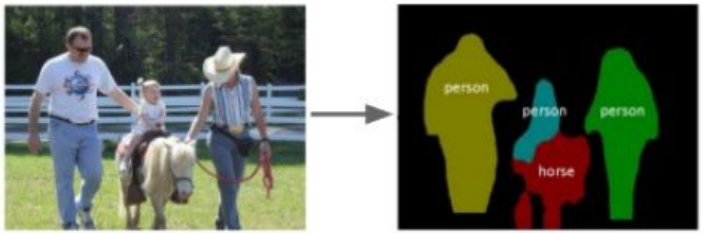
classification
“cat”



classification + localization
class + bounding box



Semantic segmentation
each pixel : class



Instance segmentation for each pixel:
class “person” + instance #1
class “person” + instance #2 ...

Outline

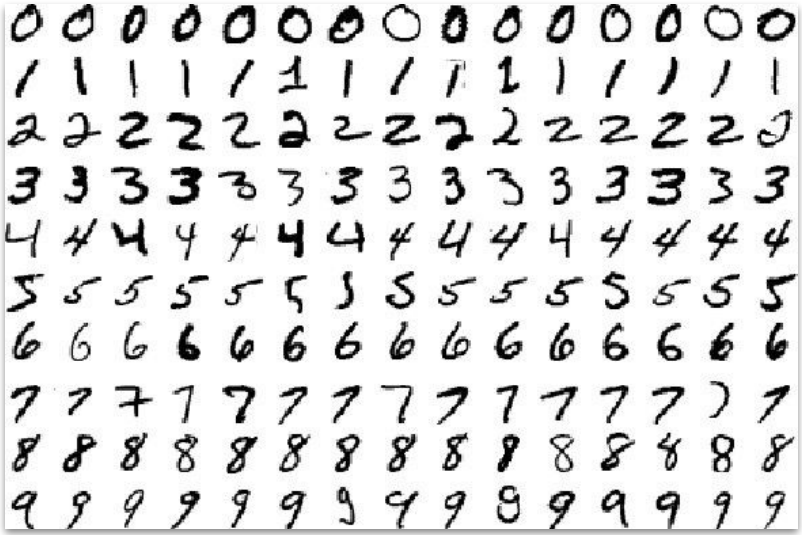
I. Image classification :

   A. Introduction to convolutional network : digit classification
   B. Application to red-blood cells classification
   C. Introduction to transfer-learning

II. Segmentation :
   A. Application of a fully convolutional network (Unet)
   B. Instance segmentation with existing tools

III. Conclusion :

Working with the MNIST database



Large database of **handwritten digits** that is commonly used for machine learning.

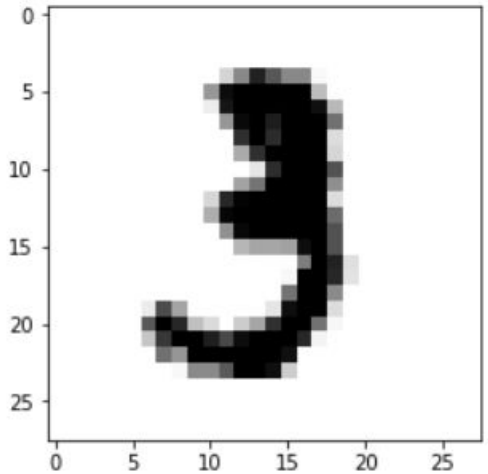
It contains :

- 60.000 images for training
- 10.000 images for testing/validation

Ex4_MNIST_dense_vs_convolutional_nn.ipynb

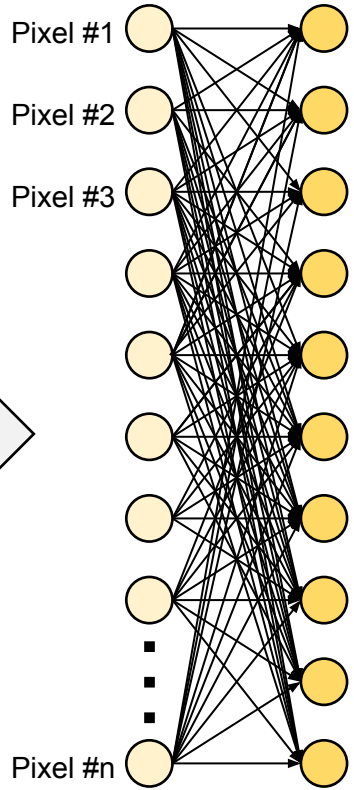
https://en.wikipedia.org/wiki/MNIST_database

Image classification with a dense network



28x28 pixels image

784 inputs



First layer 10 neurons

Number of parameters:

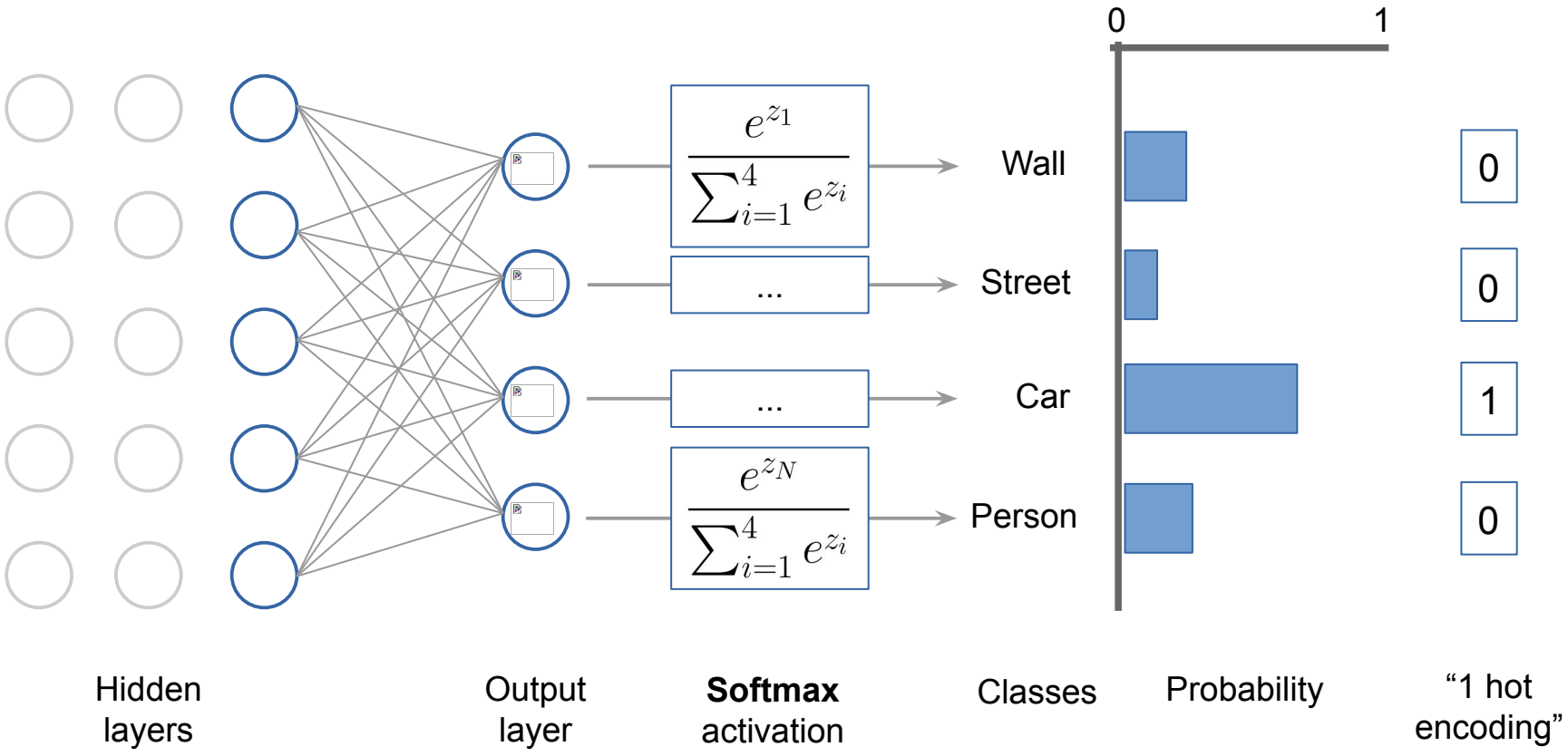
784*10+10 = 7850

How to choose the activation and loss functions ?

Problem type	Last-layer activation	Loss function	Number of neurons in the last layer
Binary classification	‘sigmoid’	‘binary-crossentropy’	1
Multiclass, single-label classification	‘softmax’	‘categorical_crossentropy’	As many as the number of classes
Regression to values between 0 and 1	‘sigmoid’ or ‘none’	‘mse’	1

As a rule-of-thumbs, use ‘relu’ everywhere else as activation function.

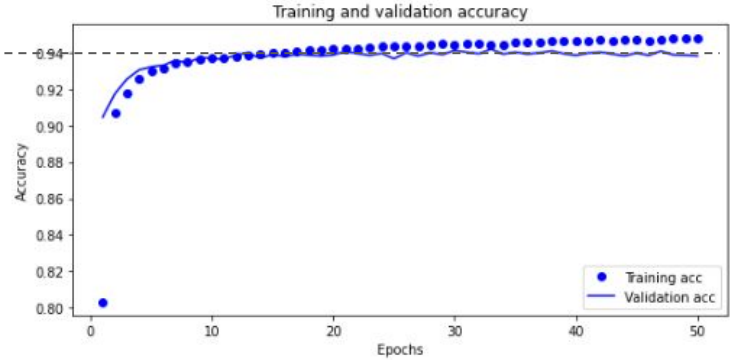
Softmax activation

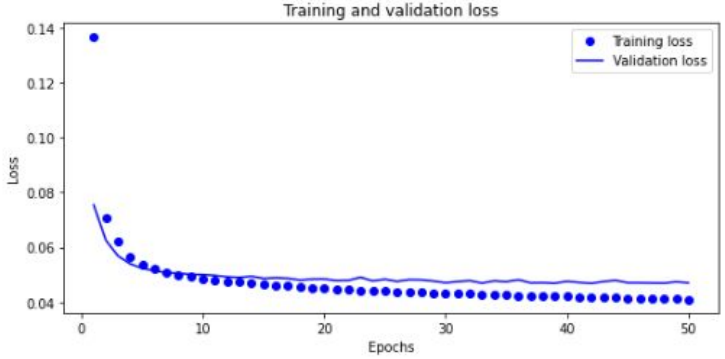


MNIST classification with a dense network

2 layers of 10 neurons : **7960 parameters**

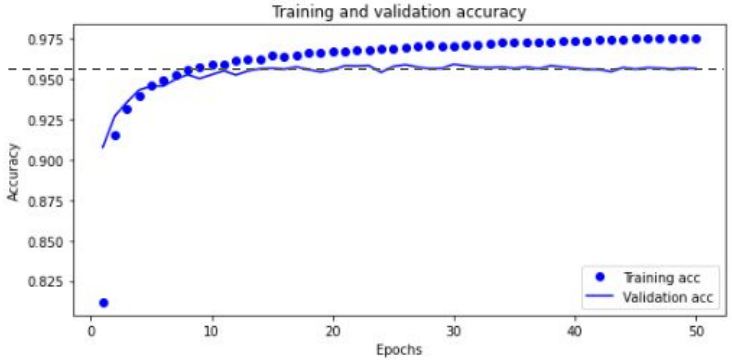
94%

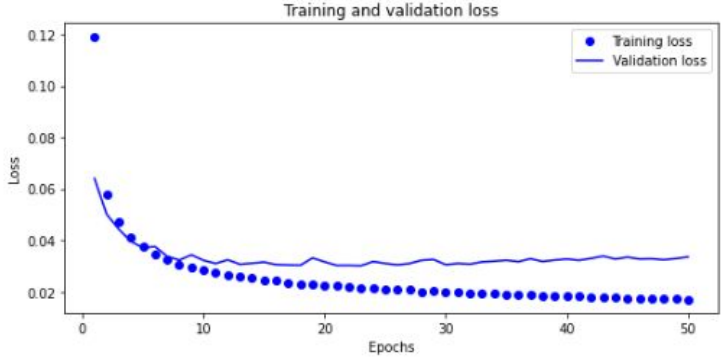




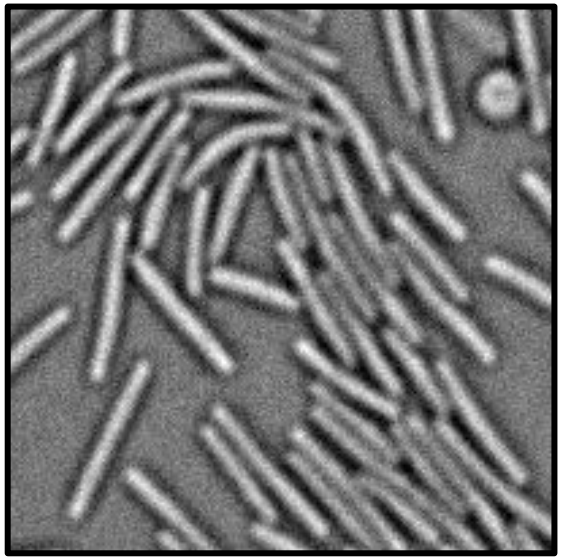
3 layers of 15 neurons : **12175 parameters**

>95%



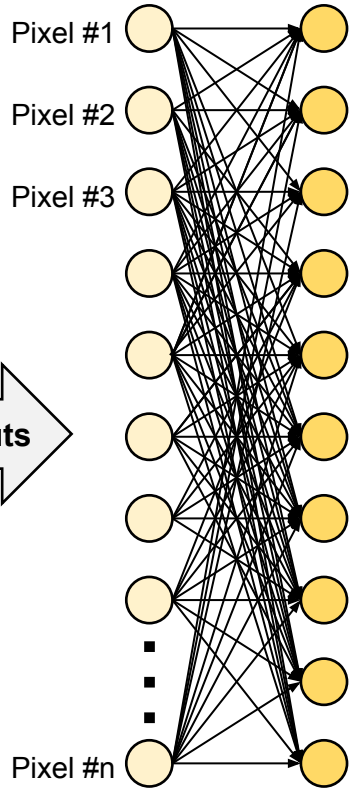


Dense network for large images?



200x200 pixels image

40.000 inputs



First layer 10 neurons

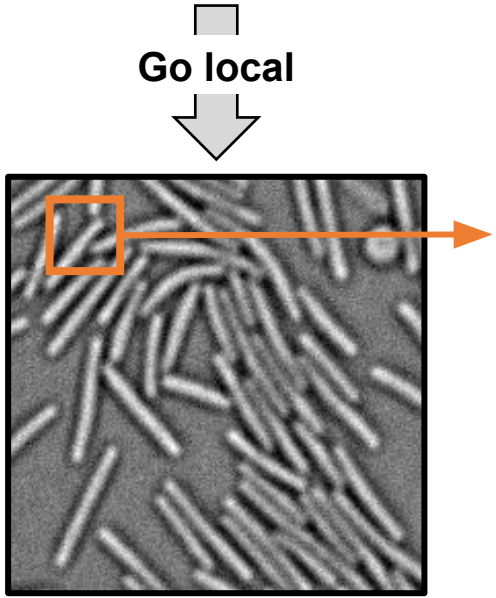
Number of parameters:

40.000*10+10 = 400.010 (!!!)

Image analysis with convolutional network

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

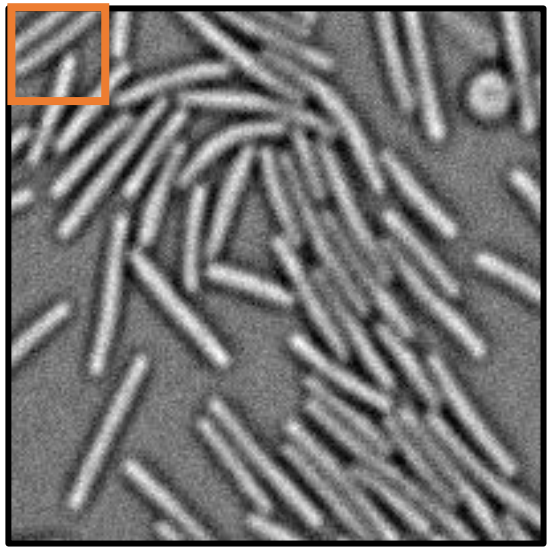
- Too many parameters, even for small images
- Loses the local information around each pixel
- The network architecture depends on the image size



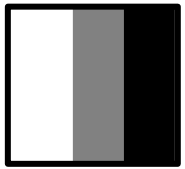
Learn local representations with **convolutional** network.

What is convolution?

Input image



Kernel / convolution matrix / filter

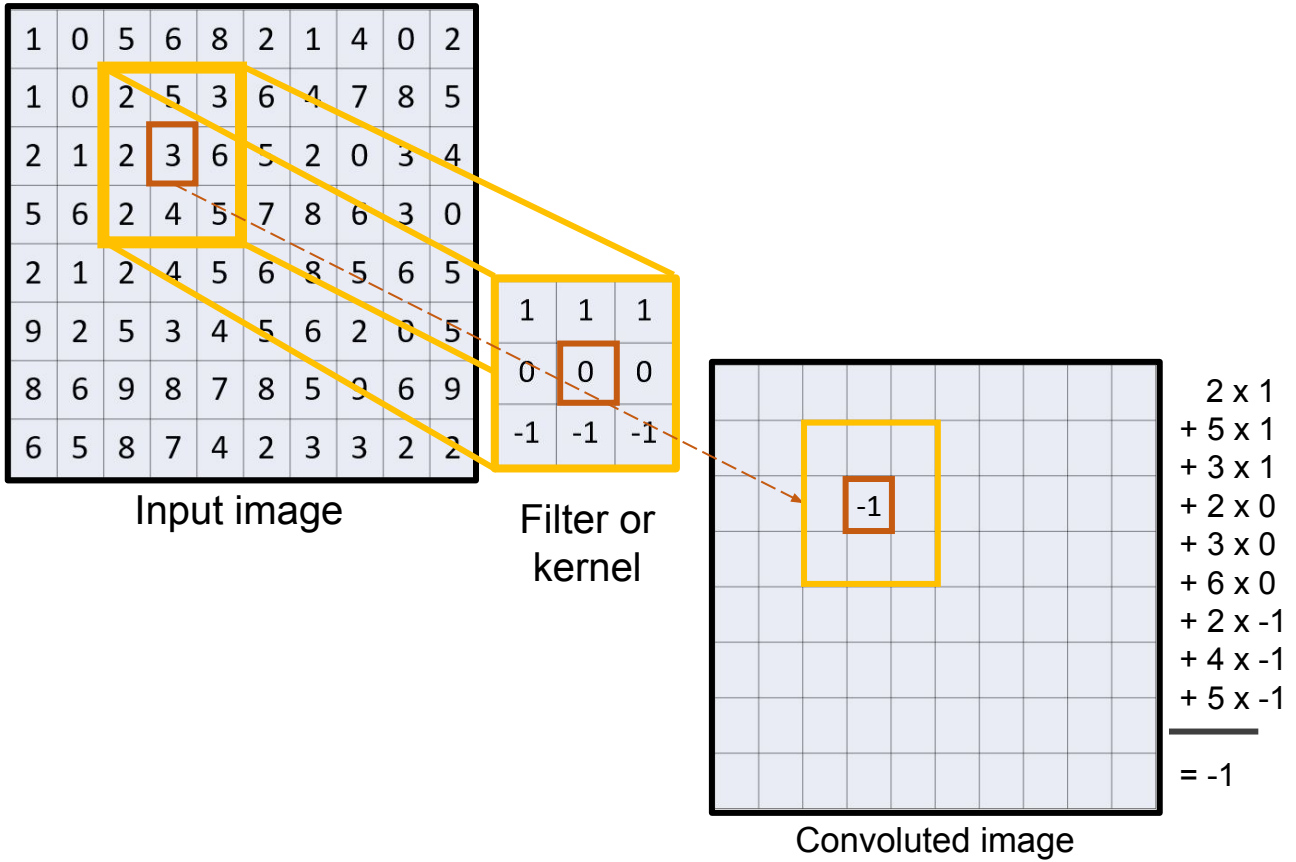


3x3

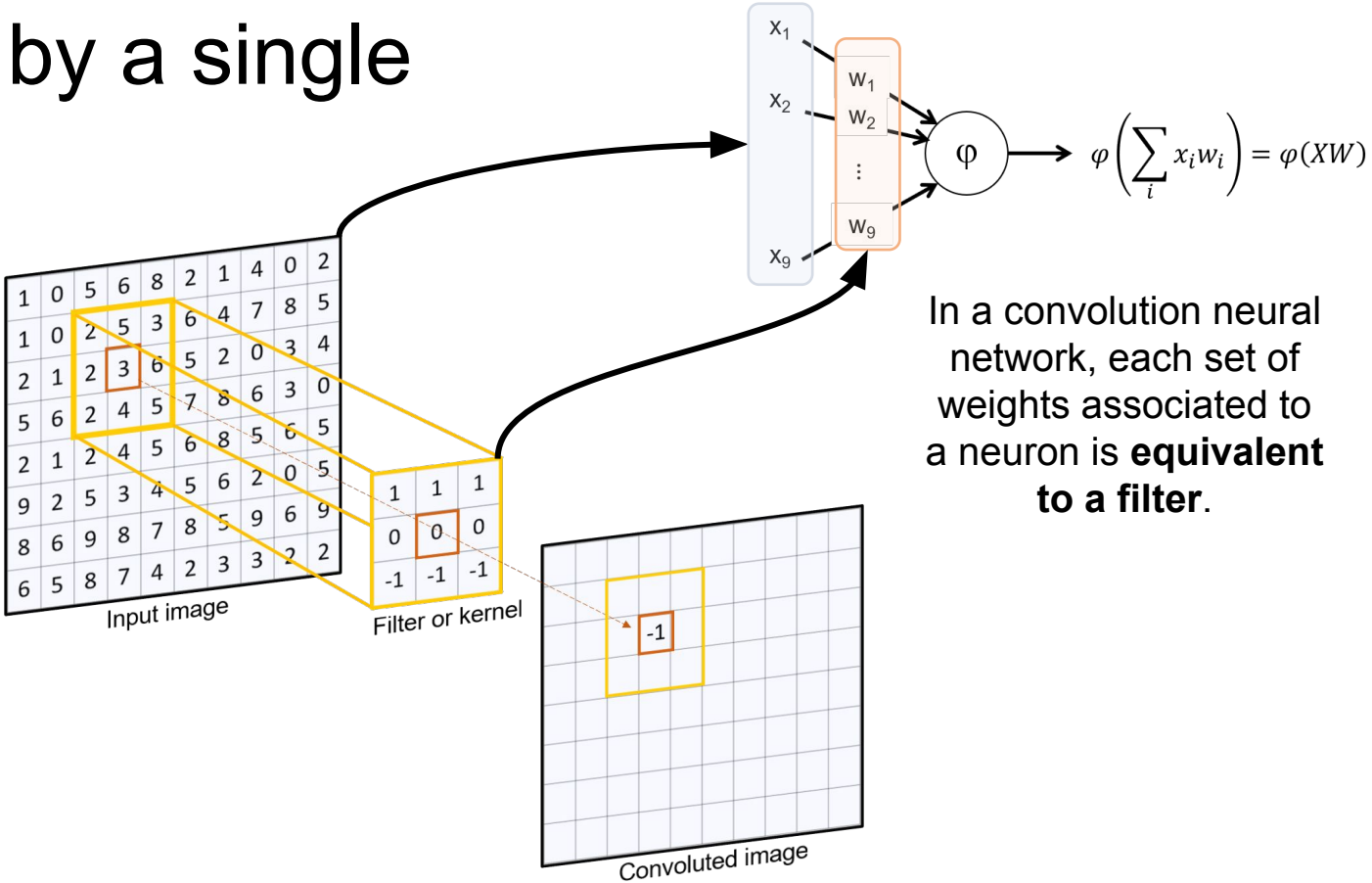
Convolution

Output image : highlight **vertically oriented objects**

Convolution operation in one scheme

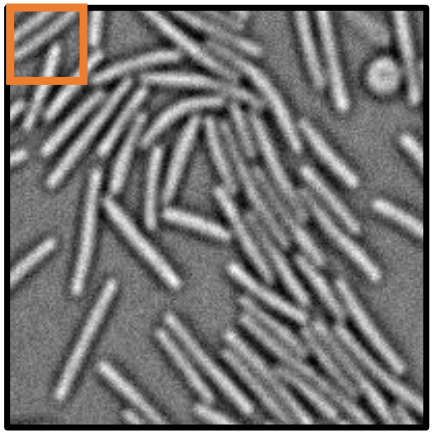


Convolution operation performed by a single neuron

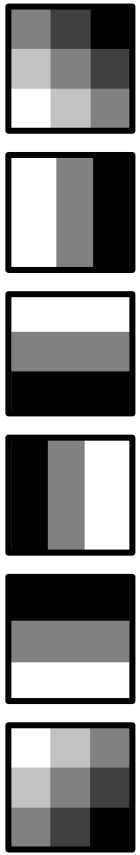


In a convolution neural network, each set of weights associated to a neuron is **equivalent to a filter**.

Features map

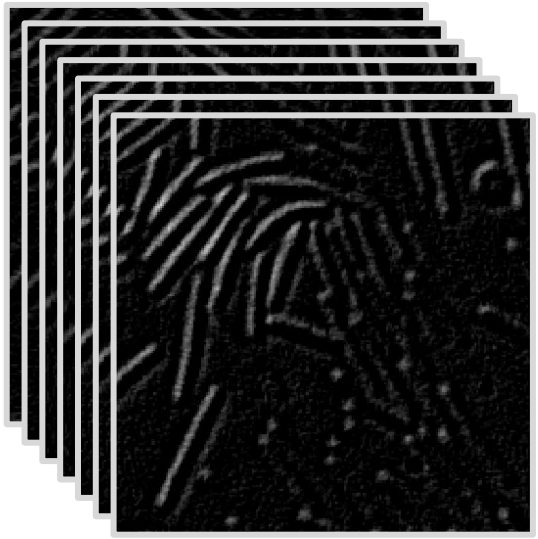


$\otimes$



Convolution
& ReLu

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



Features maps

ConvNet syntax

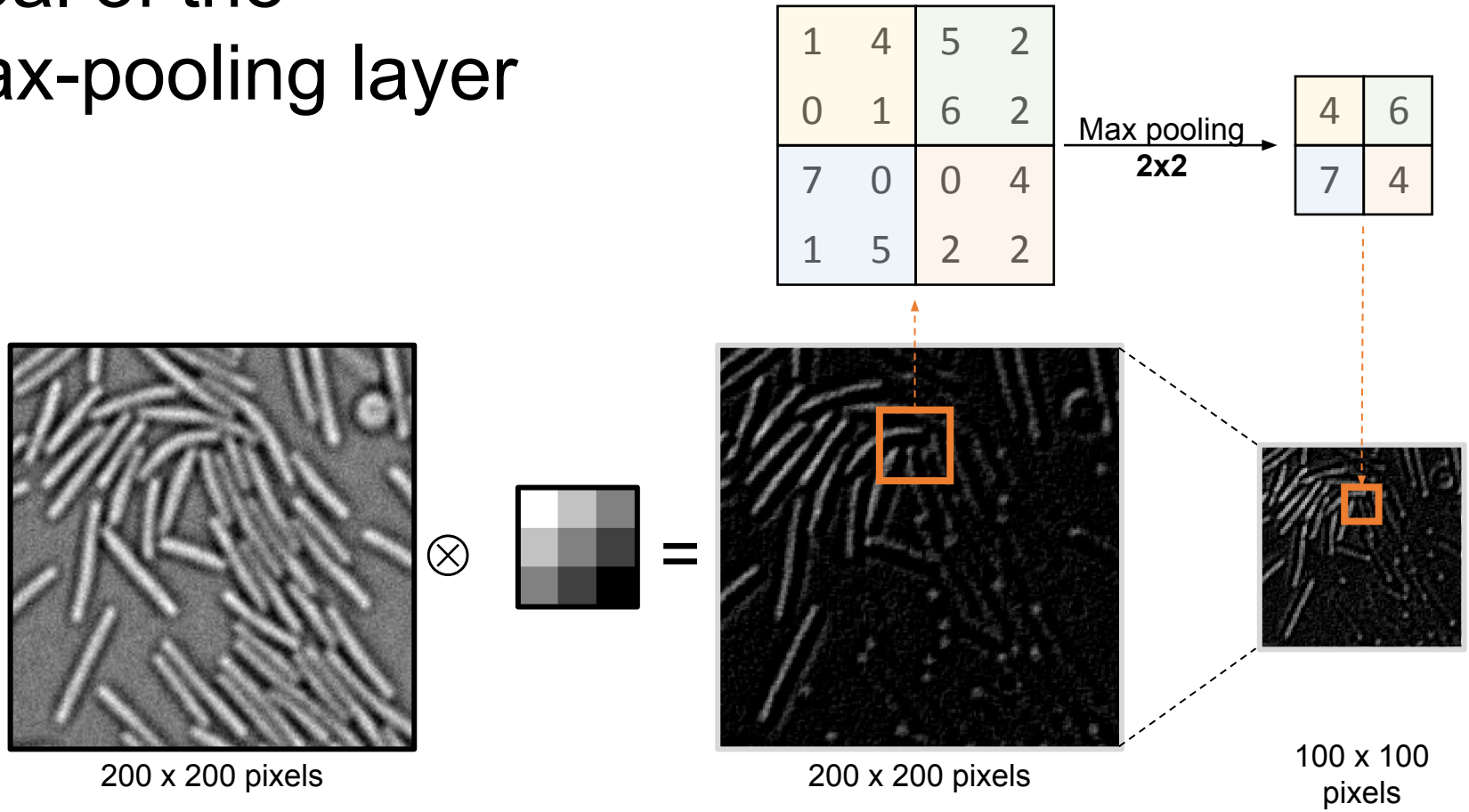
Convolution part

Dense part

```
modelCNN = Sequential([

    # Convolution Layer 1
    Conv2D(16, (3, 3), activation='relu', input_shape=(28, 28, 1)), # 16 different 3x3 kernels -- so 32 feature maps
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2 kernel

    # Convolution Layer 2
    Conv2D(16, (3, 3), activation='relu'), # 16 different 3x3 kernels
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(16, (3, 3), activation='relu'), # 16 different 3x3 kernels

    Flatten(), # Flatten final 3x3x16 output matrix into a 144-length vector

    # Fully Connected Layer 4
    Dense(10), # 10 FCN nodes
    Activation('relu'),
    Dense(10), # Necessary for the last layer since we have 10 classes
    Activation('softmax'),
])
```

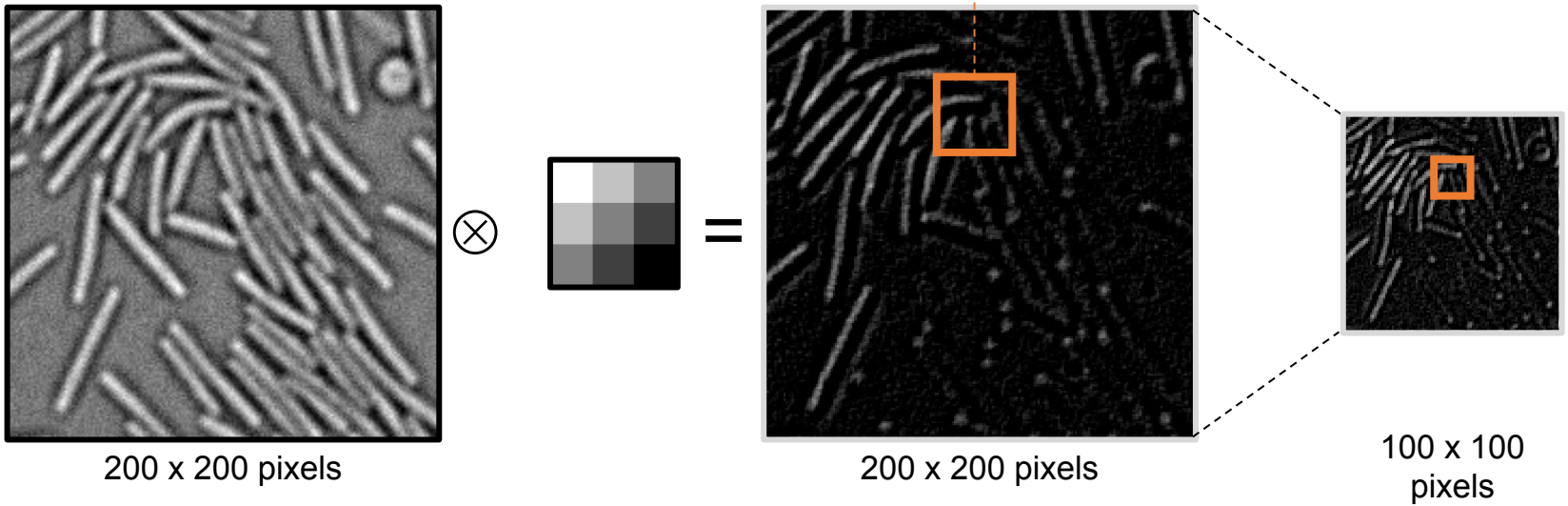
Convolution layer MaxPooling layer Dense layer

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**



MNIST classification with a ConvNet

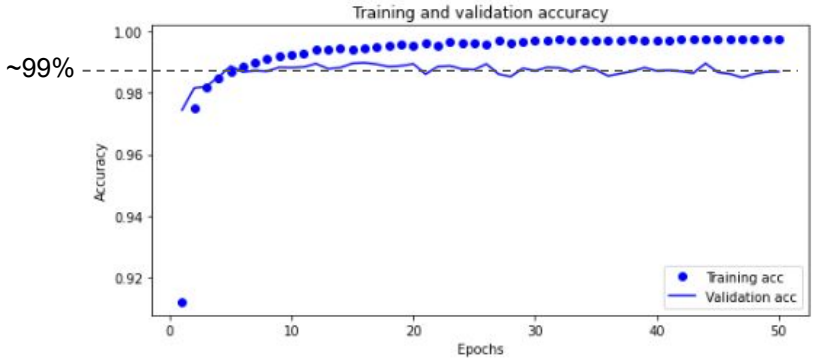
```
modelCNN = Sequential([

    # Convolution Layer 1
    Conv2D(16, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 2
    Conv2D(16, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(16, (3, 3), activation='relu'),

    Flatten(),

    # Fully Connected Layer 4
    Dense(15),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

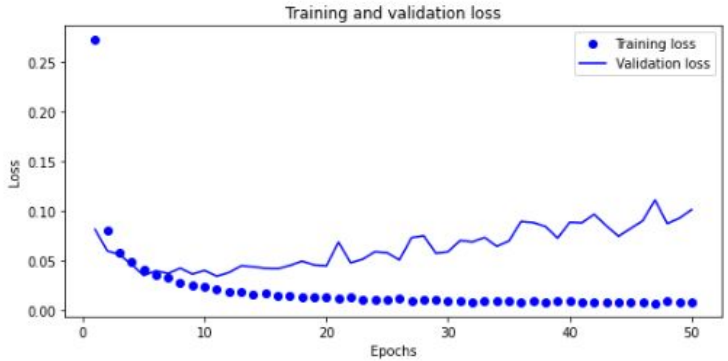
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 11, 11, 16)        2320
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 3, 3, 16)          2320
_________________________________________________________________
flatten_2 (Flatten)          (None, 144)               0
_________________________________________________________________
dense_4 (Dense)              (None, 15)                2175
_________________________________________________________________
activation_4 (Activation)    (None, 15)                0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                160
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 7,135
Trainable params: 7,135
Non-trainable params: 0
```

MNIST classification with a ConvNet

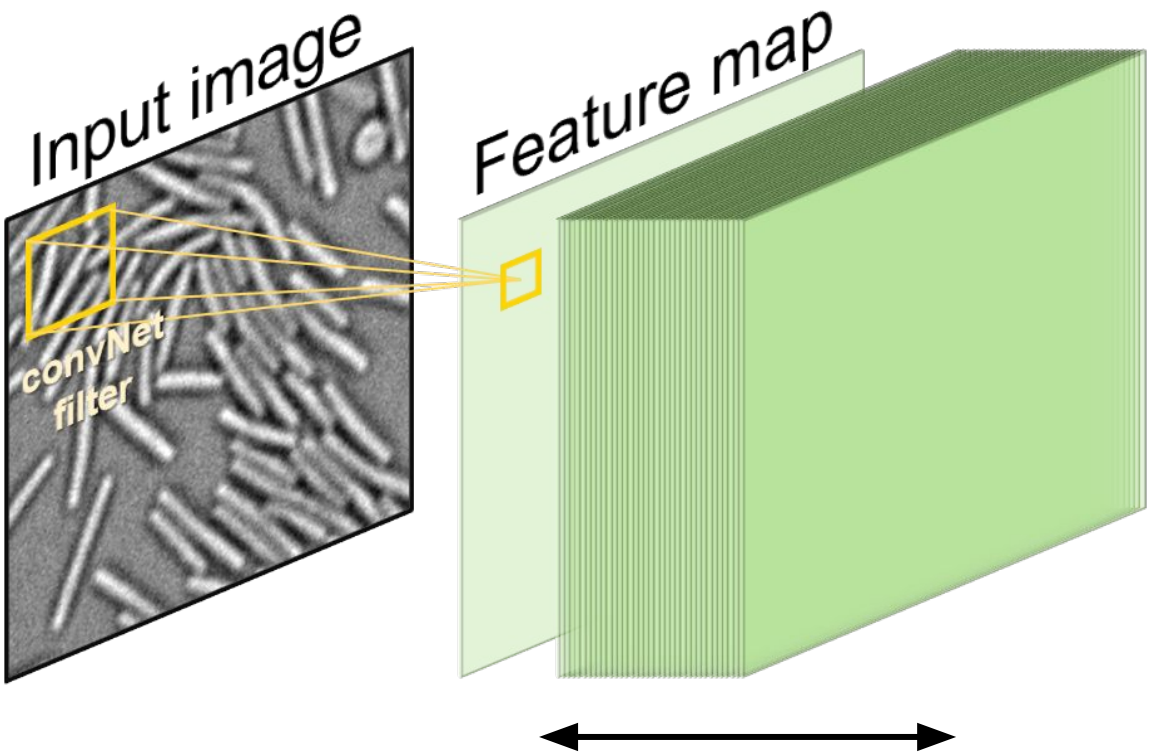
3 convolution layers of 16 kernel and 2 dense layers of 10 neurons : **7135 parameters**

```
modelCNN = Sequential([

    # Convolution Layer 1
    Conv2D(16, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 2
    Conv2D(16, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(16, (3, 3), activation='relu'),

    Flatten(),

    # Fully Connected Layer 4
    Dense(15),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```



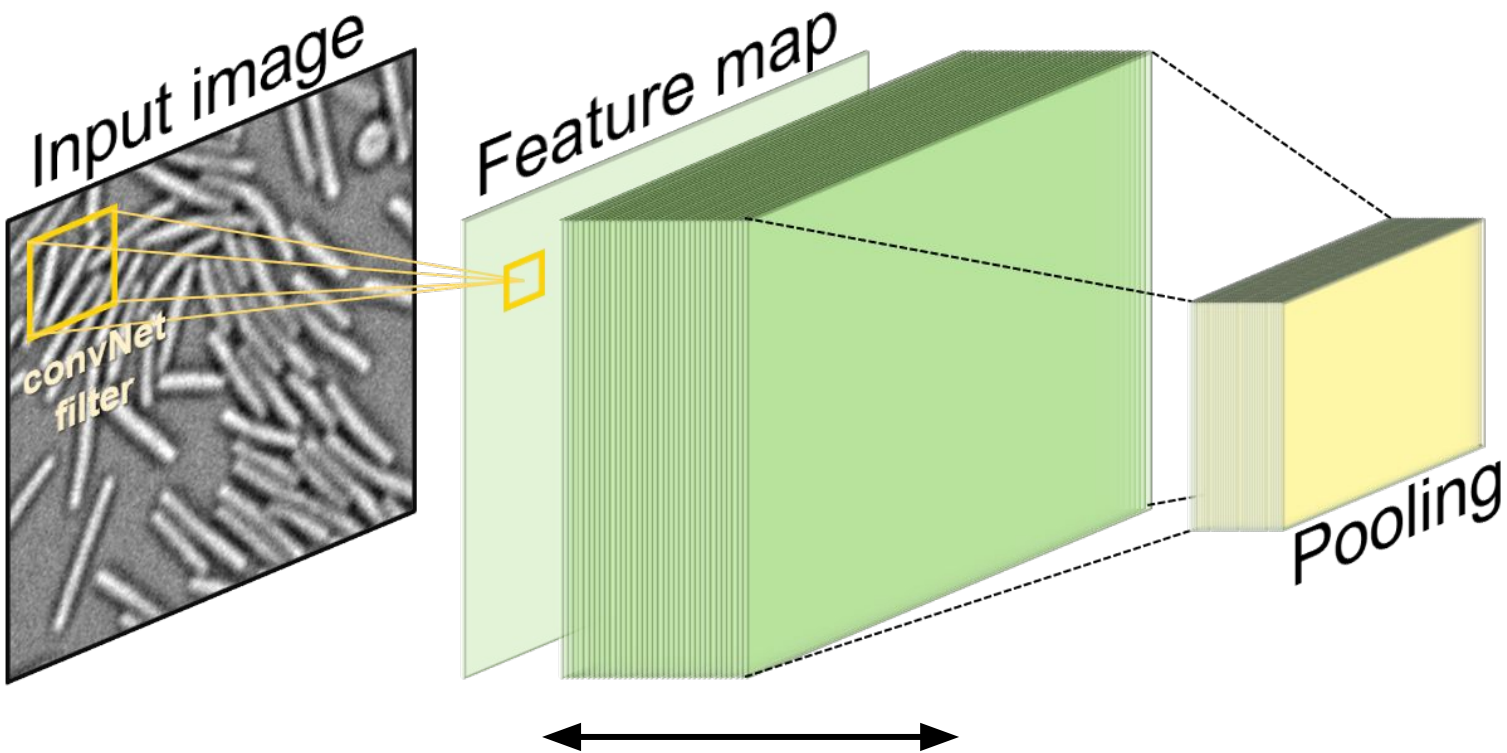


ConvNet summary



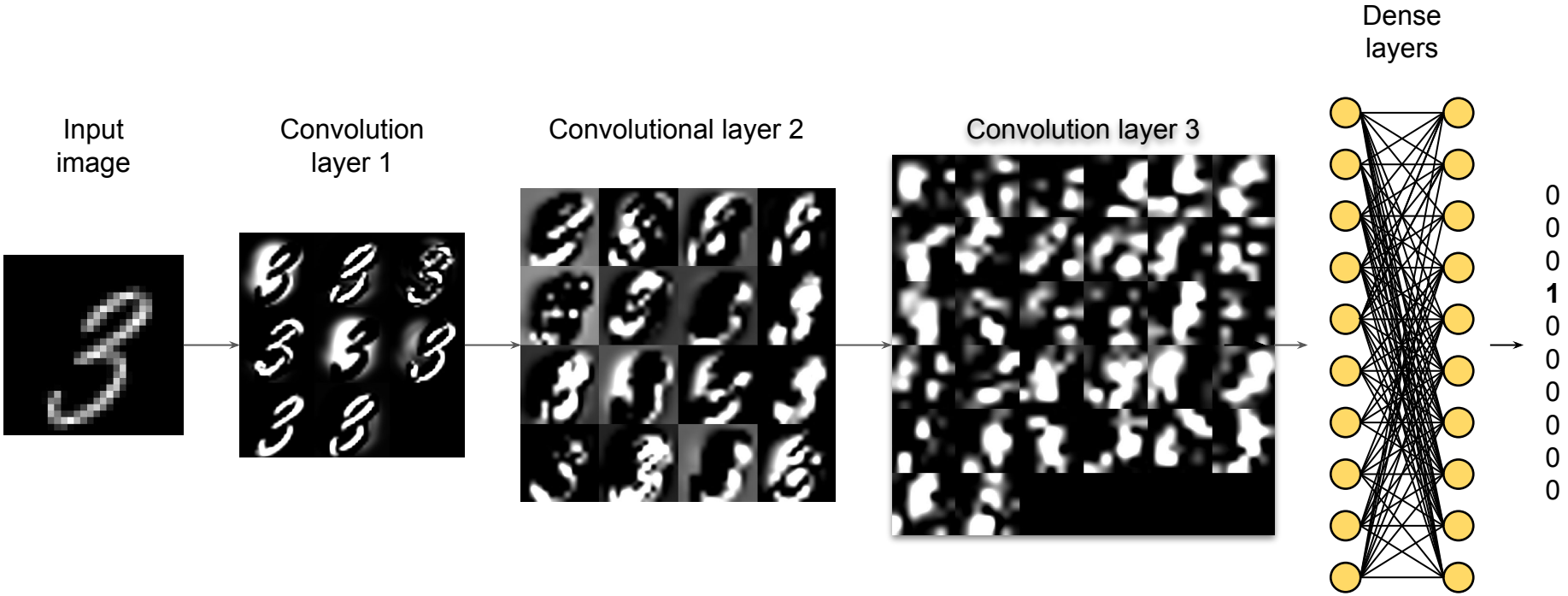
Feature maps : there is as **many maps as neurons in the convolution layer**

ConvNet summary



Feature maps : there is as **many maps as neurons in the convolution layer**

ConvNet summary



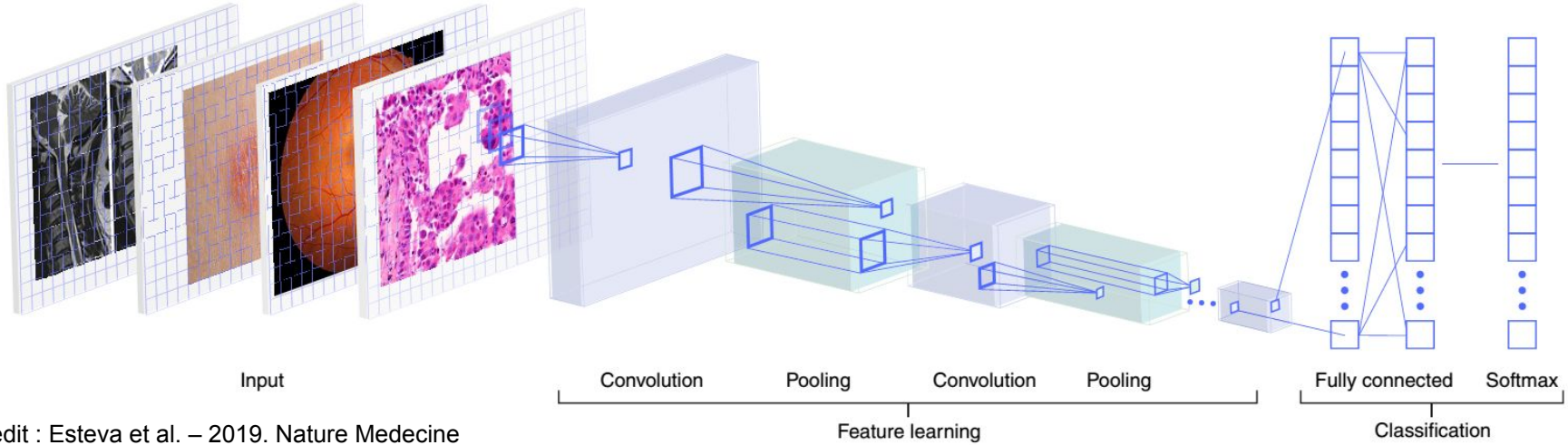
ConvNet architecture for image classification

The first part of the network is **using convolution layers to learn the features**

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

The second part is 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. Image classification :

   A. Introduction to convolutional network : digit classification
   B. Application to red-blood cells classification
   C. Introduction to transfer-learning

II. Segmentation :

   A. Application of a fully convolutional network (Unet)
   B. Instance segmentation with existing tools

III. Conclusion :

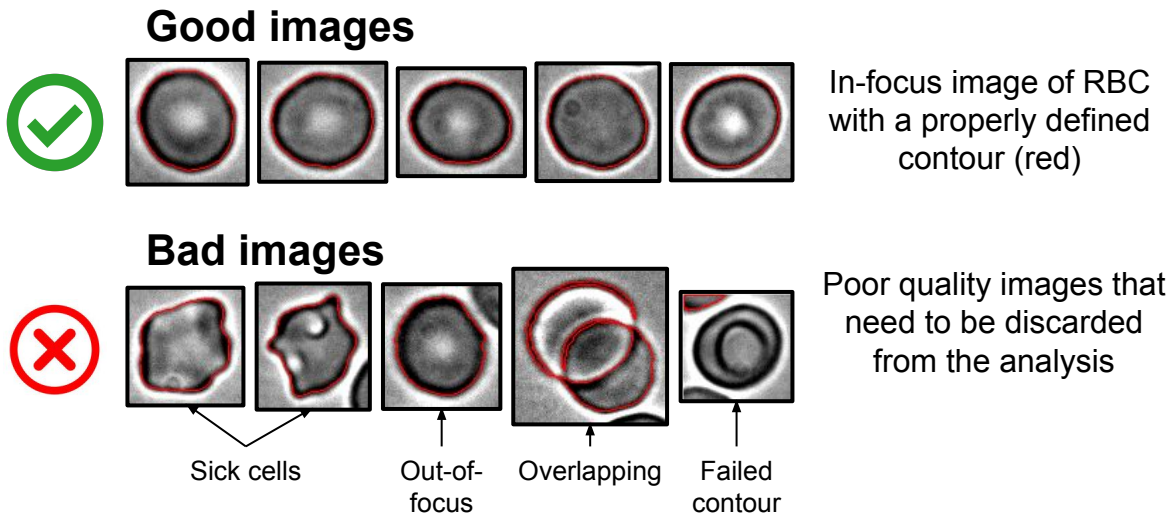
Example 5: Red blood cell image classification

Example : Ex5_RBC_image_classification.ipynb

1. Build your own convolutional network for image classification
2. Understand how to detect overfitting and which strategies to avoid it
3. Introduction to transfer learning

Viviana Claveria

Manouk Abkarian



Start with a good baseline model

A good practice is to **start working with a network architecture that is know to be efficient for your problem**. For example, the VGG16 architecture is easy to implement and well documented for image classification.

Convolution block #1

Convolution block #2

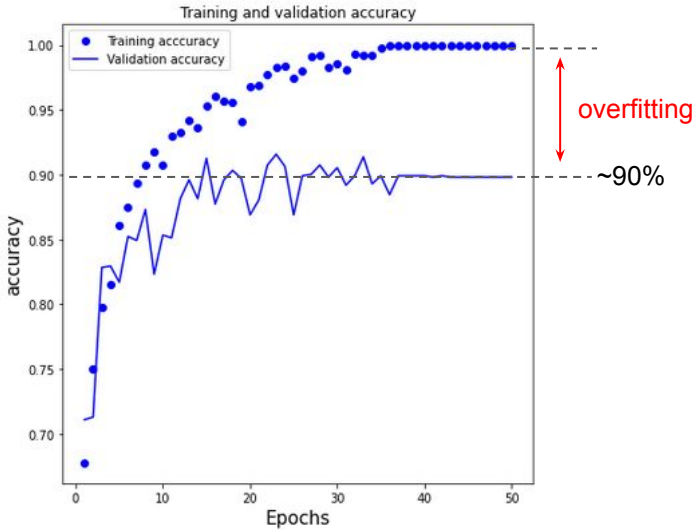
Convolution block #3

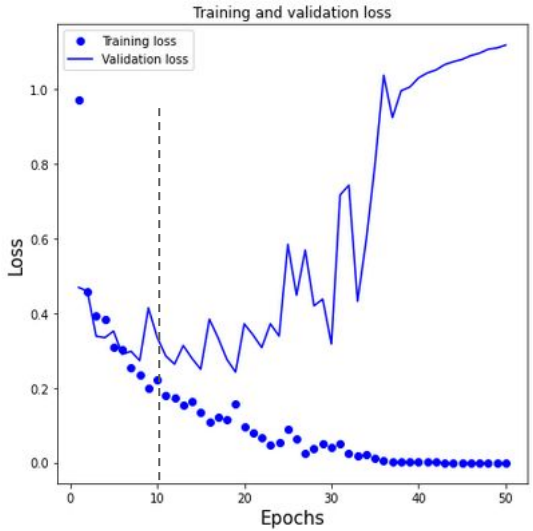
Fully connected network

```
model = Sequential([

    # Convolution Layer 1-2
    Conv2D(32, (3, 3), activation='relu', input_shape=(85,85,3)), #32 different 3x3 kernels
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)), # pool the max values over a 2x2 kernel

    # Convolution Layer 3-4
    Conv2D(64, (3, 3), activation='relu'), #64 different 3x3 kernels
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 5-6
    Conv2D(128, (3, 3), activation='relu'), #128 different 3x3 kernels
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(), # flatten the output of the last layer (7,7,128) into a single vector of length 6272

    # Fully Connected Layers
    Dense(256, activation = 'relu', kernel_initializer='random_normal', bias_initializer='zeros'), # 256 FCN nodes
    Dense(128, activation = 'relu', kernel_initializer='random_normal', bias_initializer='zeros'), # 128 FCN nodes
    Dense(1, activation = 'sigmoid', kernel_initializer='random_normal', bias_initializer='zeros'),
])

model.compile(optimizer = 'adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

VGG1 : Number of trainable parameters : <u>1,925,921</u>

Start with a good baseline model





We observe that the training and validation loss & accuracy are diverging after epoch #10. **The network is no longer learning useful features.**

- Overfitting
- Validation loss & accuracy are noisy
- The global accuracy of the network is ~92.9% when tested on the testing set.

Reduce the network size

Since overfitting means that the network is learning features specific to the training set, one strategy is to reduce the size of the network and check whether the property of the network are improving.

Convolution block #1

Convolution block #2

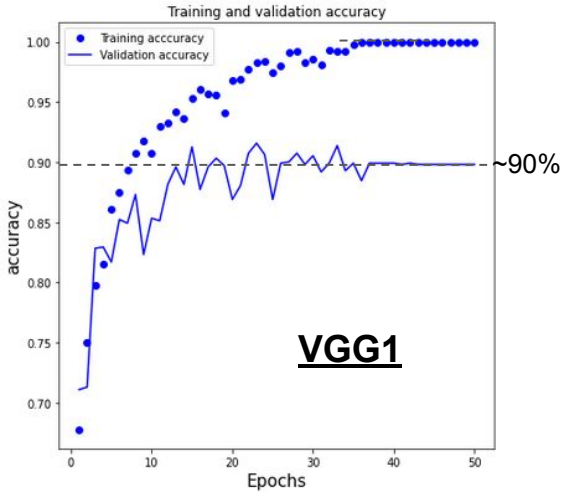
Convolution block #3

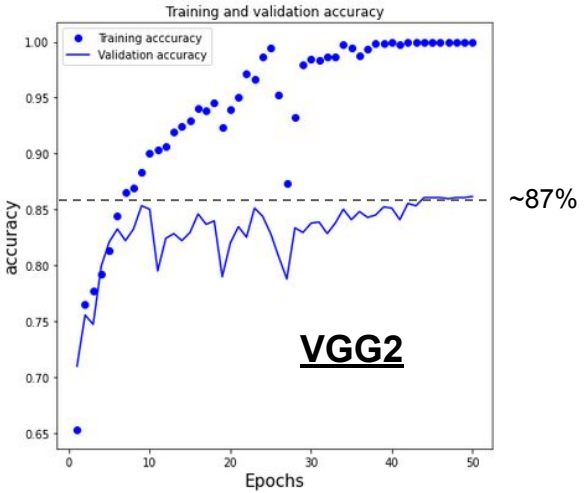
Fully connected network

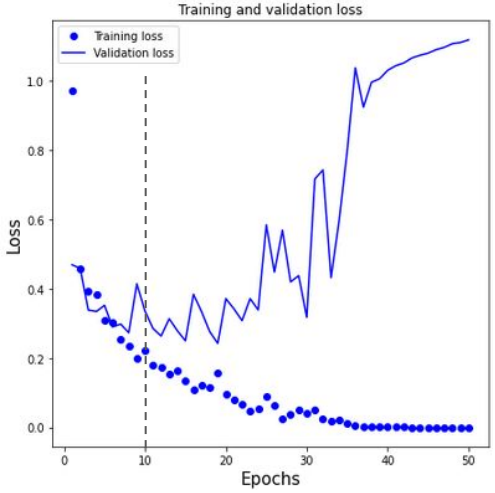
```
model = Sequential([

    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(85,85,3)), #32 different 3x3 kernels
    MaxPooling2D(pool_size=(2, 2)), # pool the max values over a 2x2 kernel

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), #64 different 3x3 kernels,
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), #128 different 3x3 kernels
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(), # flatten the output of the last convolution layer

    # Fully Connected Layers
    Dense(128, activation = 'relu', kernel_initializer='random_normal', bias_initializer='zeros'), # 128 FCN nodes
    Dense(64, activation = 'relu', kernel_initializer='random_normal', bias_initializer='zeros'), # 64 FCN nodes
    Dense(1, activation = 'sigmoid', kernel_initializer='random_normal', bias_initializer='zeros'),
])

model.compile(optimizer = 'adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

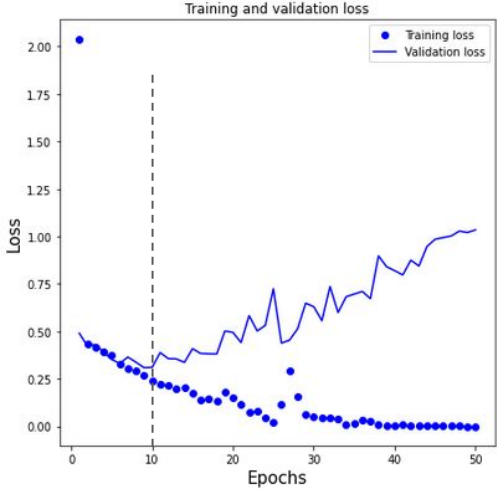
VGG2 : Number of trainable parameters : 1,150,273

Comparison VGG 1 & 2



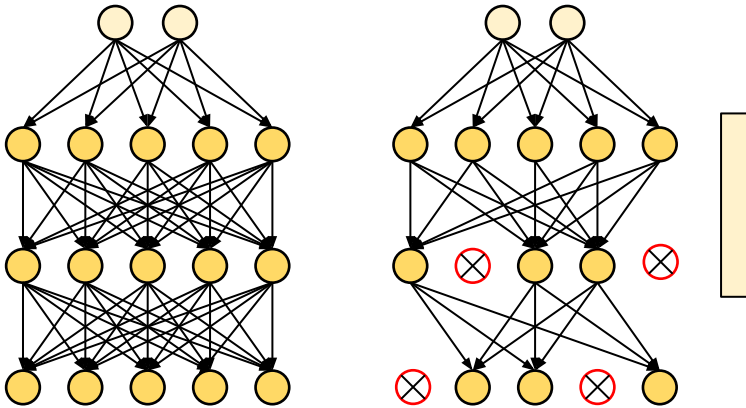






How to reduce overfitting?

- **Reduce the size of the network**
- **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 reduce overfitting?

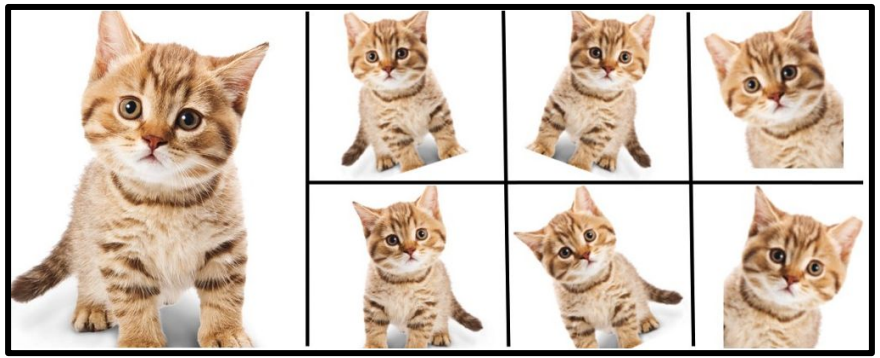
- **Reduce the size of the network**
- **Dropout**, randomly “turning-off” neurons of the network
- **Weight regularization L_1 & L_2**, a strategy to force the weights to take only small values during the training

L1/L2 regularization

Pic credit : Moen et al. 2019. Nat. Meth.

How to reduce overfitting?

- **Reduce the size of the network**
- **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
- Increase the size of the training set:
  - Add new images to the training set
  - Use **data augmentation**



Pic credit :https://www.quora.com/What-is-data-augmentation-in-deep-learning

VGG baseline and Dropout regularization :

Image augmentation layer

Dropout 20%

Dropout 20%

Dropout 20%

Dropout 50%

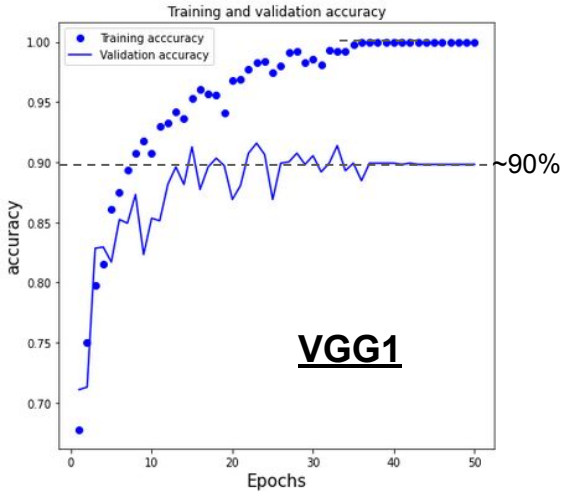
Dropout 50%

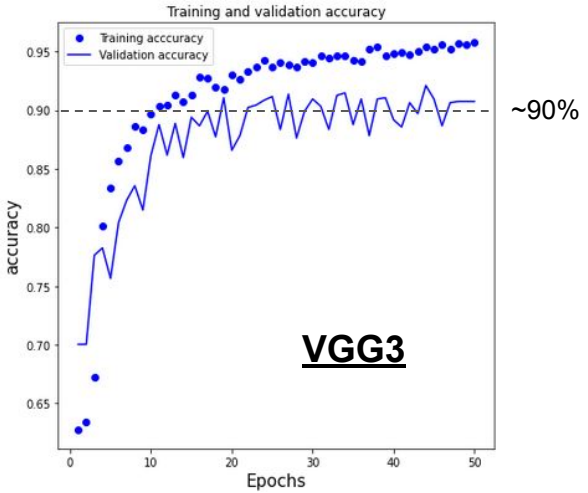
```
model = Sequential([

    Input(shape=(85, 85, 3)),
    Rescaling(scale=1./255),
    RandomFlip(mode="horizontal_and_vertical"),

    # Convolution Layer 1-2
    Conv2D(32, (3, 3), activation='relu'), #32 different 3x3 kernels
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)), # pool the max values over a 2x2 kernel
    Dropout(0.2),

    # Convolution Layer 3-4
    Conv2D(64, (3, 3), activation='relu'), #64 different 3x3 kernels,
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    # Convolution Layer 5-6
    Conv2D(128, (3, 3), activation='relu'), #128 different 3x3 kernels,
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    Flatten(), # flatten the output of the last convolution layer

    # Fully Connected Layers
    Dense(256, activation = 'relu'), # 256 FCN nodes
    Dropout(0.5),
    Dense(128, activation = 'relu'), # 128 FCN nodes
    Dropout(0.5),
    Dense(1, activation = 'sigmoid')
])
```

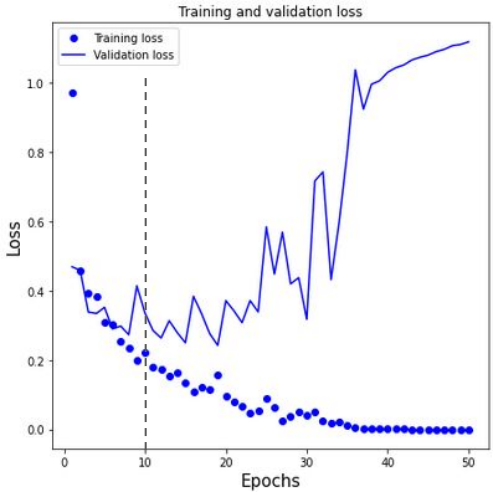
VGG3 : Number of trainable parameters : 1,925,921

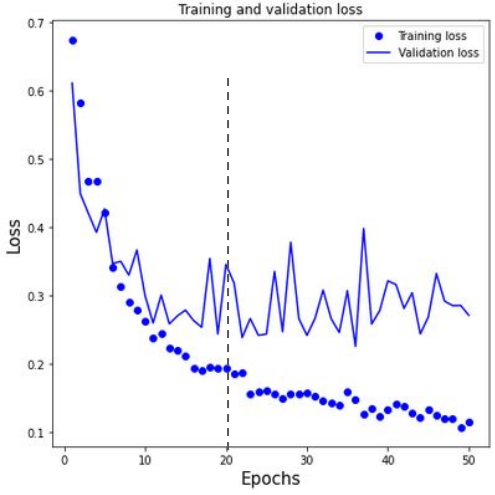
Comparison VGG 1 & 3

- Dropout & image augmentation help reduced the overfitting
- Global accuracy calculated with the testing set :
  - VGG1 : 92.9%
  - VGG2 : 89.9%
  - **VGG3 : 94.4%**



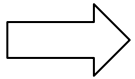






What did we learn?

- How to create and use a **convolutional neural network for image classification** :
    - convolution layer
    - max-pool layer
- Recognize **overfitting** and methods to reduce it while **optimizing the performances of the network** :
    - dropout
    - regularization
    - **image augmentation**
    - batch normalization
    - transfert learning
    - ...



There is no “unique solution”. It strongly depends on your dataset. Need to use a “try & error” strategy.

Outline

I. Image classification :

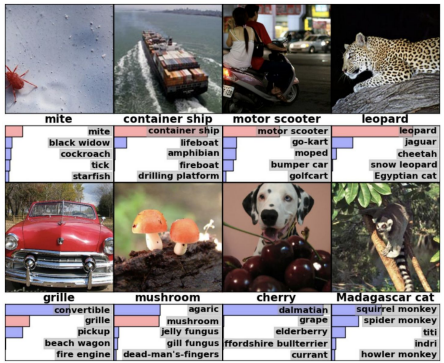
   A. Introduction to convolutional network : digit classification
   B. Application to red-blood cells classification
   C. Introduction to transfer-learning

II. Segmentation :

   A. Application of a fully convolutional network (Unet)
   B. Instance segmentation with existing tools

III. Conclusion :

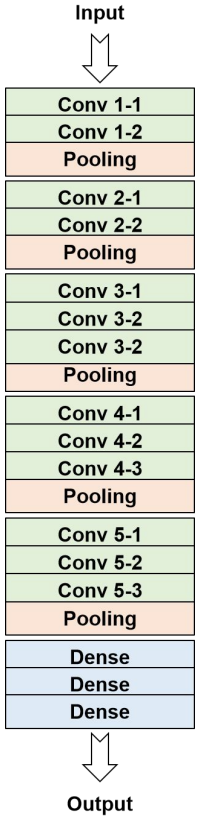
Using a pre-trained convNet for our problem



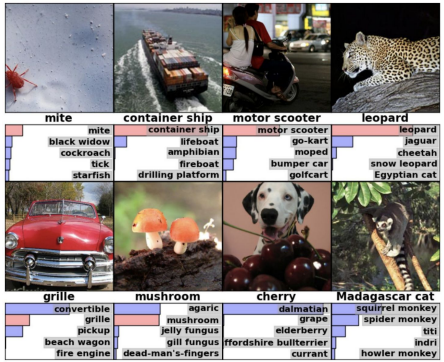
Many networks **trained on million of images** can be downloaded from internet :

https://keras.io/applications/
Greenwald et al. - 2022

138,357,544



Using a pre-trained convNet for our problem

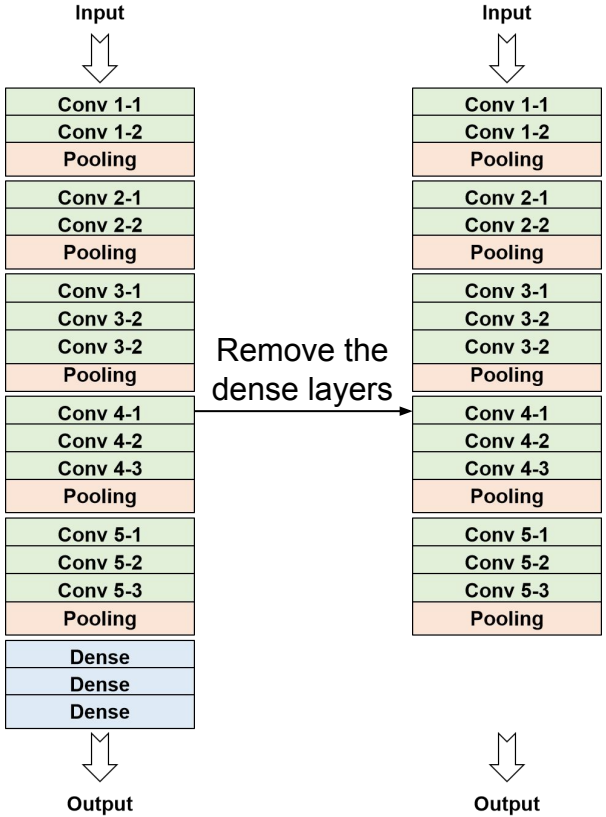


Many networks **trained on million of images** can be downloaded from internet :

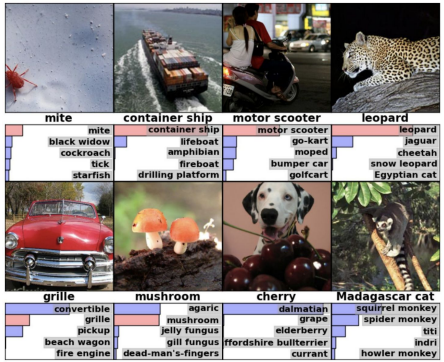
https://keras.io/applications/
Greenwald et al. - 2022

BIAS & WEIGHTS

138,357,544



Using a pre-trained convNet for our problem

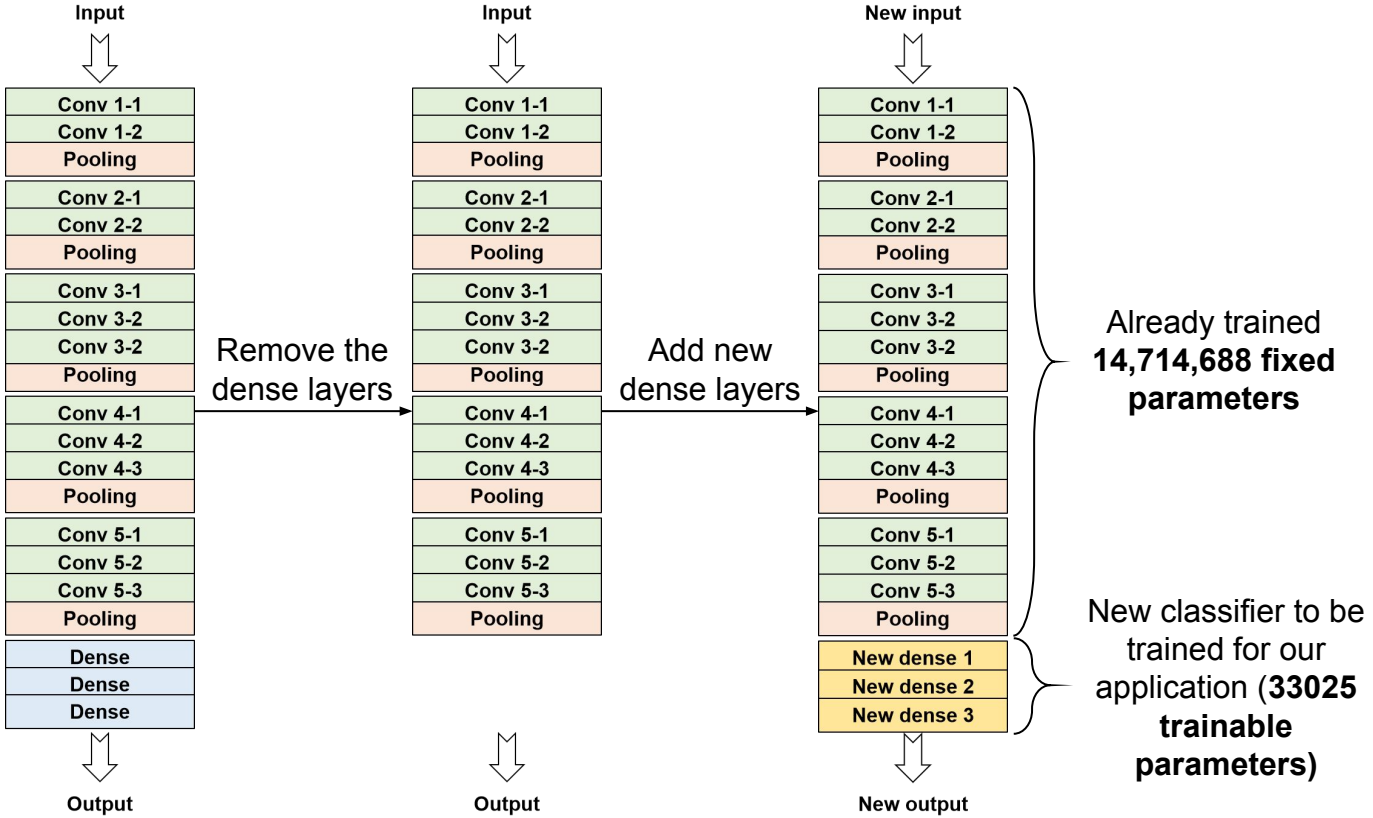


Many networks **trained on million of images** can be downloaded from internet :

https://keras.io/applications/
Greenwald et al. - 2022

BIAS & WEIGHTS

138,357,544

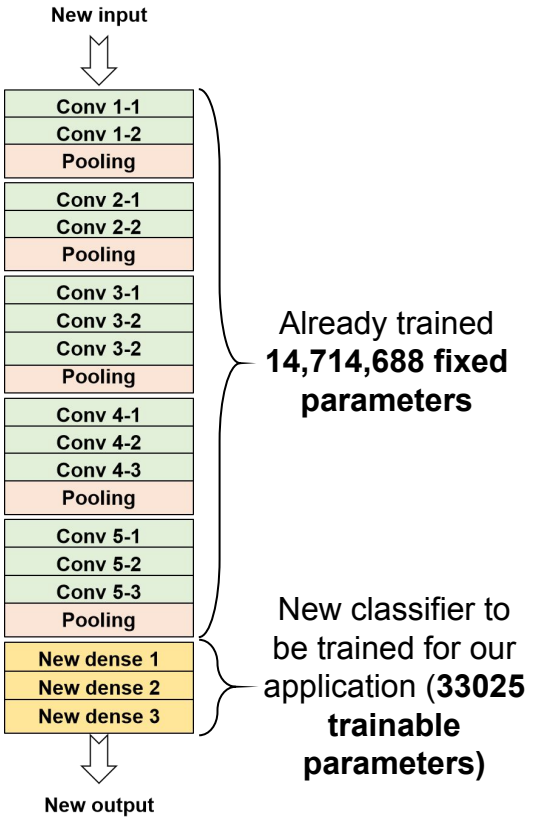


Transfer learning syntax

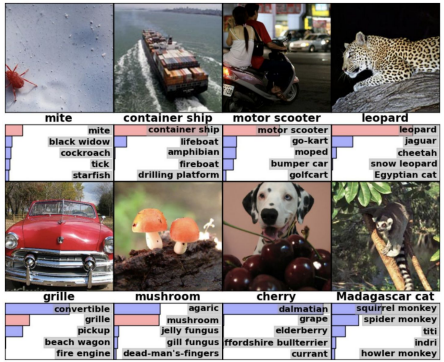
Load the pre-trained VGG16 network

Define the new network, adding the classifier

```
# from keras.applications import the vgg16 network pretrained on the imagenet
# database. The include_top option is set to zero, meaning that the classifier
# part (composed of a dense network) is removed.
# ------------------------------------------------
conv_base = keras.applications.vgg16.VGG16(
    include_top=False,
    weights='imagenet',
    input_shape=(IMG_WIDTH, IMG_HEIGHT, IMG_CHANNEL)
)

# build the new network using the trained VGG16 network and adding a new classifier
# ---------------------------------------------------------------------------------
model_VGG = Sequential([

    # Initialization, normalization and image augmentation
    Input(shape=(IMG_WIDTH, IMG_HEIGHT, IMG_CHANNEL)),
    Rescaling(scale=1./255),
    RandomFlip(mode="horizontal_and_vertical"),

    # Add the VGG16 network without the classifier
    conv_base,

    Flatten(),

    # Create the fully connected layers for the final classification
    Dense(256, activation = 'relu'), # 256 FCN nodes
    Dropout(0.5),
    Dense(128, activation = 'relu'), # 128 FCN nodes
    Dropout(0.5),
    Dense(1, activation = 'sigmoid'),
])
```



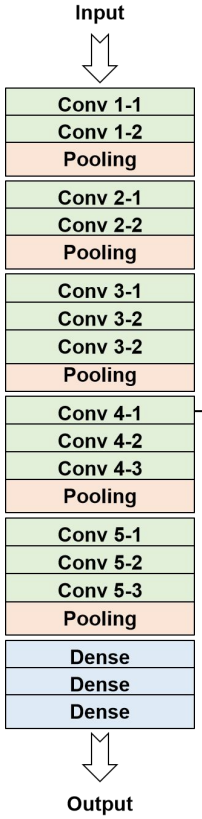
Fine-tuning



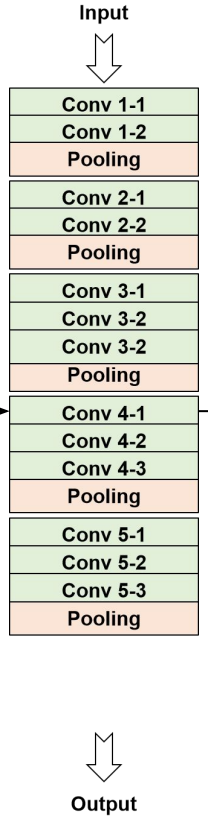
Many networks **trained on million of images** can be downloaded from internet :

https://keras.io/applications/
Greenwald et al. - 2022

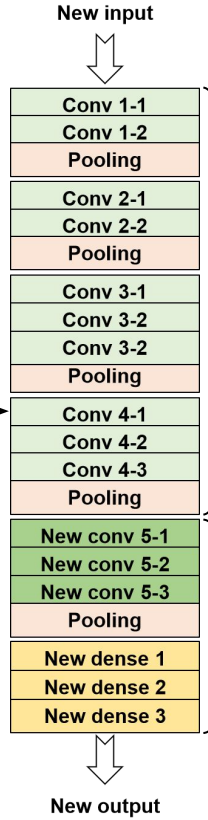
138,357,544



Remove the dense layers



Add new dense layers



Already trained **7,635,264 fixed parameters**

Fine tune the last convolutional layers and the new classifier **(7,636,993 parameters)**

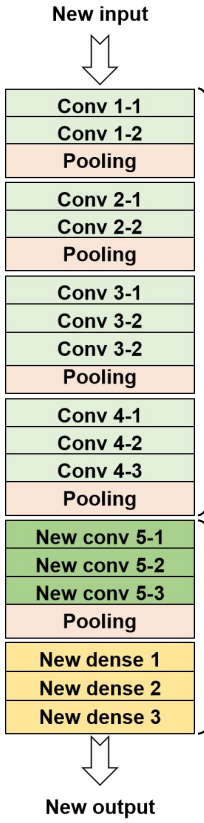
Fine tuning syntax

Load the pre-trained VGG16 network

Define the new network, adding the classifier

Allow to retrain the 4 last layers of VGG16

```
# from keras.applications import the vgg16 network pretrained on the imagenet
# database. The include_top option is set to zero, meaning that the classifier
# part (composed of a dense network) is removed.
# ------------------------------------------------
conv_base = keras.applications.vgg16.VGG16(
    include_top=False,
    weights='imagenet',
    input_shape=(IMG_WIDTH, IMG_HEIGHT, IMG_CHANNEL)
)

# build the new network using the trained VGG16 network and adding a new classifier
# ------------------------------------------------------------------------------------
model_VGG = Sequential([

    # Initialization, normalization and image augmentation
    Input(shape=(IMG_WIDTH, IMG_HEIGHT, IMG_CHANNEL)),
    Rescaling(scale=1./255),
    RandomFlip(mode="horizontal_and_vertical"),

    # Add the VGG16 network without the classifier
    conv_base,

    Flatten(),

    # Create the fully connected layers for the final classification
    Dense(256, activation = 'relu'), # 256 FCN nodes
    Dropout(0.5),
    Dense(128, activation = 'relu'), # 128 FCN nodes
    Dropout(0.5),
    Dense(1, activation = 'sigmoid'),
])

# The conv_base is composed of 19 layers. The last 4 layers are related to
# block5. Therefore, only the four last payers will be set to "trainable".
# ------------------------------------------------------------------------
conv_base.trainable = True
for n in range(15):
    conv_base.layers[n].trainable = False
```

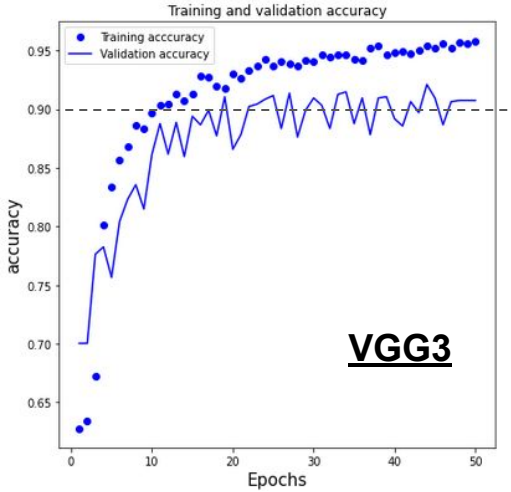


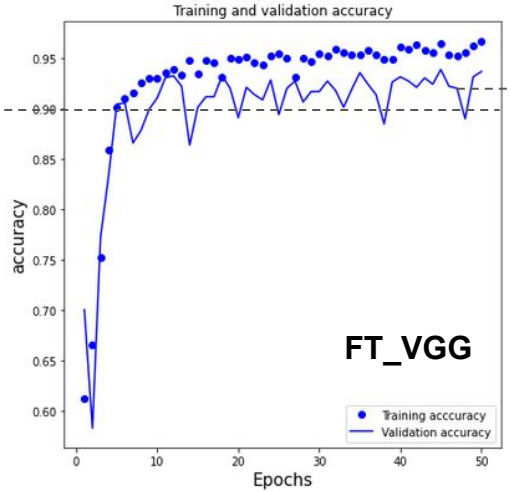
Already trained **7,635,264 fixed parameters**

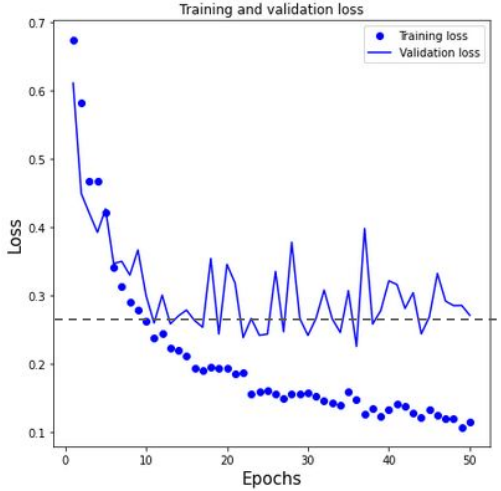
Fine tune the last convolutional layers and the new classifier **(7,636,993 parameters)**

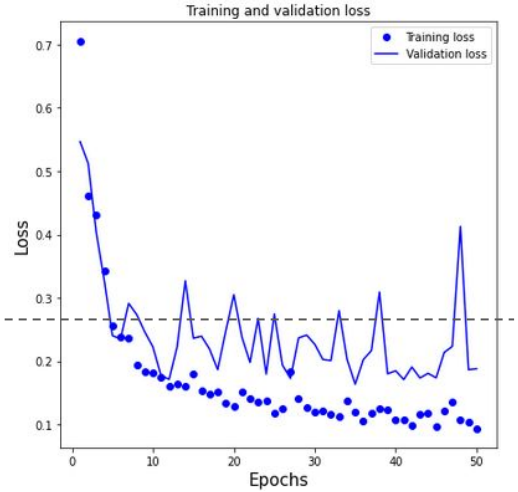
Comparison VGG 1 & Fine tuning

- Dropout & image augmentation help reduced the overfitting
- Global accuracy calculated with the testing set :
  - VGG1 : 92.9%
  - VGG2 : 89.9%
  - **VGG3 :** 94.4%
  - **FT_VGG : 95.2%**









Outline

I. Image classification :

   A. Introduction to convolutional network : digit classification
   B. Application to red-blood cells classification
   C. Introduction to transfer-learning

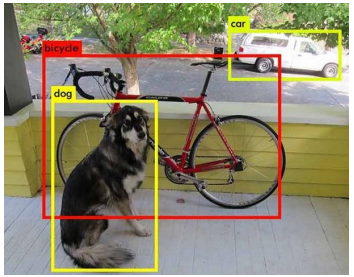
II. Segmentation :

   A. Application of a fully convolutional network (Unet)
   B. Instance segmentation with existing tools

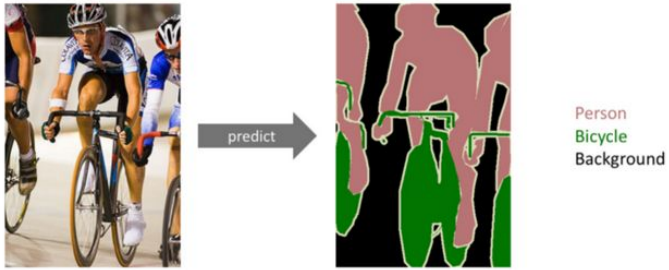
III. Conclusion :

Computer Vision - possible tasks

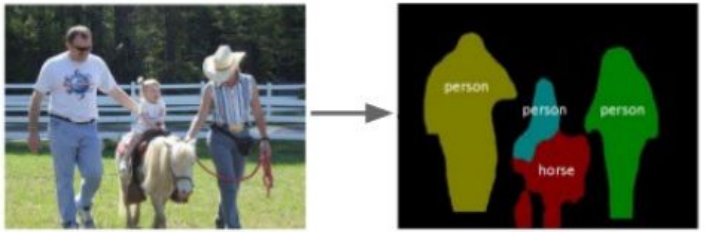
classification
“cat”



classification + localization
class + bounding box



Semantic segmentation
each pixel : class



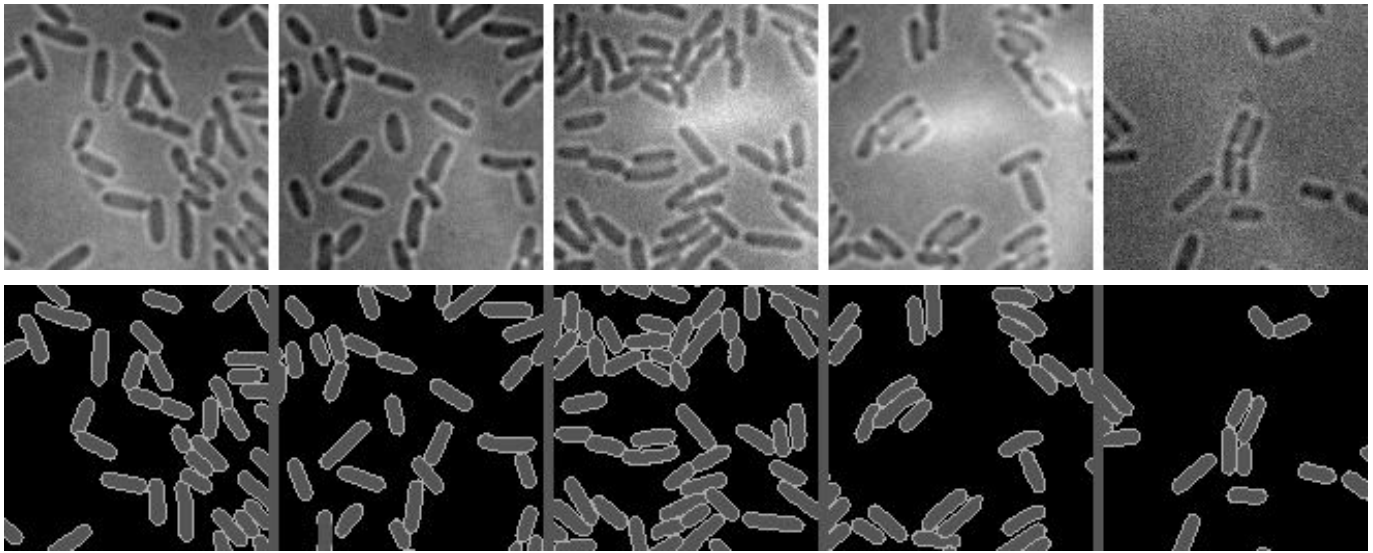
Instance segmentation for each pixel:
class “person” + instance #1
class “person” + instance #2 ...

Example 6: Bacteria segmentation

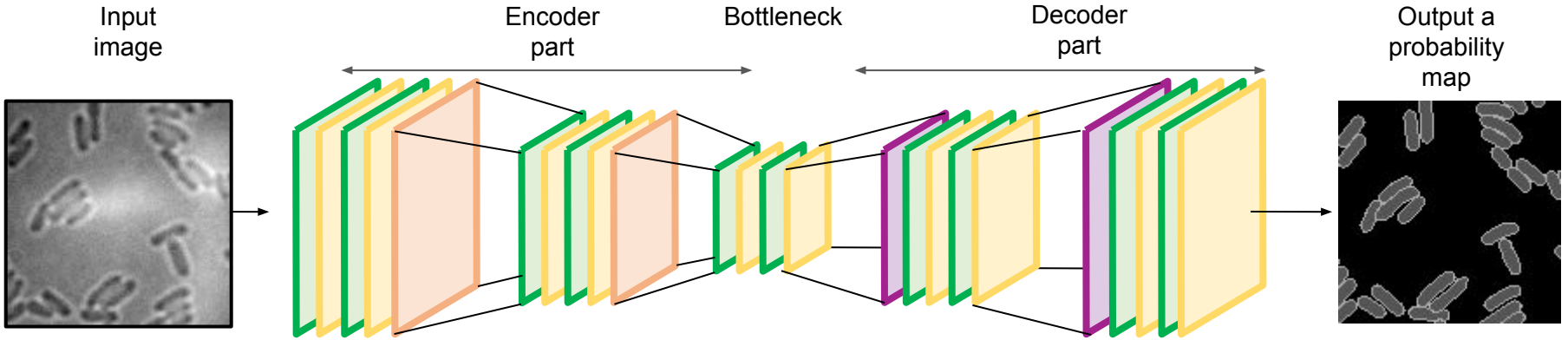
Example : Ex6_bacteria_segmentation_unet.ipynb

1. Introduction to semantic segmentation
2. Illustration of a Unet application

Caroline Clerté



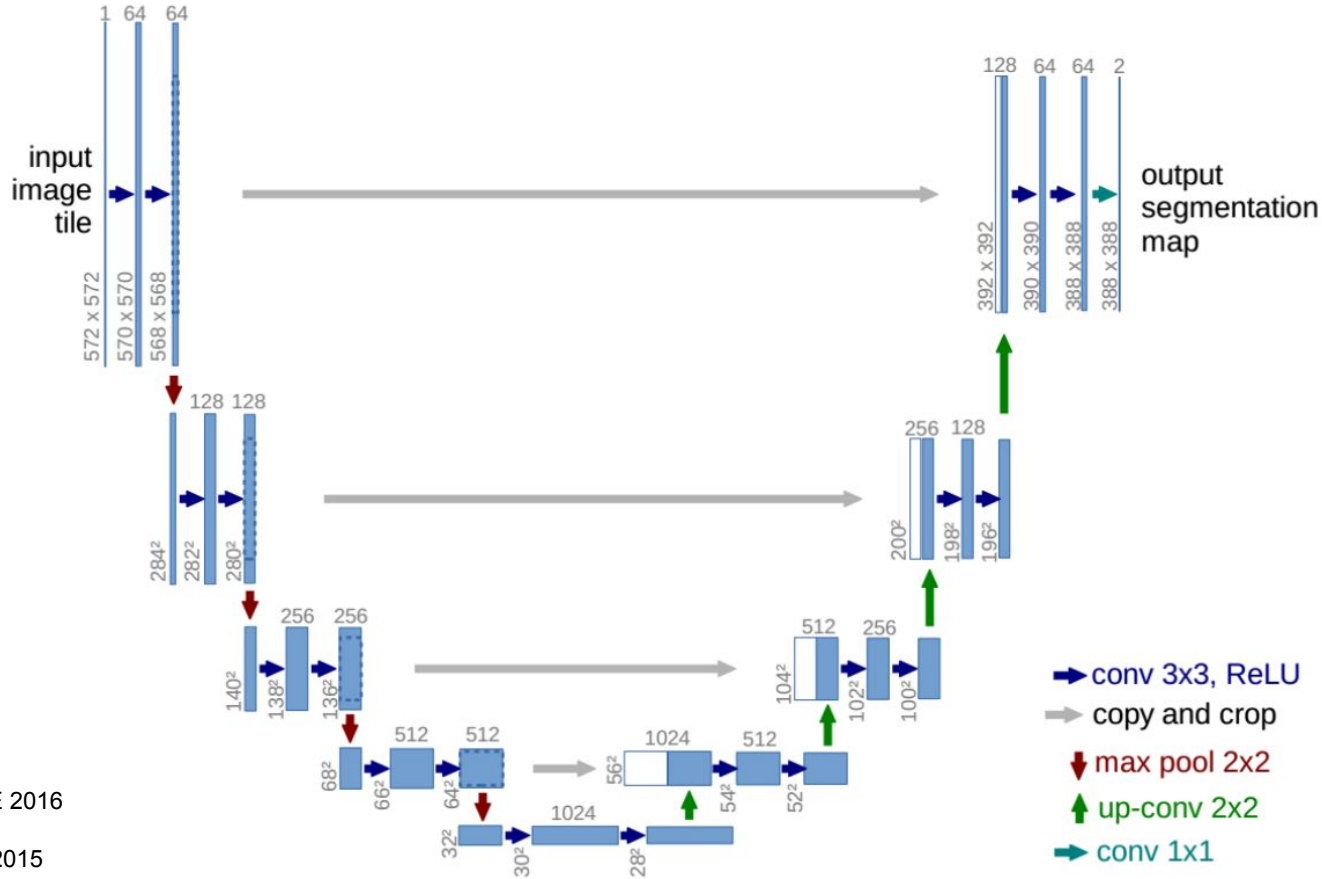
Fully convolutional network



This type of neural network architecture has many advantages :

- **simple** to implement
- **fast**
- **low number of parameters** since there is no densely connected layers
- “no” restriction on the size of the input images
- the output image is already segmented

The Unet : a very popular network



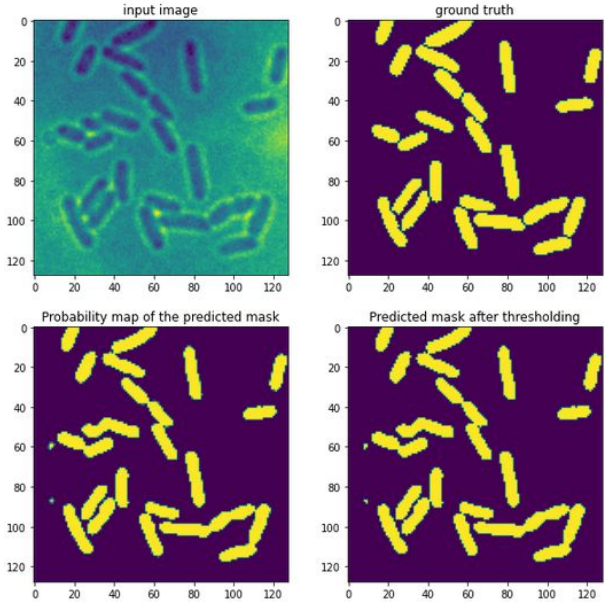
Badrinarayanan et al. IEEE 2016
Noh et al. ArXiv 2015
Ronnerberger et al. ArXiv 2015

Comments on example 6

Data set is composed of **98 training (2014 cells)** & **32 testing (512 cells)** images.

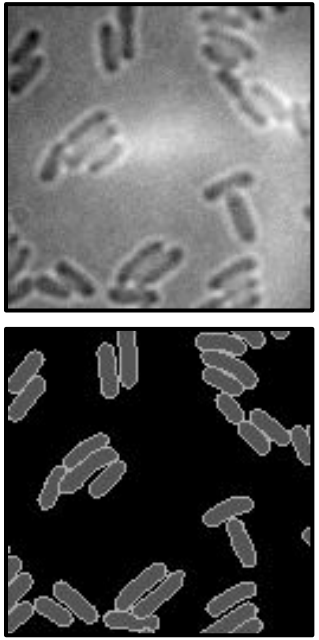
For a good generalization, the data have been acquired with different :

- density
- focus
- illumination



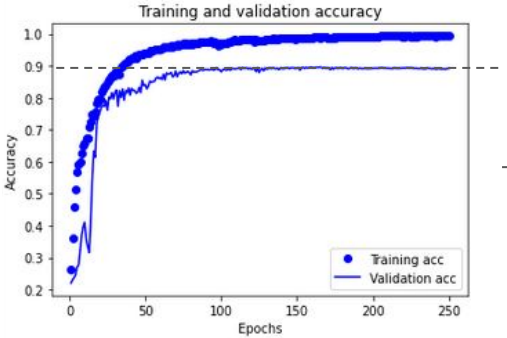
The network is rather small (for example comparing to the network used for image classification) since the number of Unet is composed of “only” 4,119,153 trainable parameters.

Comments on example 6



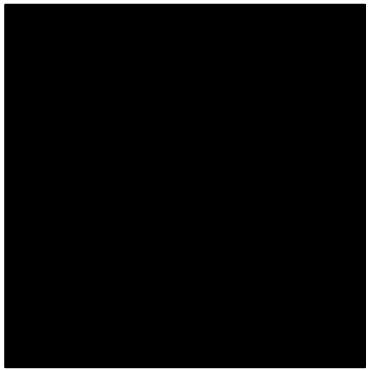
U net

Training



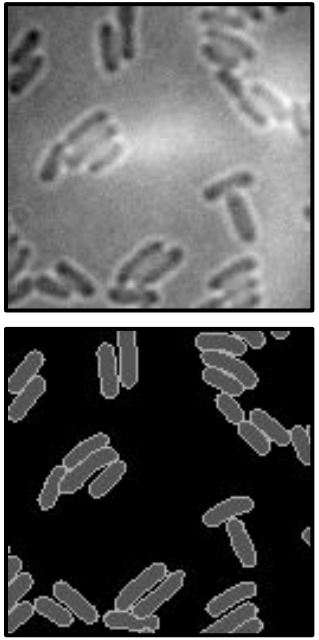
~90%

Test



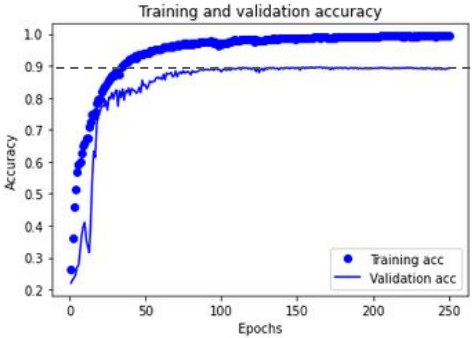
Comments on example 6

For segmentation problem, the pixel **accuracy is not a good metrics**, particularly when the density of objects to segment is low.
Metrics such as **IoU or dice** should be used instead.



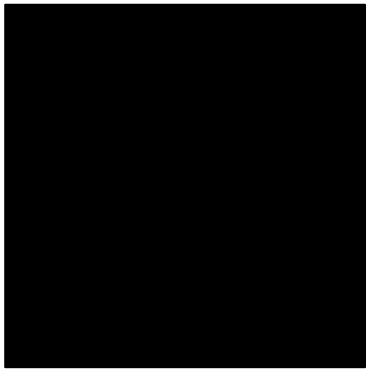
U net

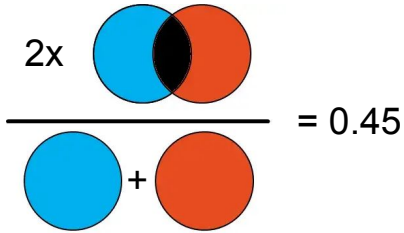
Training



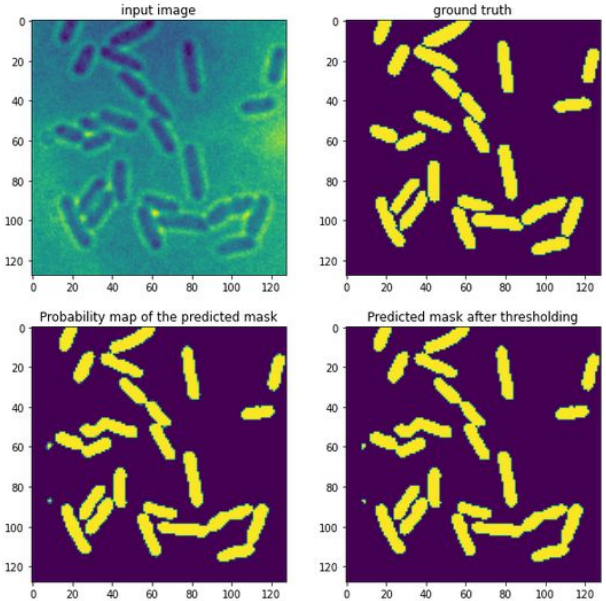
~90%

Test

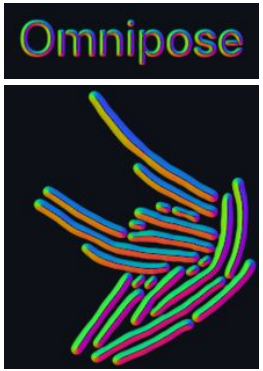




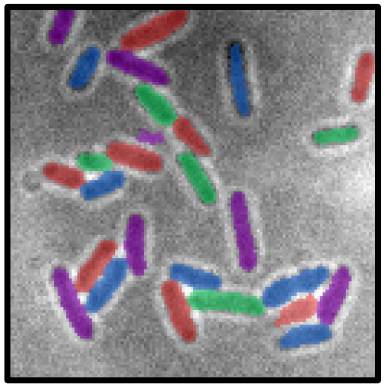
Take advantage of existing tools!



Segmented map is fairly good but it would require further image analysis & tuning of the network to get the cells well separated.



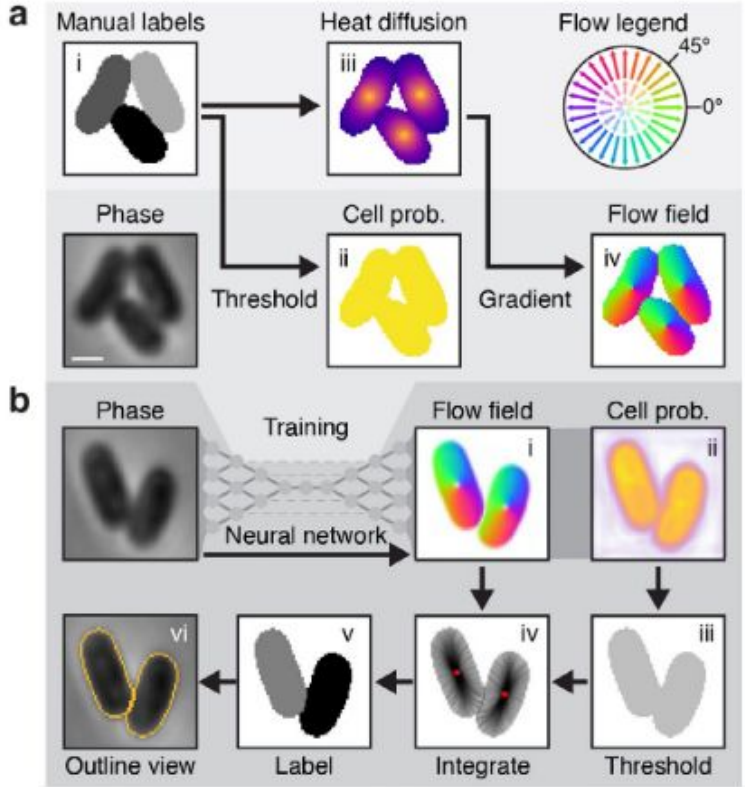
Many pre-trained network are freely available on github and can be **used or re-trained for your applications**.



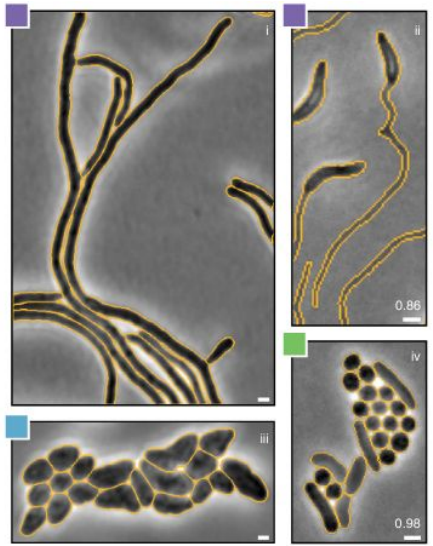
Segmentation results obtained with the model **bact_phase_omni** freely accessible on https://github.com/kevinjohncutler/omnipose/

Stringer et al. 2021 - Cutler et al. - 2022

Omnipose … is still Unet, but smarter!



Omnipose is a Unet not only trained to **output a probability map** but also **a flow-field map that will detecting the cells, whatever their shape**.



Stringer et al. 2021 - Cutler et al. - 2022

Many other tools available for segmentation

https://github.com/stardist/stardist - Schmidt et al. - 2018 Cell Detection with Star-Convex Polygons

https://github.com/hci-unihd/plant-seg - Wolny et al. - 2020 Accurate and versatile 3D segmentation of plant tissues at cellular resolution

https://github.com/MouseLand/cellpose - Stringer et al. 2021 Cellpose: a generalist algorithm for cellular segmentation

https://github.com/kevinjohncutler/omnipose - Cutler et al. - 2022 Omnipose: a high-precision morphology independent solution for bacterial cell segmentation

https://github.com/vanvalenlab/intro-to-deepcell - Greenwald et al. - 2022 Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning

All these networks are **freely accessible**, **well documented and providing :**

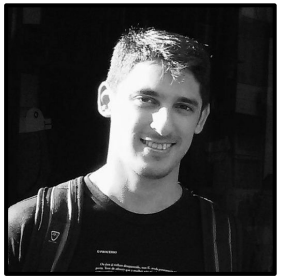
- pre-trained models
- user-friendly GUI for simple tests
- scripts for running / training

https://github.com/HenriquesLab/ZeroCostDL4Mic

Example 7: Nuclei segmentation

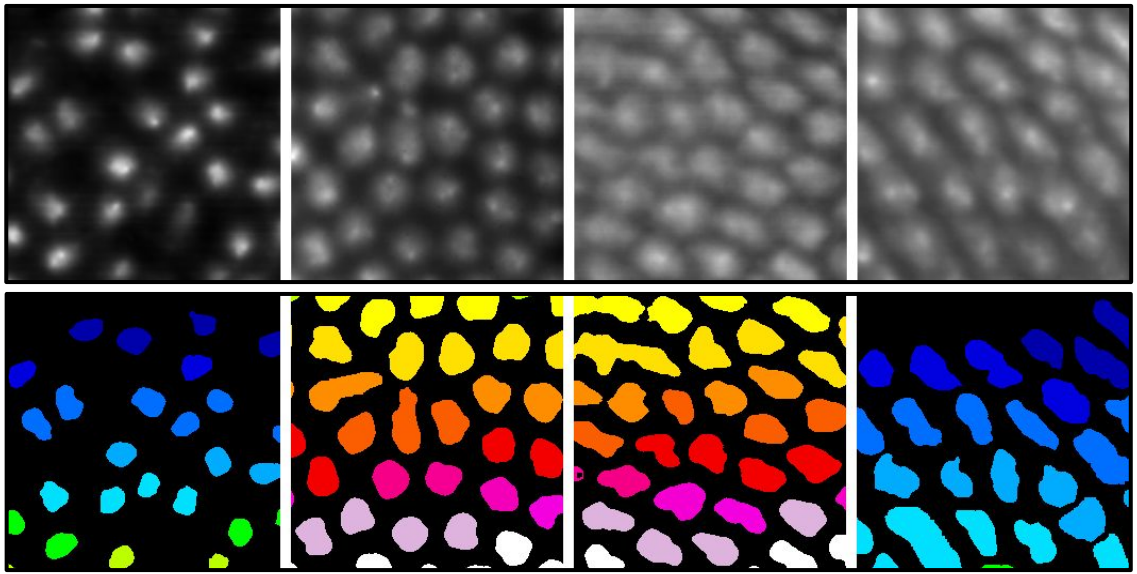
Example : Ex7_nuclei_segmentation_stardist.ipynb

1. Example of instance segmentation
2. Application of starDist for 2D segmentation

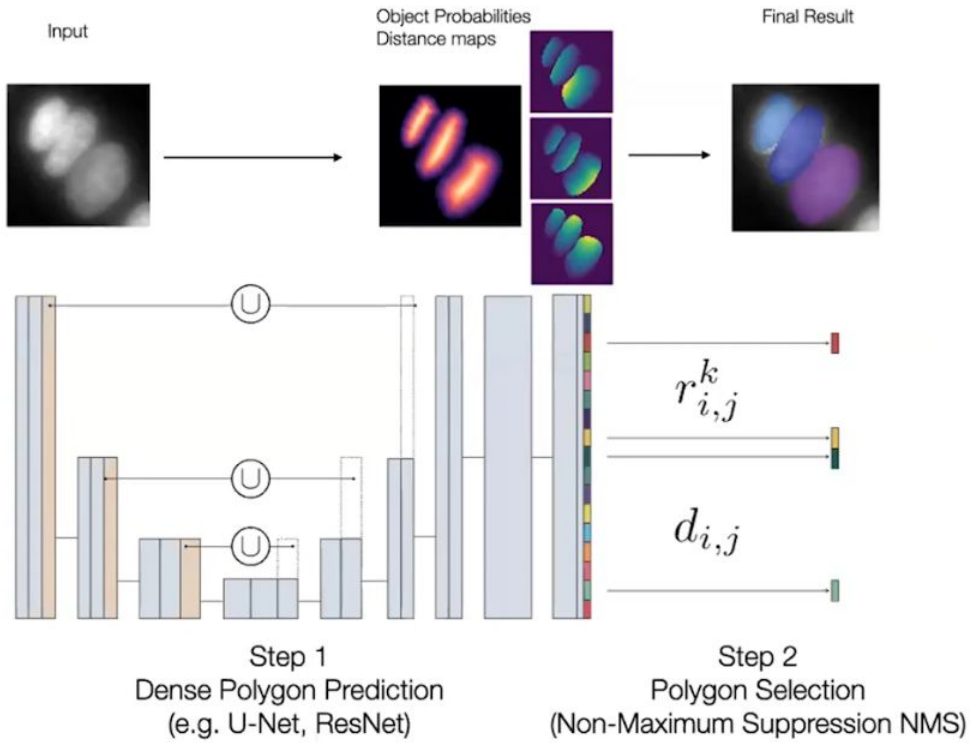


Sergio Espinola

Marcelo Nöllmann



StarDist



Important:
Assumes objects are *star-convex*

Martin Weigert presentation of starDist- https://www.youtube.com/watch?v=Amn_eHRGX5M - Schmidt et al. 2018 Cell detection with star-convex polygons

Outline

I. Image classification :

   A. Introduction to convolutional network : digit classification
   B. Application to red-blood cells classification
   C. Introduction to transfer-learning

II. Segmentation :

   A. Application of a fully convolutional network (Unet)
   B. Instance segmentation with existing tools

III. Conclusion :

Take home messages

I. Do I need deep learning for my problem ?

> “The hardest and the most time-consuming part of any deep learning is acquiring training data. [...] You typically need hundreds or thousands of examples at minimum, and creating the annotations itself is tedious.”
>
> Beth Cimini

Ravindran - 2022 Five ways deep learning has transformed image analysis

Take home messages

I. Do I need deep learning for my problem ? Can it work for my problem?

> “The hardest and the most time-consuming part of any deep learning is acquiring training data. [...] You typically need hundreds or thousands of examples at minimum, and creating the annotations itself is tedious.”
>
> Beth Cimini

> “People kind of expect that these models can just perform miracles, but if the information that you want to pull out isn’t there in the data, then in my view and also in my experience, it’s unlikely to work,”
>
> David Van Valen

Take home messages

I. Do I need deep learning for my problem ? Can it work for my problem?

II. Start with **simple data** and a **solid baseline**

> How much data is enough? We observed diminishing returns to training data at ~10 4 –10 5 labels. We believe that the effort required to go beyond this scale is warranted when accuracy is a paramount concern, for example for models applied across projects or in clinical contexts, which is the case for Mesmer. This effort is also worthwhile for generating gold-standard datasets to benchmark model performance. For other use cases, smaller datasets and bespoke models may suffice.
>
> Greenwald et al. - 2022

Take home messages

I. Do I need deep learning for my problem ? Can it work for my problem?

II. Start with **simple data** and a **solid baseline**

III. Setup your **own evaluation strategy** :

- build a test set representative of your problem
- choose the right performance indicators for your problem

Take home messages

I. Do I need deep learning for my problem ? Can it work for my problem?

II. Start with **simple data** and a **solid baseline**

III. Setup your **own evaluation strategy**

IV. **Know your data** : **Supervised machine learning is DATA driven, not algorithm driven**

- your training set should be as representative as possible of your data
- double/triple check for errors

Take home messages

I. Do I need deep learning for my problem ? Can it work for my problem?

II. Start with **simple data** and a **solid baseline**

III. Setup your **own evaluation strategy**

IV. **Know your data** : **Supervised machine learning is DATA driven, not algorithm driven**

V. Check for already existing tools that could be adapted to your data

How to start with Deep Learning (for free)?

Python 3 – open source

For DL, the open-source **TensorFlow** and **PyTorch** libraries are used.

Colab (google)
free GPU
python jupyter

https://csbdeep.bioimagecomputing.com/

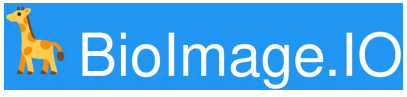
https://github.com/HenriquesLab/ZeroCostDL4Mic

\+

https://www.youtube.com/c/DigitalSreeni

https://www.youtube.com/c/CNRSFormationIDLE?app=desktop

https://cs230.stanford.edu/lecture/



https://bioimage.io/#/

https://www.kaggle.com/

https://bbbc.broadinstitute.org/image_sets