Introduction to Deep Learning

Graduate School Orléans Numérique

Lecture 3: convolutional networks

Daniel Förster
UFR Sciences et Techniques
Université d'Orléans

Agenda

- Why dense layers are a poor fit for images
- The convolution: filters, feature maps, channels
- Padding, stride, pooling, receptive field
- A convolutional network for MNIST
- Deeper networks: residual connections, transfer learning
- Limits, and beyond classification
- A research example: nanoparticles in the electron microscope

`daniel-forster.fr/dl`

slides, problem sheets, notebooks, project ideas

Why dense layers are a poor fit for images

Too many parameters. A 224×224 colour photo has 150 528 input values; a first dense layer of 1 000 units needs 150 million weights.

No notion of neighbourhood. Shuffle the pixels of every image with the same permutation: a dense network trains exactly as well. It never uses the fact that nearby pixels belong together.

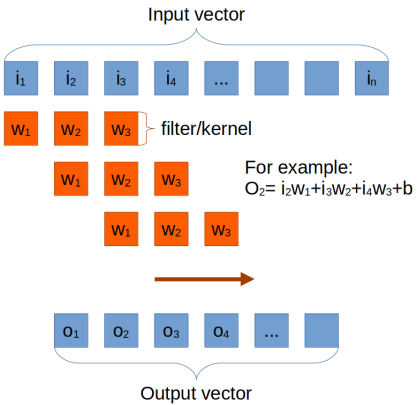
No notion of position. A digit moved by three pixels is a new input vector: the digit must be learned again at every position.

Idea: build what we know about images into the architecture

- **locality**: a unit looks at a small patch
- **weight sharing**: the same detector everywhere
- **hierarchy**: edges $\rightarrow$ strokes $\rightarrow$ parts $\rightarrow$ objects

Inspiration: the visual cortex. Hubel & Wiesel (1959, Nobel 1981): neurons that respond to edges of one orientation in one small region of the visual field.
Neocognitron (Fukushima, 1980), then convolutional networks trained by backpropagation (LeCun, 1989).

Convolution in 1D



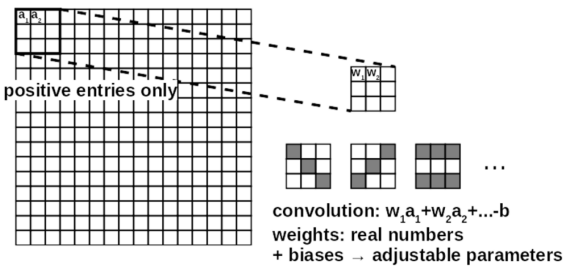
A small **kernel** (or **filter**) of K weights slides along the input:

$$y_i = \sum_{k=1}^{K} w_k \, x_{i+k-1} + b, \qquad \text{then an activation}$$

- every output sees only K neighbouring inputs (**local**)
- the same K weights and bias at every position (**shared**)
- a kernel of size 3 has 4 parameters, whatever the input length
- without padding, n inputs give $n - K + 1$ outputs

Mathematicians call this a cross-correlation: their convolution flips the kernel. Learned weights absorb the flip.

Convolution in 2D



A $k \times k$ kernel slides over the image:

$$y_{ij} = \sum_{m=1}^{k} \sum_{n=1}^{k} w_{mn}\, x_{i+m-1,\, j+n-1} + b$$

- the output is again an image: a **feature map**
- a 3×3 kernel has $9 + 1 = 10$ parameters
- an $n \times n$ input gives an $(n - k + 1) \times (n - k + 1)$ output: $28 \times 28 \to 26 \times 26$
- one kernel detects one pattern; a layer has many kernels

Your turn: a convolution by hand

Image X (5×5) and kernel K (3×3), no padding, no bias:

$$X = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \qquad K = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

1. What is the size of the output?
2. Compute it.
3. The image is shifted by one pixel to the right (a column of zeros enters on the left). What happens to the output?

Your turn: a convolution by hand

Image X (5×5) and kernel K (3×3), no padding, no bias:

$$X = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \qquad K = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

1. What is the size of the output?
2. Compute it.
3. The image is shifted by one pixel to the right (a column of zeros enters on the left). What happens to the output?

1. $(5 - 3 + 1) \times (5 - 3 + 1) = 3 \times 3$
2. Each row of the window gives $x_3 - x_1$:

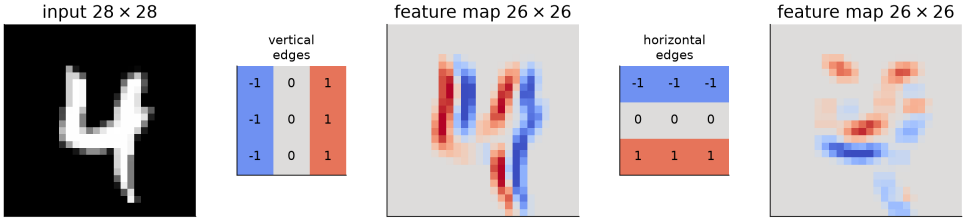
$$Y = \begin{pmatrix} 3 & 3 & 0 \\ 3 & 3 & 0 \\ 3 & 3 & 0 \end{pmatrix}$$

Large where the window straddles the dark-to-bright edge, zero on the uniform region.

3. The edge moves, and the response moves with it: $Y = \begin{pmatrix} 0 & 3 & 3 \\ 0 & 3 & 3 \\ 0 & 3 & 3 \end{pmatrix}$

Shift the input, the feature map shifts the same way: convolution is **translation equivariant**.

Filters detect features



- A filter responds where the image locally looks like the filter: here, vertical and horizontal edges of a 4 (red: dark to bright, blue: bright to dark)
- Classical image processing designs such kernels by hand (edges, blur, sharpening). A convolutional network **learns** its kernels by backpropagation, like any other weights

A convolution is a constrained dense layer

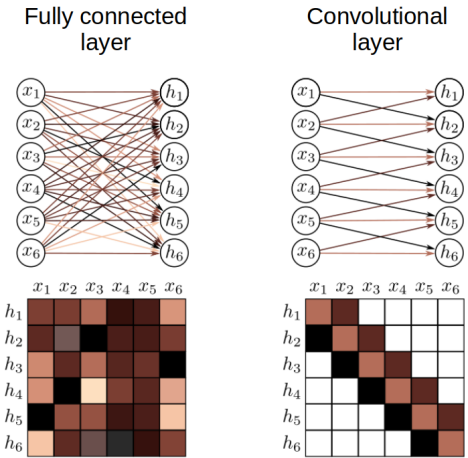


Image source: S. Prince, Understanding Deep Learning

Six inputs, six outputs; below, the weight matrices, one colour per value:

- **dense**: 36 independent weights
- **convolution** (kernel size 3): most weights are zero, the others repeat the same 3 numbers along the diagonals

Everything from lecture 2 still applies: forward pass, backpropagation, optimisers. Only the structure of W has changed.

Fewer parameters to learn from the same data: less overfitting.

Channels and filters

Channels: the input has a depth C_{in}: 1 for a grey image, 3 for colour (red, green, blue), and in a hidden layer as many as the previous layer had filters.

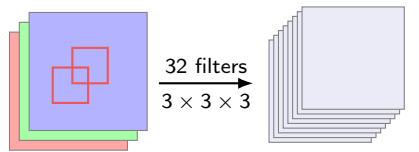
A filter spans all input channels: $k \times k \times C_{\text{in}}$ weights and one bias, giving one feature map.

A layer has C_{out} filters: output $H' \times W' \times C_{\text{out}}$.

Parameters of a convolutional layer:

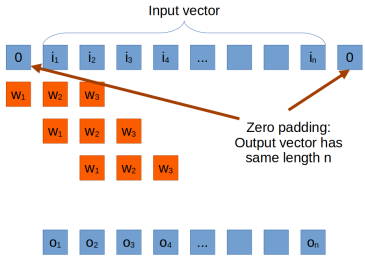
$$(k^2\, C_{\text{in}} + 1)\, C_{\text{out}}$$

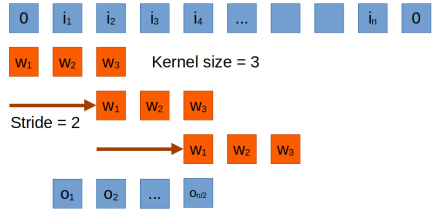
independent of the image size.



Layer	Input	Parameters
`Conv2D(32, 3)`	grey	$(9+1)\cdot 32 = 320$
`Conv2D(32, 3)`	RGB	$(27+1)\cdot 32 = 896$
`Conv2D(64, 3)`	32 maps	$(288+1)\cdot 64 = 18\,496$

Padding and stride





Padding: a border of zeros keeps the output at the input size (`padding="same"`). `"valid"` (the Keras default) means no padding: each layer shrinks the image by $k - 1$.

Stride s**:** move the kernel by s pixels at a time; the output is s times smaller in each direction.

Output size, with padding p on each side:

$$n_{\text{out}} = \left\lfloor \frac{n_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

Pooling

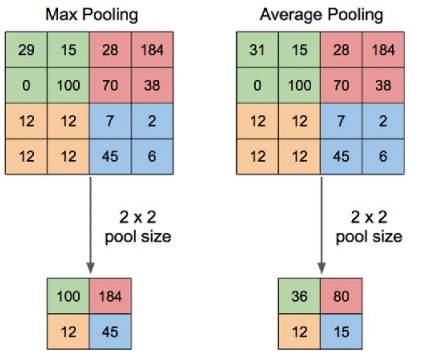


Image source: J. Phys.: Conf. Ser. 1201, 012052

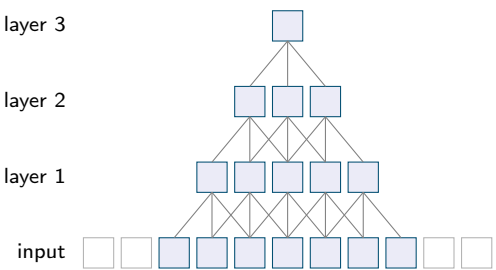
Max pooling 2×2: keep the largest value of each 2×2 block

- halves width and height: 4 times fewer values
- no parameters
- keeps *whether* a feature was found, forgets exactly *where*: a little tolerance to small shifts

Average pooling takes the mean; inside the network max pooling usually works better.

Global average pooling: one average per feature map over the whole image, often used before the output layer instead of `Flatten`.

Receptive field: depth builds up context



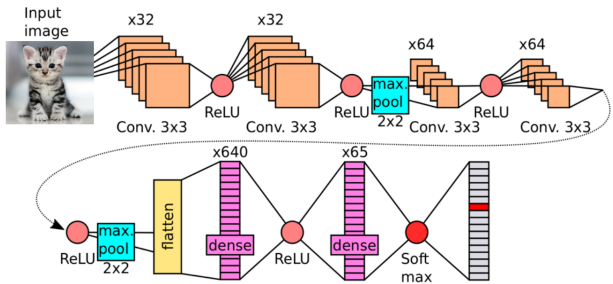
The **receptive field** of a unit: the part of the input it depends on.

- one 3×3 layer: 3×3 pixels
- three stacked 3×3 layers: 7×7 pixels
- each 2×2 pooling doubles the growth that follows

Early layers see small patches (edges), deep layers see large regions (parts, objects).

Why small kernels? Three 3×3 layers cover 7×7 with $3 \cdot 9 = 27$ weights per channel pair instead of 49, and with three nonlinearities instead of one (VGG, 2014).

A typical convolutional network



- **Feature extraction:** blocks of convolutions with ReLU, then pooling; the images get smaller, the number of filters grows (32, 64, 128, ...)
- **Classification head:** `Flatten` (or global average pooling), one or two dense layers, softmax output
- Trained end to end with the same loss and optimiser as in lectures 1 and 2

Your turn: shapes and parameters

The network of practical 3, for 28×28 MNIST images, no padding (`padding="valid"`):

Layer	Output shape	Parameters
`Input`	$28 \times 28 \times 1$	0
`Conv2D(32, 3, activation="relu")`	?	?
`MaxPooling2D(2)`	?	?
`Conv2D(64, 3, activation="relu")`	?	?
`MaxPooling2D(2)`	?	?
`Flatten`	?	?
`Dense(128, activation="relu")`	128	?
`Dense(10, activation="softmax")`	10	?
Total		?

Your turn: shapes and parameters

The network of practical 3, for 28×28 MNIST images, no padding (`padding="valid"`):

Layer	Output shape	Parameters
`Input`	$28 \times 28 \times 1$	0
`Conv2D(32, 3, activation="relu")`	$26 \times 26 \times 32$	$(9 + 1) \cdot 32 = 320$
`MaxPooling2D(2)`	$13 \times 13 \times 32$	0
`Conv2D(64, 3, activation="relu")`	$11 \times 11 \times 64$	$(9 \cdot 32 + 1) \cdot 64 = 18\,496$
`MaxPooling2D(2)`	$5 \times 5 \times 64$	0
`Flatten`	1 600	0
`Dense(128, activation="relu")`	128	$1\,600 \cdot 128 + 128 = 204\,928$
`Dense(10, activation="softmax")`	10	1 290
Total		225 034

91 % of the parameters sit in the first dense layer; the two convolutional layers together have fewer than 19 000. Pooling of an odd size rounds down: $\lfloor 11/2 \rfloor = 5$.

Dense versus convolutional on MNIST

```
model = keras.Sequential([
    layers.Input(shape=(28, 28, 1)),
    layers.Conv2D(16, 3, activation="relu"),
    layers.MaxPooling2D(2),
    layers.Conv2D(32, 3, activation="relu"),
    layers.MaxPooling2D(2),
    layers.Flatten(),
    layers.Dense(10, activation="softmax"),
])
```

Compiled and trained as before. Images need a channel axis: shape $(n, 28, 28, 1)$.

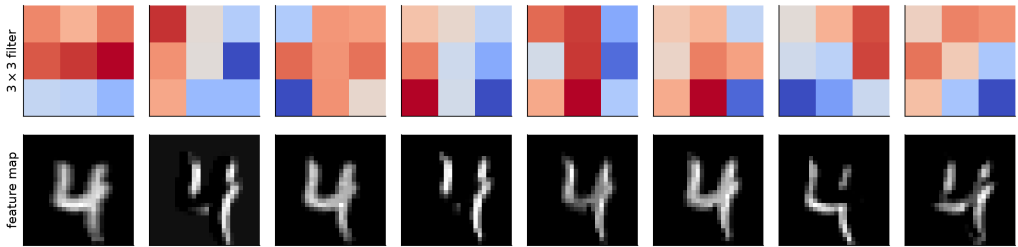
Same data, 5 epochs, batch 64, Adam:

Model	Param.	Test acc.
Dense(16), lecture 1	12 730	93.7 %
CNN on the left	12 810	**98.2 %**
Dense 64–32, practical 1	52 650	96.5 %
CNN of practical 3	225 034	98.6 %

Same number of parameters: **3.5 times fewer errors** (1.8 % instead of 6.3 %).

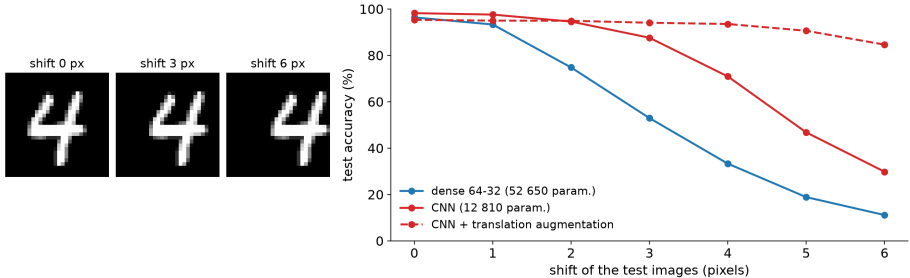
Price: more computation per parameter, since every weight is used at every position: 3 to 4 times slower on a laptop CPU.

What has the network learned?



- Top: 8 of the 32 learned 3×3 filters of the first layer of the practical-3 network. Bottom: their feature maps for a 4, after ReLU
- First-layer filters are edge and stroke detectors, similar to the hand-made ones; deeper layers combine them into larger patterns
- In networks trained on photos, the deep layers respond to textures, then to object parts: eyes, wheels, faces (Zeiler & Fergus, 2014)

How robust to shifts?



- Test digits moved right by 0 to 6 pixels, training data unchanged: the dense network collapses, the CNN holds better, but not fully
- Convolutions are equivariant and pooling adds a little invariance, but the dense head still sees positions
- **Data augmentation** (lecture 2) with random shifts of up to 20 % makes it robust: about 93 % at 4 pixels

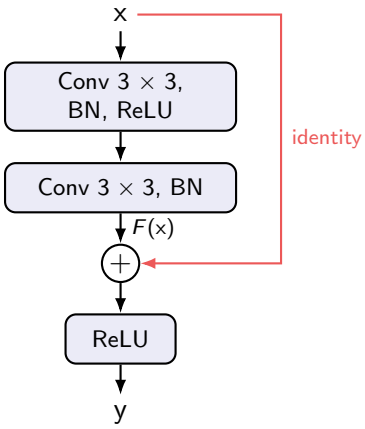
Deeper and deeper: the ImageNet years

ImageNet challenge: 1.2 million photos, 1 000 classes; error = true class not among the 5 best guesses.

Network	Year	Layers	Parameters	Top-5 error
LeNet-5 (LeCun et al.)	1998	7	60×10^3	(MNIST: 0.9 %)
AlexNet (Krizhevsky, Sutskever, Hinton)	2012	8	60×10^6	15.3 % (next best: 26.2 %)
VGG-16 (Simonyan & Zisserman)	2014	16	138×10^6	7.3 %
GoogLeNet (Szegedy et al.)	2014	22	7×10^6	6.7 %
ResNet-152 (He et al.)	2015	152	60×10^6	3.6 %
A trained human (Karpathy)	2014			5.1 %

What made the jump of 2012 possible: **GPUs**, a large labelled dataset, **ReLU**, **dropout**, **data augmentation**. Then: small 3×3 kernels, batch normalisation, and residual connections.

Residual connections



Problem: plain networks deeper than about 20 layers trained *worse*, even on the training set: vanishing gradients (lecture 2), hard optimisation.

Idea (He et al., 2015): each block learns only a correction to its input,

$$y = x + F(x) \quad \Rightarrow \quad \frac{\partial y}{\partial x} = 1 + \frac{\partial F}{\partial x}$$

The "1" carries the gradient through any number of blocks. A useless block can learn $F \approx 0$.

```
h = layers.Conv2D(64, 3, padding="same",
                  activation="relu")(x)
h = layers.Conv2D(64, 3, padding="same")(h)
y = layers.Activation("relu")(layers.Add()([x, h]))
```

ResNet: 152 layers. Now everywhere, transformers included (lecture 5).

Transfer learning: do not start from zero

The early layers of a network trained on ImageNet detect edges, textures and shapes that are useful for almost any image task.

1. take a pretrained network without its output layers
2. **freeze** it, train a new head on your data
3. optionally unfreeze and **fine-tune** everything with a small learning rate

Works with a few thousand, sometimes a few hundred, labelled images.

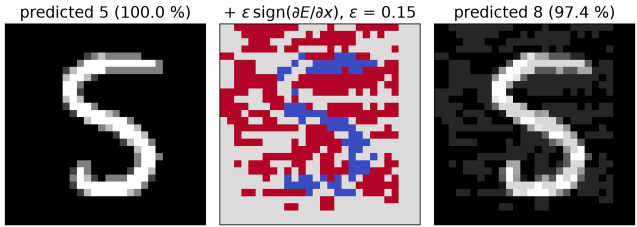
CIFAR-10 with only 2 000 training images: a small CNN from scratch reaches **47 %**; a frozen MobileNetV2 with a new output layer (only 12 810 trained weights) **80 %**.

```
base = keras.applications.MobileNetV2(
    input_shape=(96, 96, 3),
    include_top=False,
    weights="imagenet", pooling="avg")
base.trainable = False # freeze

model = keras.Sequential([
    layers.Input(shape=(32, 32, 3)),
    layers.Resizing(96, 96),
    layers.Rescaling(1 / 127.5, offset=-1),
    base,
    layers.Dense(10, activation="softmax"),
])
```

Images enlarged to 96×96, pixels scaled to $[-1, 1]$ as MobileNetV2 expects. `keras.applications`: ResNet50, EfficientNet, … For the **projects**: often the fastest route to a good baseline.

Adversarial examples

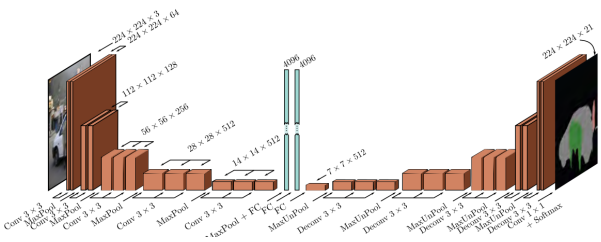


Backpropagate down to the **input pixels** to get $\partial E/\partial x$, then step in the direction that *increases* the loss (Goodfellow et al., 2014):

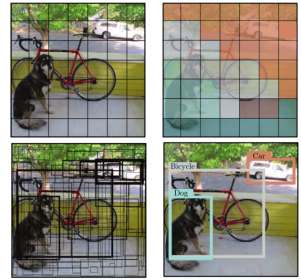
$$x' = x + \epsilon \operatorname{sign}\left(\frac{\partial E}{\partial x}\right)$$

- practical-3 CNN, $\epsilon = 0.15$: test accuracy 98.6 % $\rightarrow$ **about 67 %**
- random ± 0.15 noise: still 98.5 %. The direction matters, not the size
- on photos an invisible change suffices: a panda classified as a gibbon with 99 % confidence

Beyond classification



Semantic segmentation: one class per pixel. An **encoder** (convolutions, pooling) shrinks the image to a compact representation, a **decoder** (upsampling, convolutions) grows it back (SegNet, U-Net, 2015).
→ The encoder–decoder shape is the **autoencoder** of lecture 4.



Object detection (YOLO, 2016): a grid of cells, each predicting boxes, confidences and classes, in one pass: real time.

Image source: S. Prince, Understanding Deep Learning

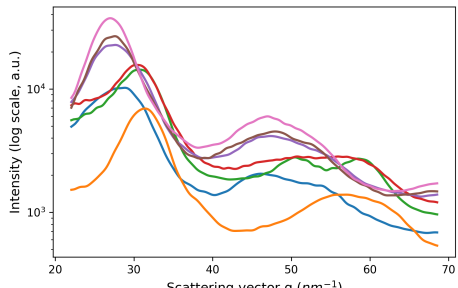
Convolutions in 1D and 3D

1D (`Conv1D`): signals and spectra

- audio, speech
- electrocardiograms, seismograms
- spectra and diffractograms
- DNA and protein sequences

3D (`Conv3D`): volumes and videos

- CT and MRI scans, tomography
- videos (two dimensions of space, one of time)
- simulation boxes



Small-angle X-ray scattering curves of nanoparticles, the input of a 1D CNN (research example, next slides)

Wherever nearby values belong together and a pattern can appear anywhere, convolutions help.

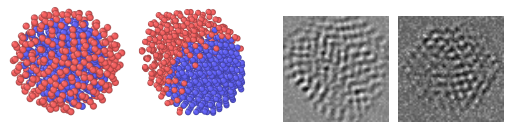
Research example: nanoparticles in the electron microscope

Question: silver–cobalt nanoparticles of a few nanometres are either **core–shell** or **Janus** (two-faced). Which one, how large, what composition, from one high-resolution TEM image?

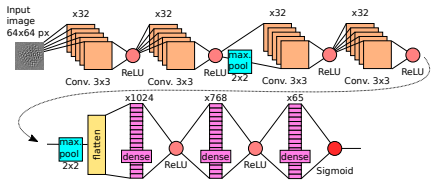
Problem: no labelled experimental images.

Solution: simulate the training set

1. molecular dynamics: many realistic particles
2. image simulation (multislice) with aberrations and noise: up to a million labelled images
3. train a CNN on simulated images, apply it to real ones

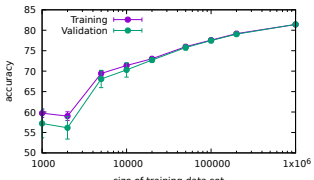


core–shell and Janus particles, and their simulated images

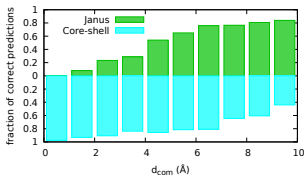


64×64 image $\rightarrow$ four 3×3 convolutional layers, two poolings $\rightarrow$ dense layers $\rightarrow$ class, size, composition

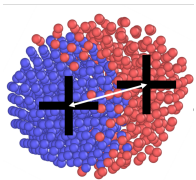
Research example: what limits the accuracy



Accuracy grows with the logarithm of the dataset size: below 60 % with 10^3 images, 81 % with 10^6



Janus particles whose two halves have close centres of mass (small d_{com}) look like core–shell ones



Distance Ag-Co centers of mass

- Simulation provides unlimited, perfectly labelled data: the physics of image formation is learned from it
- The errors are physical, not random: some particles are ambiguous in projection, whatever the network
- The model only knows what the simulation knows: validation on real images remains essential

Summary and outlook

Convolution	a small filter slides over the image: local, shared weights, translation equivariant
A layer	C_{out} filters of $k \times k \times C_{\text{in}}$ weights: $(k^2 C_{\text{in}} + 1)\, C_{\text{out}}$ parameters
Sizes	$n_{\text{out}} = \lfloor (n_{\text{in}} + 2p - k)/s \rfloor + 1$; pooling halves, padding `"same"` keeps
Architecture	convolution and pooling blocks, then a dense head; receptive fields grow with depth
Why it works	prior knowledge in the architecture: 3.5 times fewer errors on MNIST, same parameter count
Deep networks	residual connections; transfer learning from pretrained models
Limits	shifts need augmentation; adversarial examples

Next session, practical 3: convolutional networks on MNIST: filters, pooling, dropout, batch normalisation, kernel sizes. Then **multiple-choice test 2**.

Lecture 4: autoencoders and generative models: networks that compress and reconstruct their input, and that generate new images.