

Convolutional Neural Networks

From Pixels to Learned Features

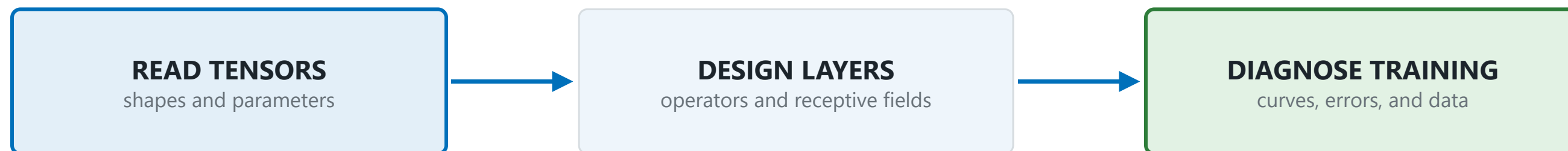
Matouš Cejnek

Department of Instrumentation and Control Engineering
Faculty of Mechanical Engineering, CTU in Prague

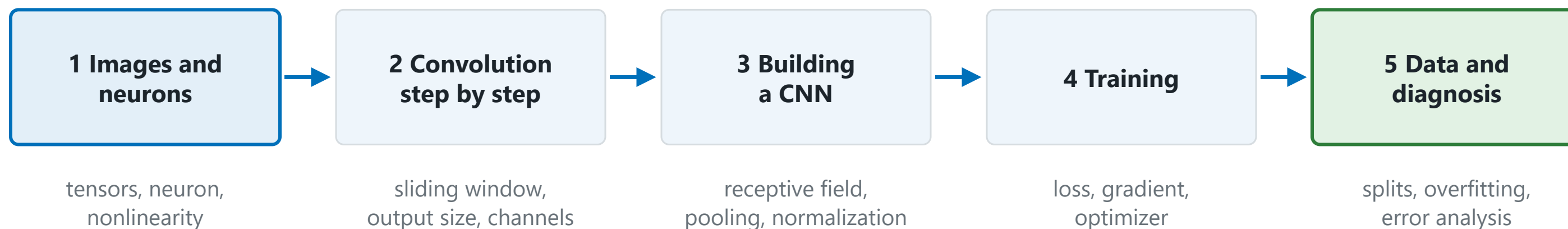
Why This Lecture

Part 1 · Images and neurons

- CNNs transform images into hierarchies of features.
- The goal is to understand tensor shapes, parameters, and gradient flow.
- By the end, you will be able to design and diagnose a small CNN.



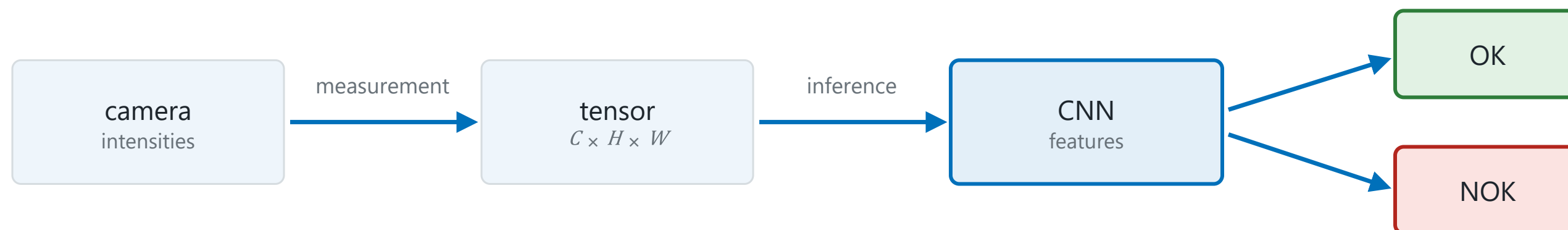
- Five parts; each one builds only on the previous ones.
- Every new operation is first shown on a tiny numeric example.
- Check questions close each block.



From Measurement to Decision

Part 1 · Images and neurons

- A camera provides an array of intensities, not objects.
- The model must separate changes in the object from changes in image acquisition.
- The output must match the task and its decision tolerance.



Four Typical Outputs

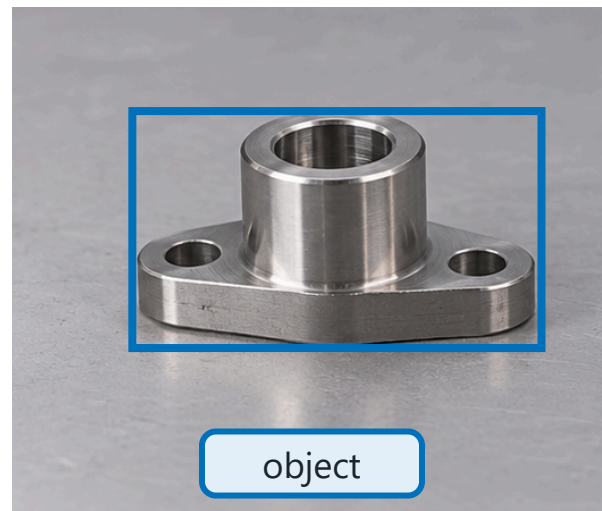
Part 1 · Images and neurons

- Classification: one label (or a vector of independent attributes).
- Detection: a variable number of boxes, each with a class.
- Segmentation: a class for every pixel. Regression: continuous quantities.

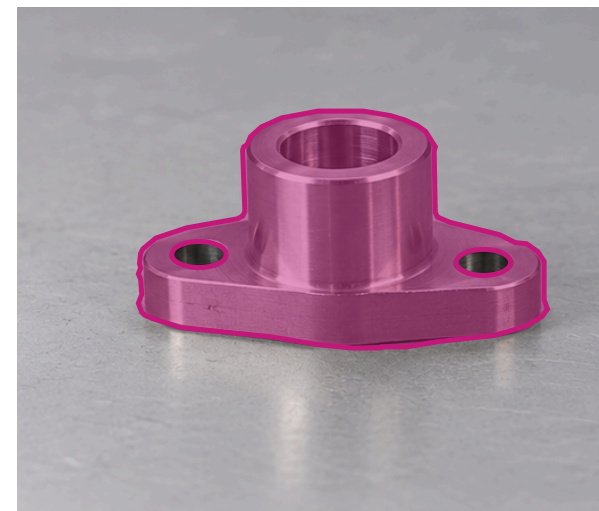
CLASSIFICATION



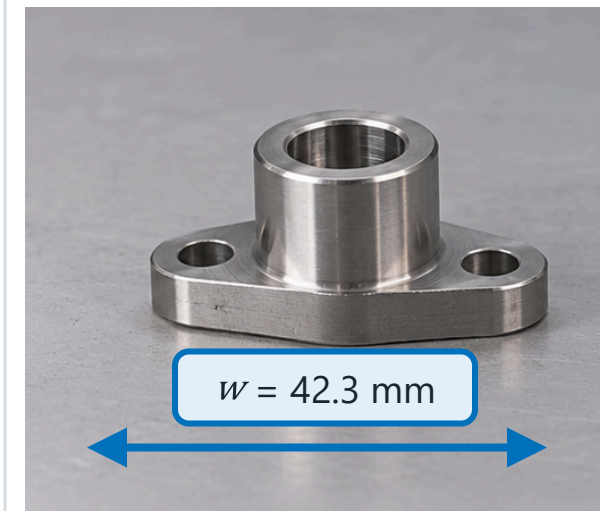
DETECTION



SEGMENTATION



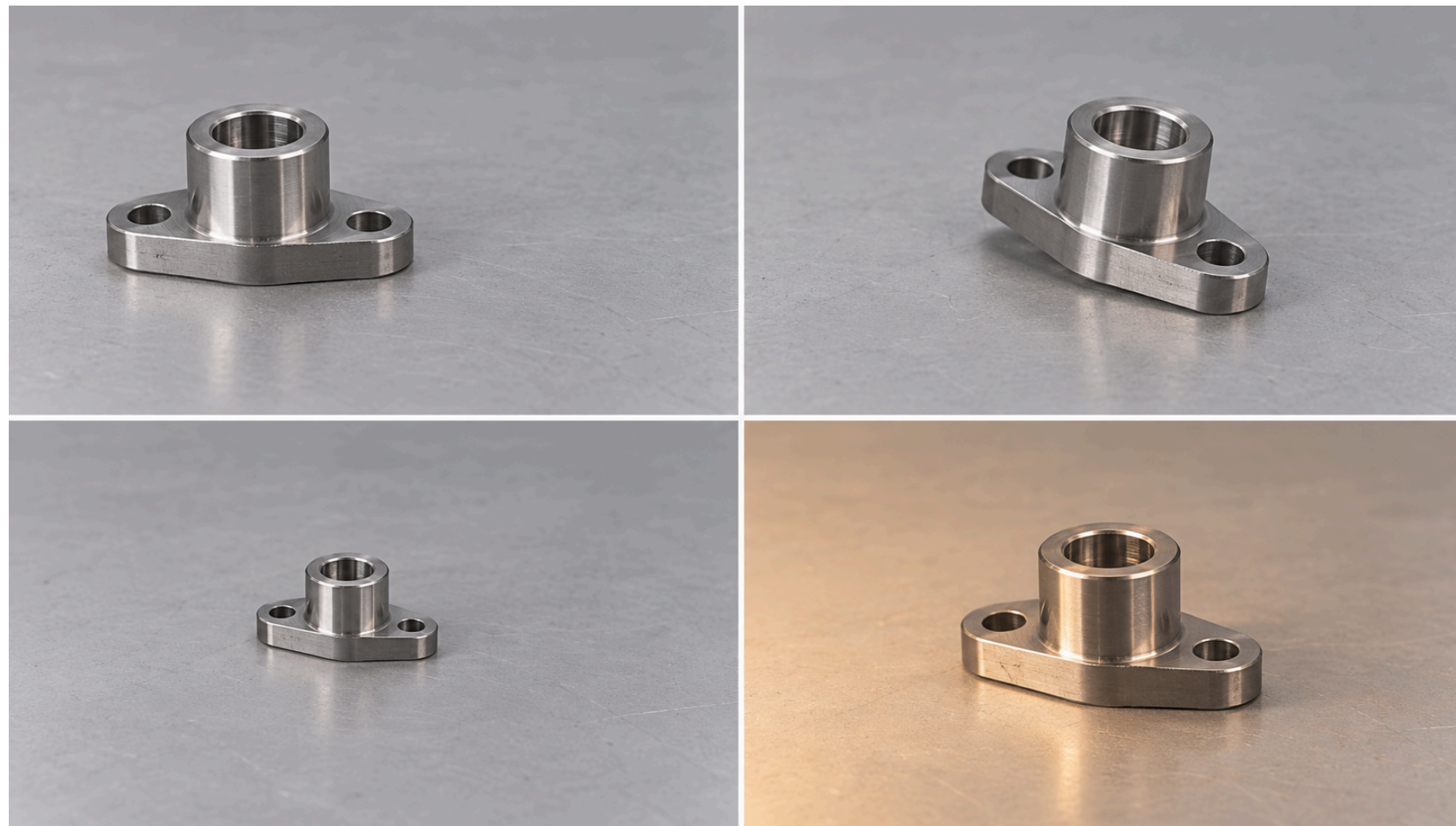
REGRESSION



What Changes in an Image?

Part 1 · Images and neurons

- Position, scale, rotation, and deformation of the object.
- Illumination, exposure, noise, and background.
- Occlusion and within-class variability.



Check question

Discuss in pairs

What should remain invariant when a product is inspected by a camera?

Give one desirable and one undesirable invariance.

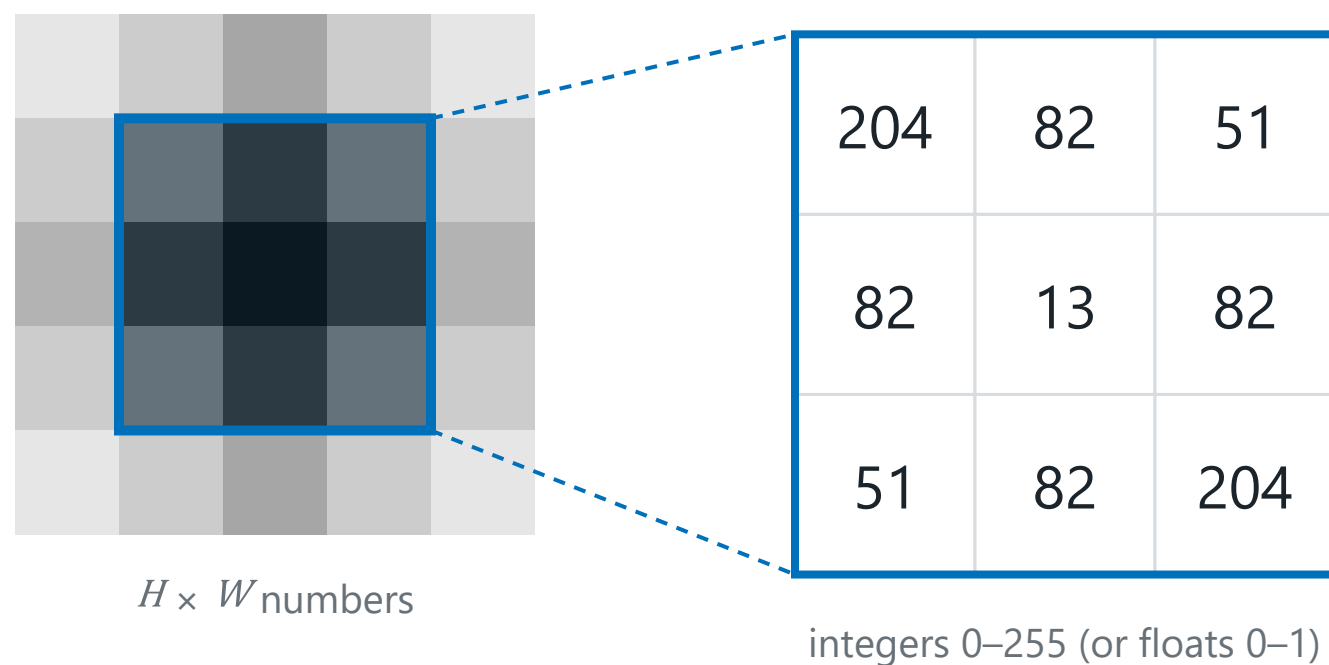


An Image Is a Tensor

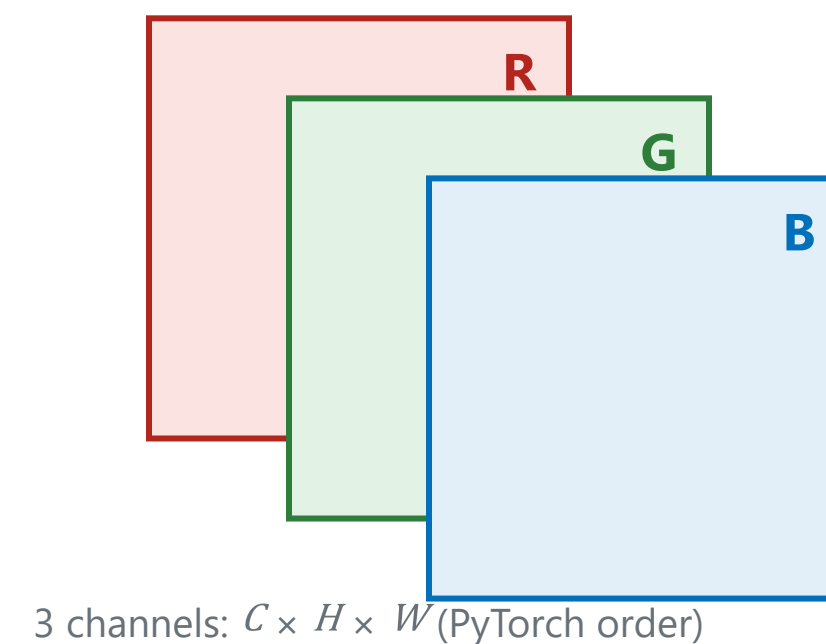
Part 1 · Images and neurons

- A grayscale image is a matrix of numbers, one per pixel.
- A colour image adds a third axis: channels. In PyTorch the order is $C \times H \times W$.
- Everything a CNN does is arithmetic on such arrays.

grayscale image



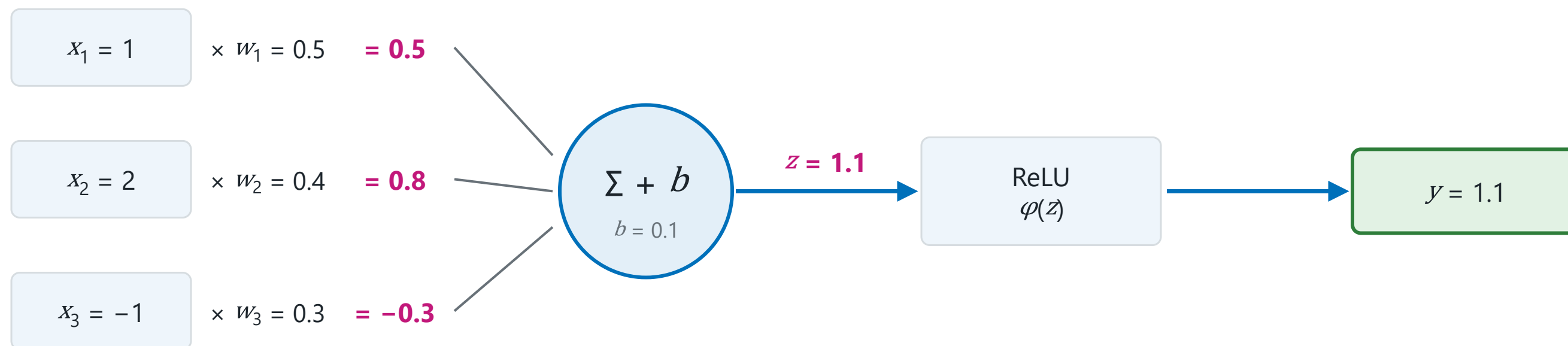
colour image



A Single Neuron

Part 1 · Images and neurons

- Weighted sum of inputs plus a bias: $z = \mathbf{w}^T \mathbf{x} + b$.
- An activation function φ turns z into the output $y = \varphi(z)$.
- Learning means finding $\mathbf{w}$ and b that reduce the loss.

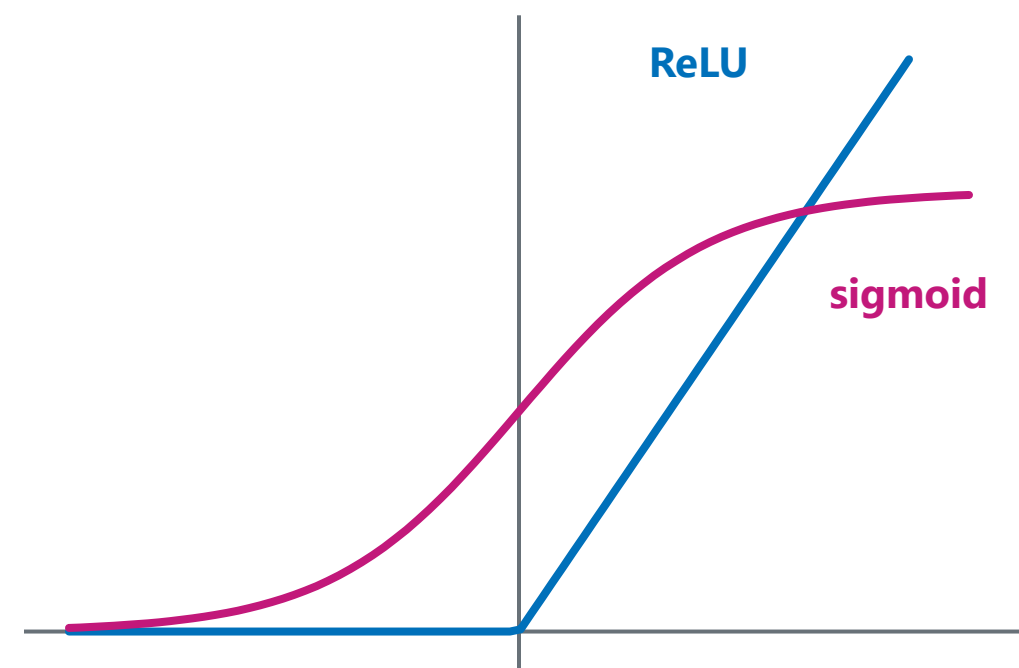
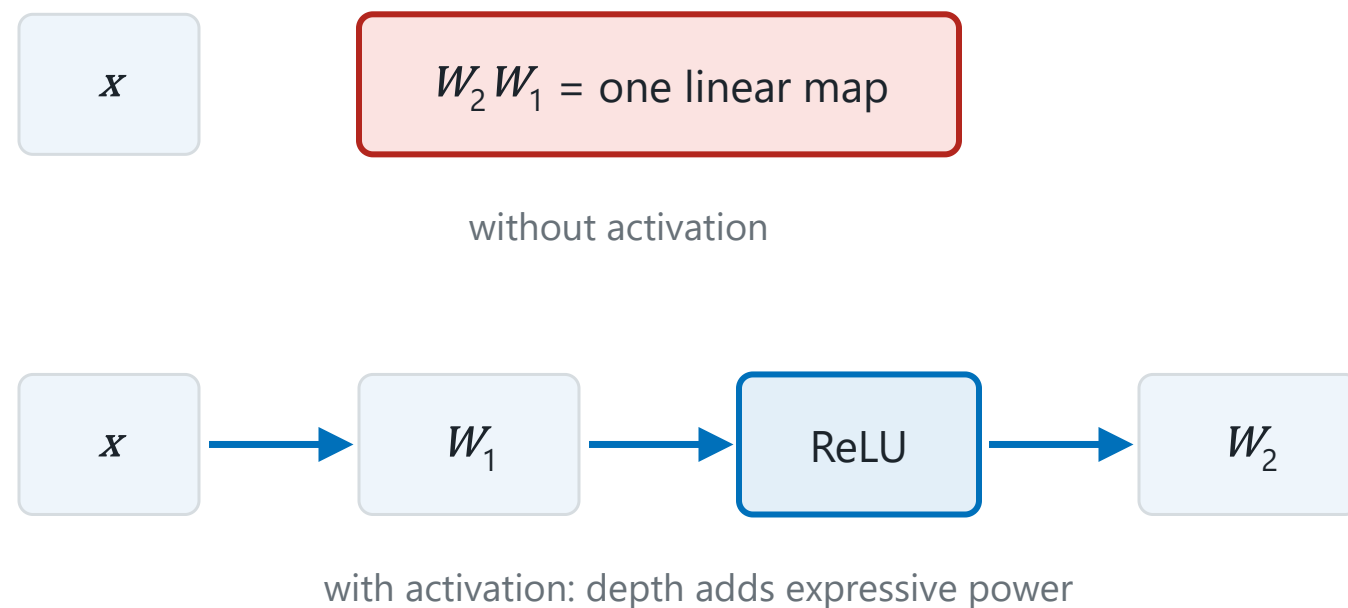


$$z = 0.5 + 0.8 - 0.3 + 0.1 = 1.1 \quad y = \max(0, z) = 1.1$$

Why We Need Nonlinearity

Part 1 · Images and neurons

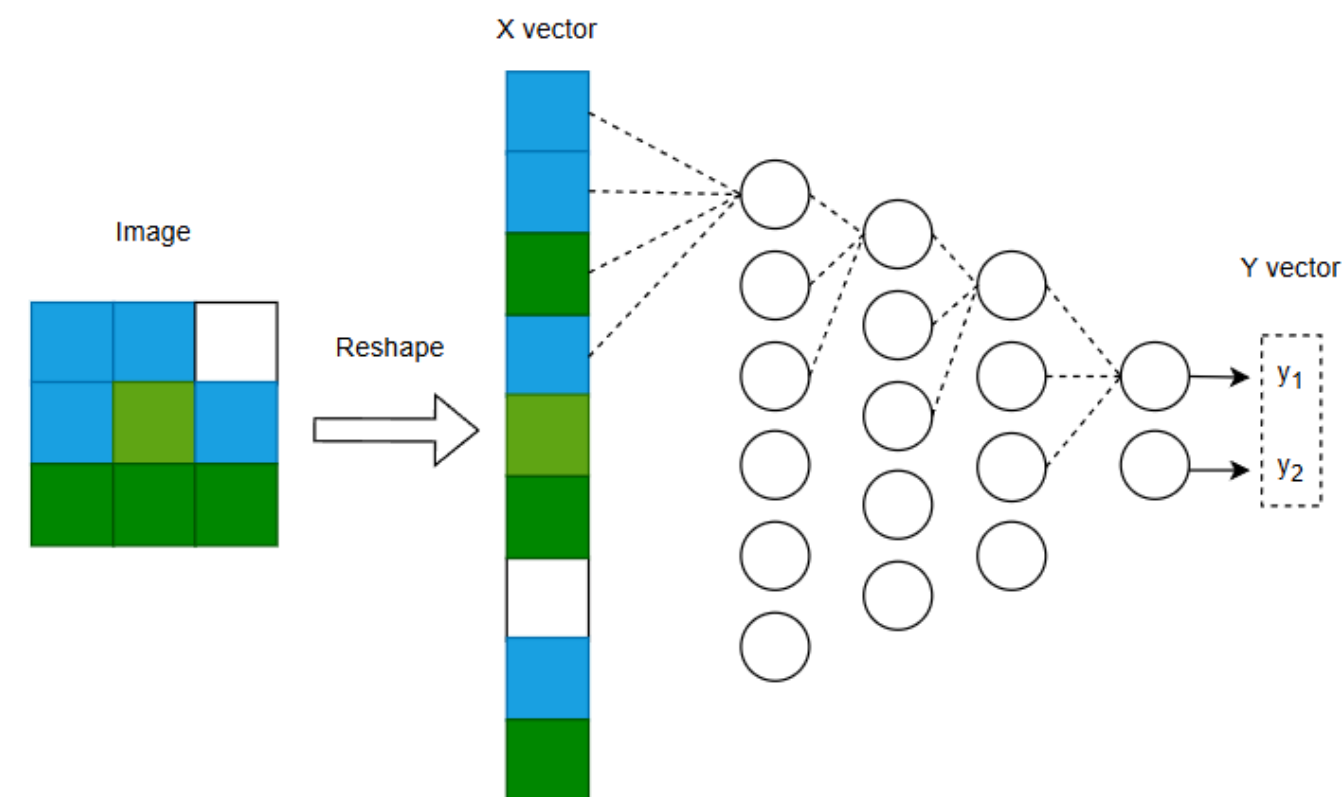
- Two linear layers in a row are again one linear map: $W_2 W_1 = W$.
- An activation between them breaks this collapse; then depth adds expressive power.
- ReLU = $\max(0, z)$ is the default in CNNs: cheap, with well-behaved gradients.



Fully Connected Networks on Images

Part 1 · Images and neurons

- The image is flattened into one long vector.
- Every input pixel is connected to every unit of the next layer.
- $224 \times 224 \times 3$ inputs $\rightarrow$ 1000 units: **150 million** weights in one layer.
- Flattening also throws away which pixels are neighbours.

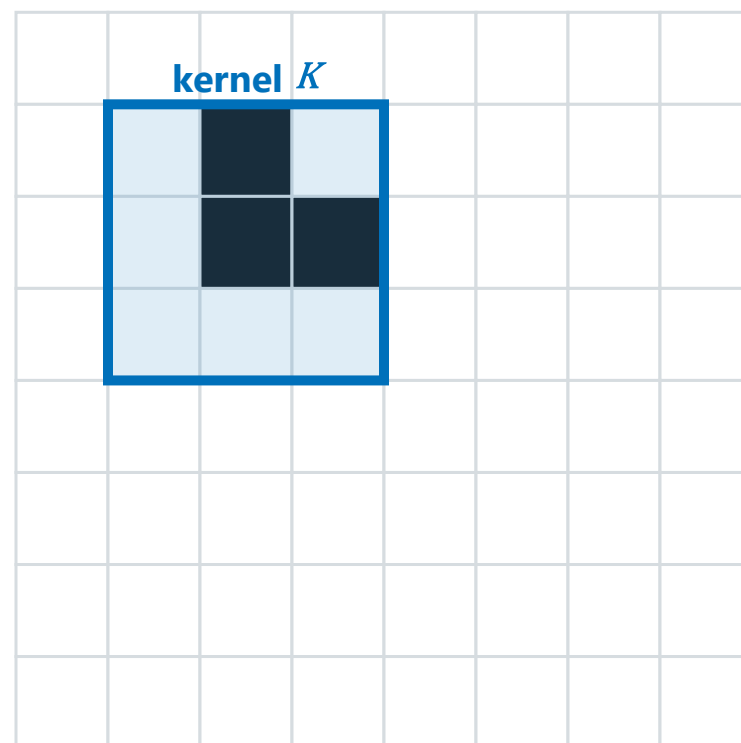


The Same Pattern at Another Position

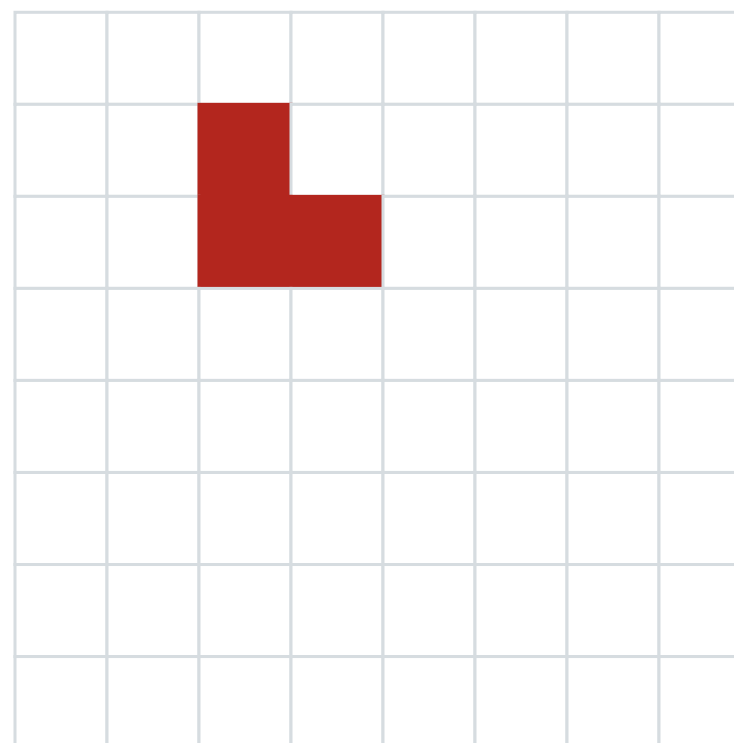
Part 1 · Images and neurons

- A fully connected layer has a separate weight for every pixel index.
- A pattern seen at position A teaches nothing about position B.
- We want one detector that is reused everywhere in the image.

input: the same pattern moves



fully connected: weights touched

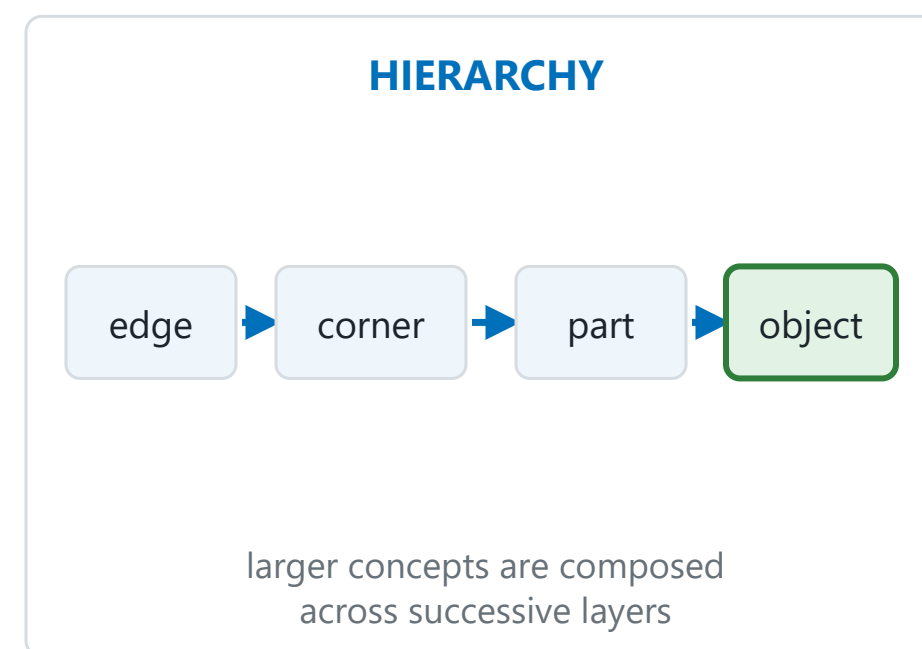
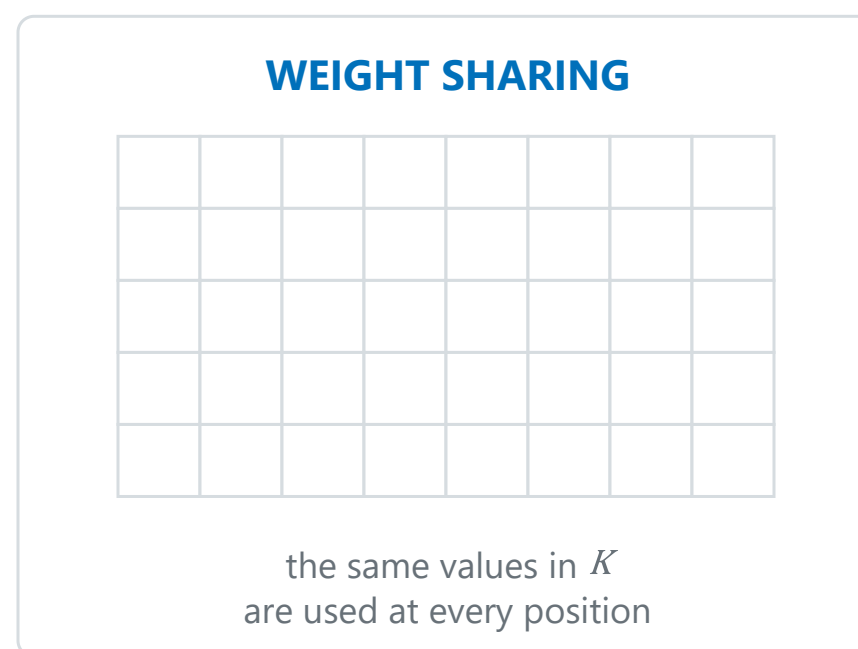
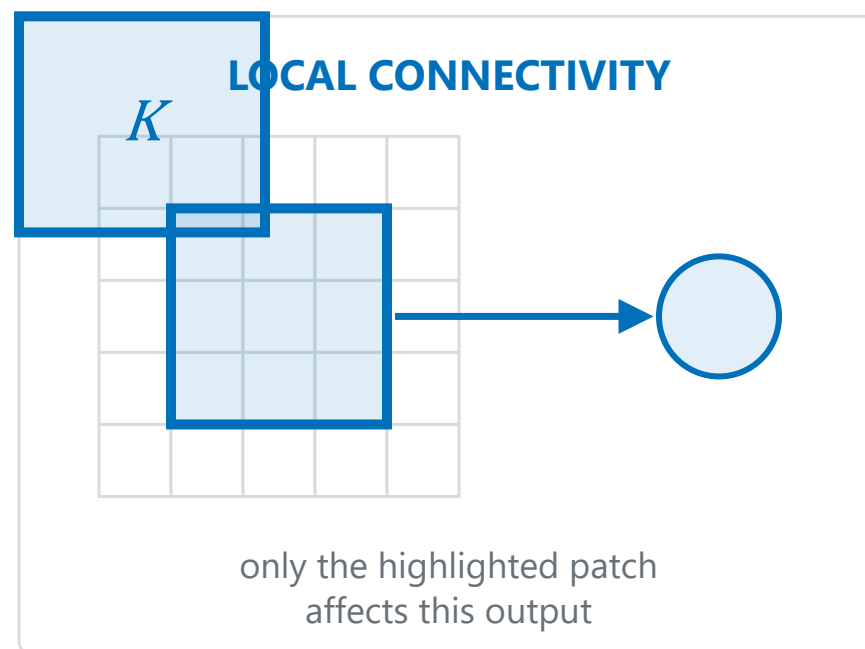


position 1: FC weights used 3

convolution:
one kernel for all positions

fully connected:
learns each position anew

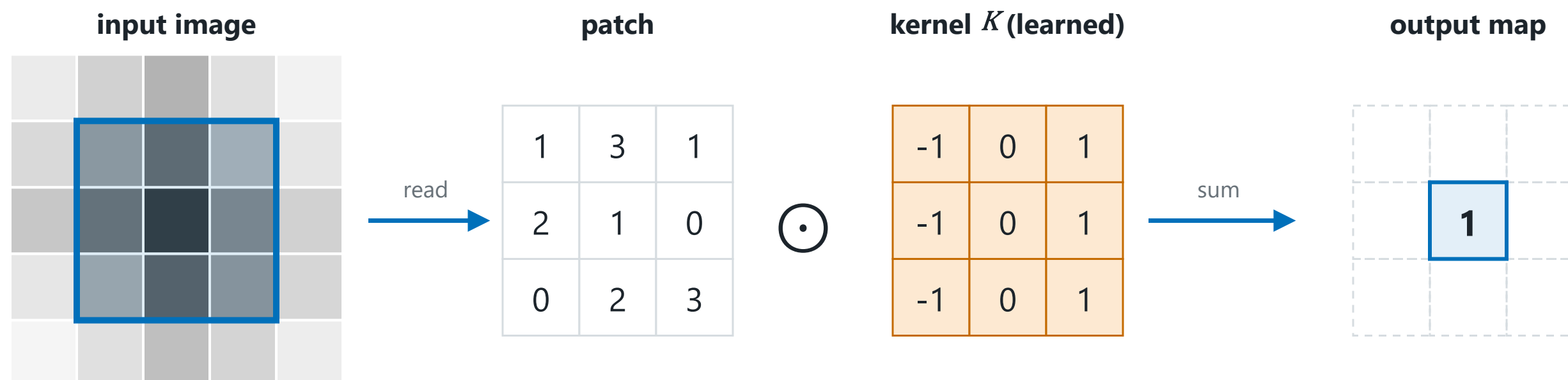
- Local connectivity: an output depends on a small neighbourhood only.
- Weight sharing: the same small kernel is applied at every position.
- Hierarchy: stacked layers compose edges into parts and parts into objects.



Convolution as a Local Operator

Part 2 · Convolution step by step

- Take a small window of the input, e.g. 3×3 pixels.
- Multiply it cell by cell with a kernel of the same size and add up the products.
- The result is one number of the output map. The kernel values are learned weights.



$\odot$: multiply matching cells, then add all nine products

The Sliding Window in 1D

Part 2 · Convolution step by step

Start with a 1D signal: the kernel $w = [-1, 0, 1]$ is placed over three neighbouring inputs, gives one output, and then moves one cell to the right.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6
input x	1	1	1	4	4	4	1

output y	0	3	3	0	-3
------------------------------	---	---	---	---	----

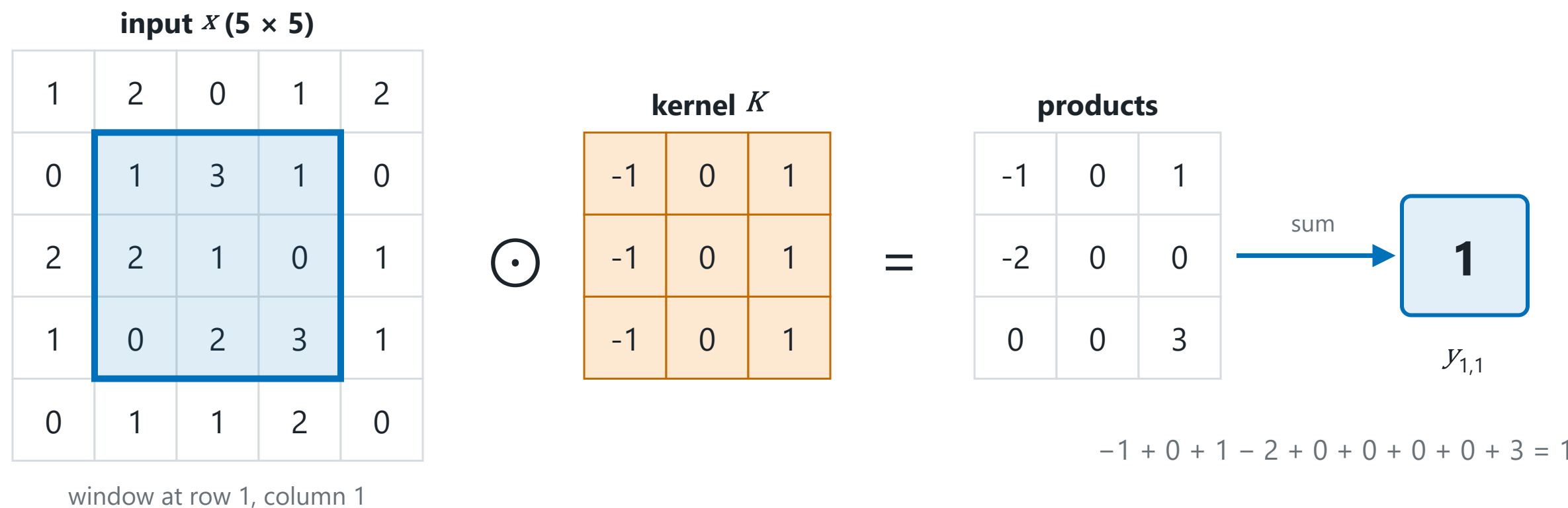
7 inputs, kernel of 3
⇒ 5 window positions
⇒ 5 outputs

non-zero only where
the signal changes

One Output Value in 2D

Part 2 · Convolution step by step

- In 2D the window is a 3×3 patch; the kernel is a 3×3 matrix.
- Multiply the 9 pairs of matching cells and add them: one number.
- In general: $y_{i,j} = \sum_u \sum_v x_{i+u, j+v} \cdot k_{u,v}$



The Sliding Window in 2D

Part 2 · Convolution step by step

The window moves left to right, then down by one row. Each position fills one cell of the output; the kernel never changes.

input 5×5

1	2	0	1	2
0	1	3	1	0
2	2	1	0	1
1	0	2	3	1
0	1	1	2	0

output 3×3

1	-3	-1
3	1	-4
1	2	-2

kernel K

-1	0	1
-1	0	1
-1	0	1

the same kernel
at every position

left to right, top to bottom: $3 \times 3 = 9$ positions give a 3×3 feature map

What Does a Kernel Detect?

Part 2 · Convolution step by step

- The output is large where the input patch looks like the kernel.
- $[-1, 0, 1]$ in every row compares right with left: a vertical-edge detector.
- A CNN does not get such kernels by hand; it learns them from data.

input: dark | bright

1	1	1	8	8	8
1	1	1	8	8	8
1	1	1	8	8	8
1	1	1	8	8	8

horizontal-edge kernel

-1	-1	-1
0	0	0
1	1	1



feature map

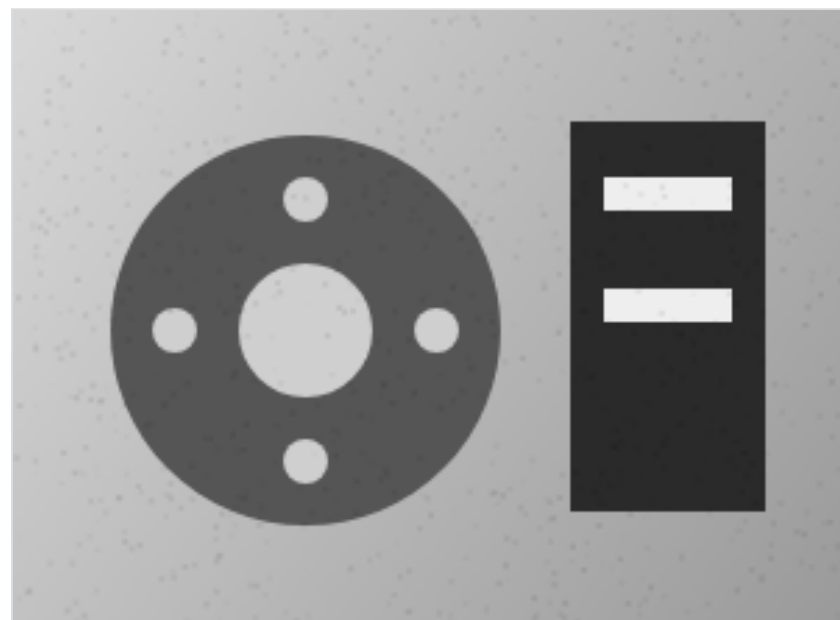
0	0	0	0
0	0	0	0

no response: pattern not present

The Same Idea on a Real Image

Part 2 · Convolution step by step

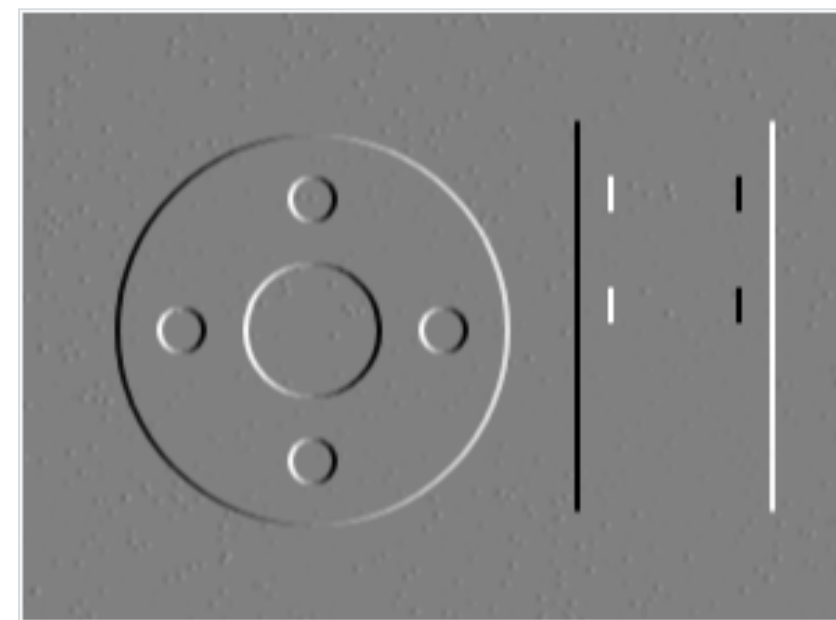
- One 3×3 kernel slides over the whole image, row by row; its output is a *feature map*.
- Flat regions give values near zero (grey); edges give strong responses. Try other kernels, or drop your own image.
- Libraries actually compute cross-correlation (no kernel flip); for learned kernels the difference does not matter.



input (drop your own image here)

-1	0	1
-2	0	2
-1	0	1

vertical edge kernel



feature map (grey = 0)

What does the kernel

$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
respond to?

Horizontal edges: bright rows above dark rows give a large positive output, the reverse a negative one.



Output Size: Counting Window Positions

Part 2 · Convolution step by step

- The output has as many values as there are valid window positions.
- Without padding the window must stay inside the input.
- Each layer therefore shrinks the signal by $K - 1$.

input

x_0	x_1	x_2	x_3	x_4	x_5	x_6
-------	-------	-------	-------	-------	-------	-------

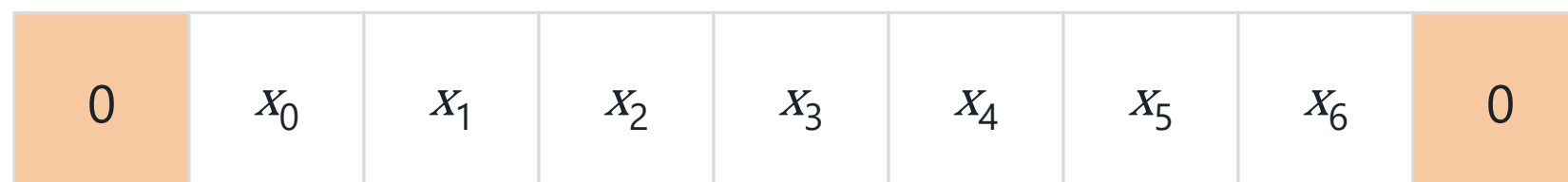
output

y_0	y_1	y_2	y_3	y_4
-------	-------	-------	-------	-------

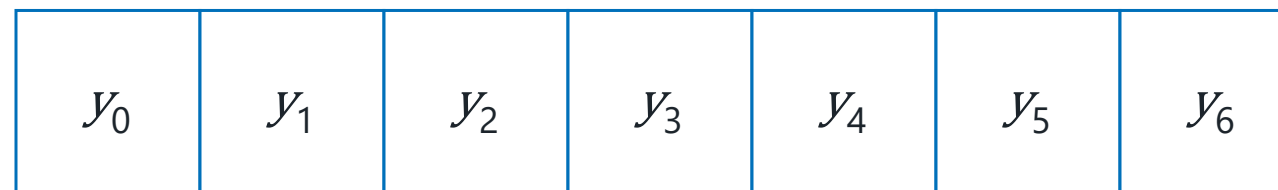
the window starts at $0, 1, \dots, N - K \Rightarrow O = N - K + 1 = 7 - 3 + 1 = 5$

- Padding adds P extra values (usually zeros) at each border.
- With $P = (K - 1)/2$ and stride 1 the output keeps the input size ("same").
- Border outputs partly see artificial values; keep this in mind when measuring near edges.

padding $P = 1$

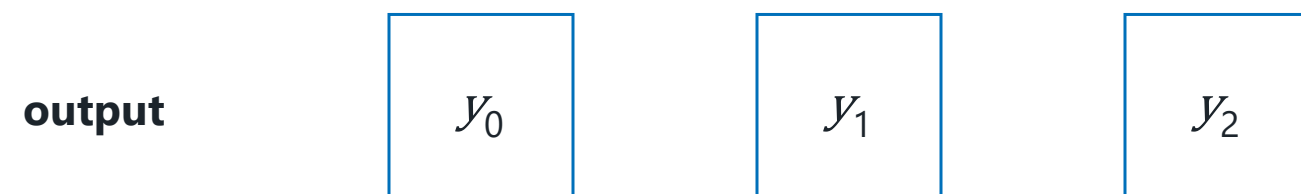
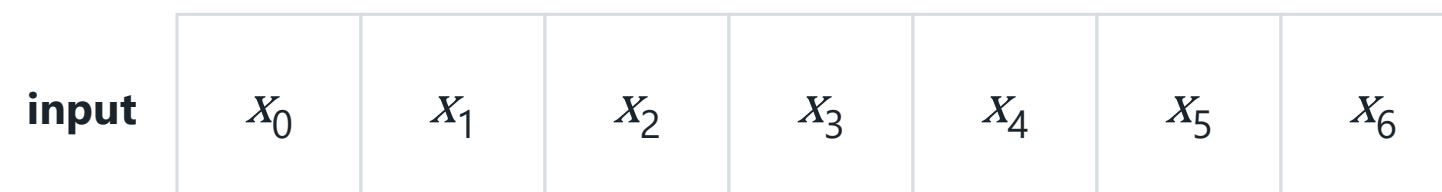


output



$$O = N + 2P - K + 1 = 7 + 2 - 3 + 1 = \mathbf{7} \text{ (same size as the input)}$$

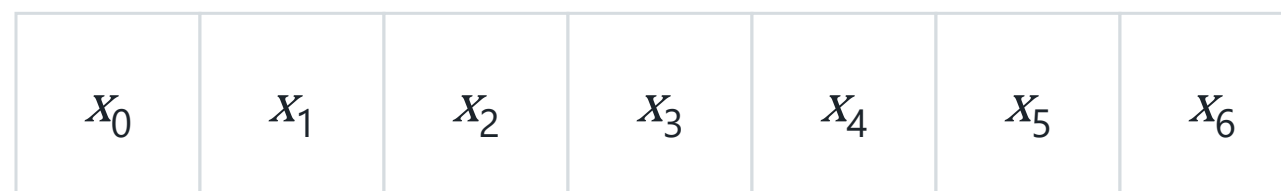
- Stride S is the step of the window; $S = 2$ skips every second position.
- It filters and downsamples in one operation, so later layers are cheaper.
- Large strides can skip small objects and cause aliasing.



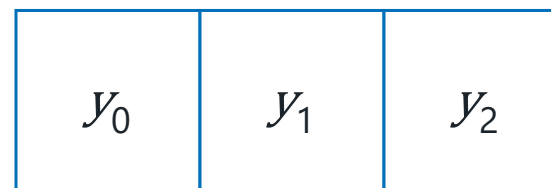
$$O = \lfloor (N + 2P - K) / S \rfloor + 1 = \lfloor 4 / 2 \rfloor + 1 = \mathbf{3}$$

- Dilation D inserts gaps between the kernel taps.
- The kernel covers a span of $D(K - 1) + 1$ inputs with only K weights.
- It enlarges the view without downsampling (common in segmentation).

input



output



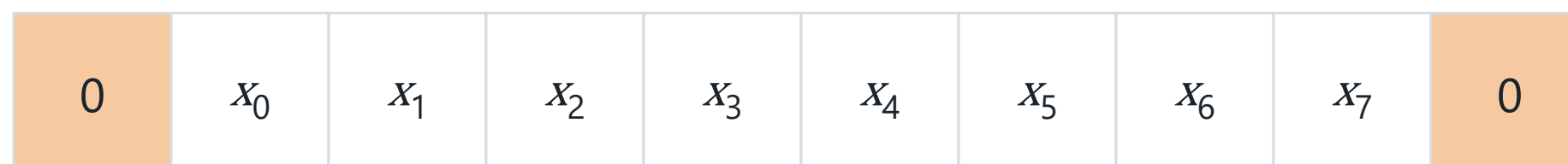
$$\text{span} = D(K - 1) + 1 = 5 \Rightarrow O = N - D(K - 1) = 7 - 4 = \mathbf{3}$$

The General Output-Size Formula

Part 2 · Convolution step by step

- All four effects combine into one formula, applied separately to height and width.
- Move the sliders and predict the output size before you look.

padding $P = 1$



output



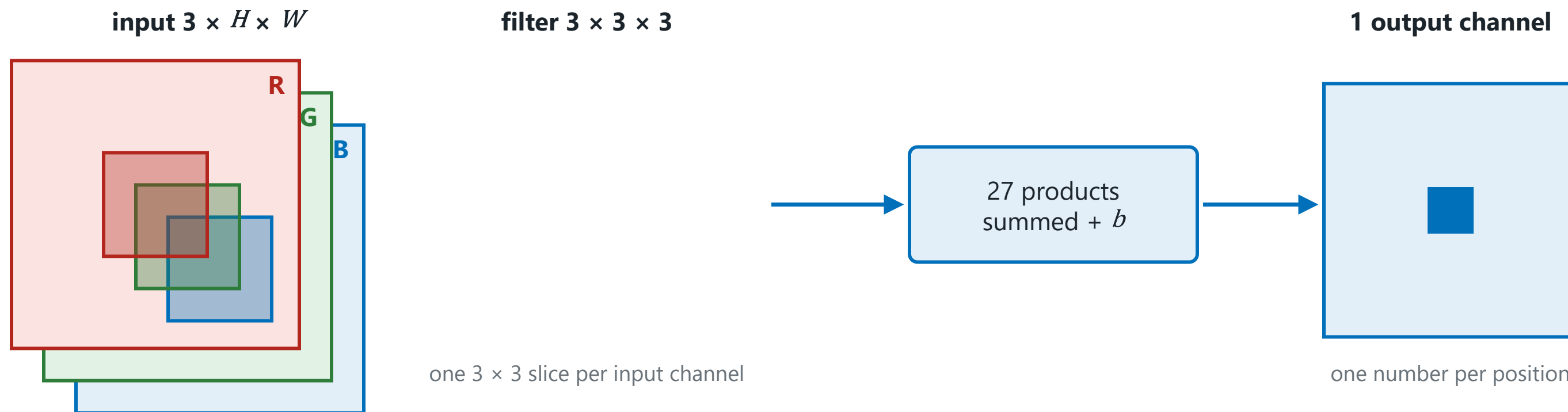
$$O = \lfloor (N + 2P - D(K - 1) - 1) / S \rfloor + 1 = \lfloor (8 + 2 - 2 - 1) / 2 \rfloor + 1 = 4$$

Input 32, kernel 3, padding 1, stride 2. What is the output size?

$$\lfloor (32 + 2 - 3) / 2 \rfloor + 1 = \lfloor 15.5 \rfloor + 1 = \mathbf{16}.$$



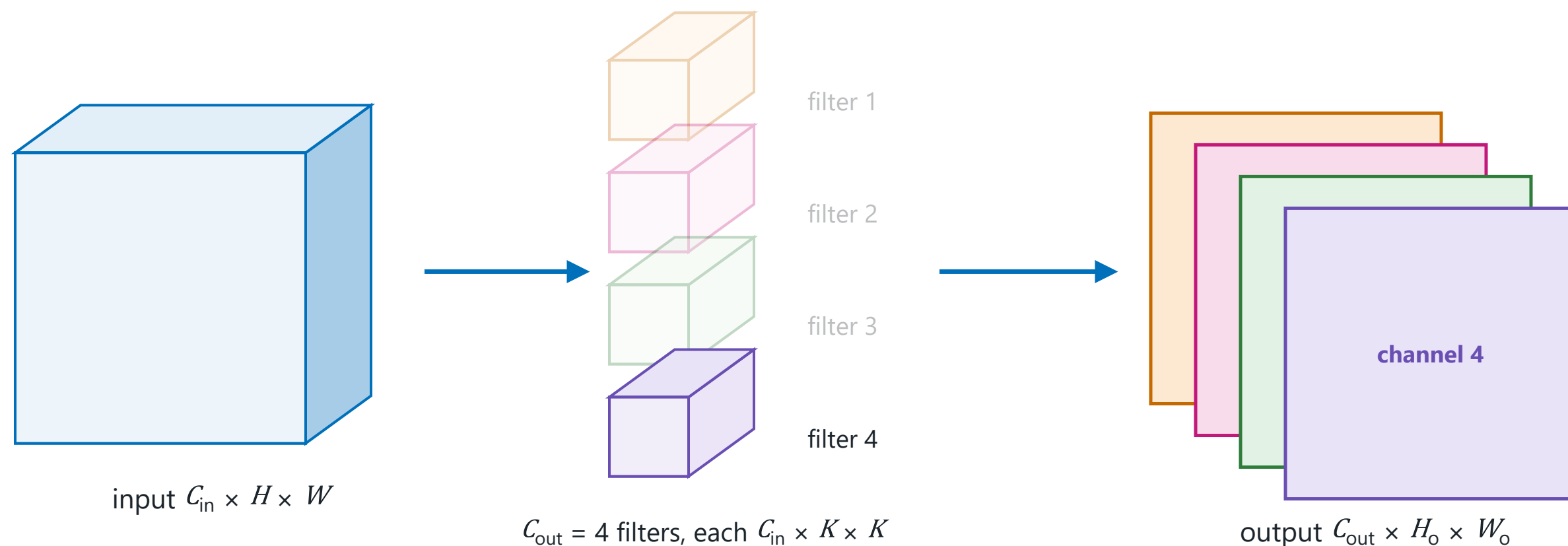
- An RGB input has three channels, so the filter also has three 3×3 slices.
- Filter depth always equals the number of input channels C_{in} .
- All $3 \cdot 3 \cdot 3 = 27$ products are summed: one filter gives one output channel.



Multiple Output Channels

Part 2 · Convolution step by step

- A layer has C_{out} independent filters, each looking for a different pattern.
- Their feature maps are stacked into the output tensor $C_{\text{out}} \times H_o \times W_o$.
- The next layer sees C_{out} as its number of input channels.



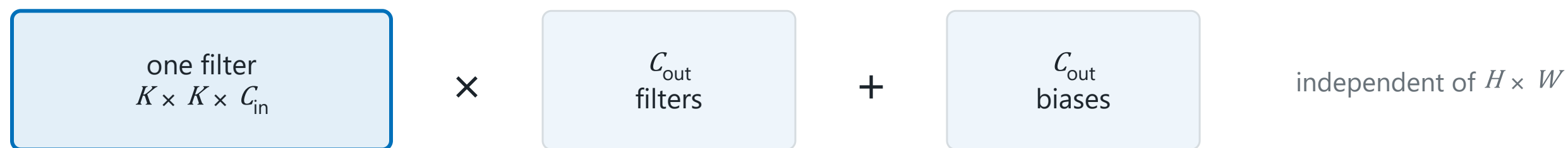
- A batch of images is a 4D tensor $N \times C \times H \times W$.
- The weight tensor of a convolution is $C_{\text{out}} \times C_{\text{in}} \times K \times K$.
- Spatial output size follows the formula; the channel count is chosen by the designer.

```
conv = nn.Conv2d(in_channels=3, out_channels=32,  
                 kernel_size=3, stride=2, padding=1)  
  
x.shape           # (N, 3, 128, 128)  
conv(x).shape     # (N, 32, 64, 64)    [(128+2-3)/2]+1 = 64  
conv.weight.shape # (32, 3, 3, 3)  
conv.bias.shape   # (32,)
```

Parameter Count

Part 2 · Convolution step by step

- Weights: C_{out} filters of size $K \times K \times C_{\text{in}}$, plus one bias per filter.
- The count does not depend on the image resolution $H \times W$.
- Thanks to locality and sharing, it is orders of magnitude smaller than a dense layer.



example: $3 \cdot 3 \cdot 3 \cdot 32 + 32 = 896$

conv $3 \rightarrow 32, 3 \times 3$



dense $224^2 \cdot 3 \rightarrow 1000$

150 528 000

bar length on a logarithmic scale

Check question

Estimate the parameters

How many weights are in a 3×3 convolution with 32 input and 64 output channels?

Without bias: $3 \cdot 3 \cdot 32 \cdot 64 = \mathbf{18\ 432}$; with bias 18 496.



Parameters, Computation, and Memory

Part 2 · Convolution step by step

- Few parameters do not mean little computation: each weight is used at every output position.
- Activations grow with resolution and must be stored for the backward pass.
- High-resolution layers dominate cost and memory; low-resolution layers dominate parameters.

example: 3×3 conv, $32 \rightarrow 64$ channels, output 64×64

PARAMETERS

$$K^2 C_{\text{in}} C_{\text{out}}$$

18.4 k

independent of image size

MULTIPLY-ADDS

$$H_{\text{o}} W_{\text{o}} \cdot K^2 C_{\text{in}} C_{\text{out}}$$

75.5 M

every weight is used at
every output position

ACTIVATIONS

$$C_{\text{out}} H_{\text{o}} W_{\text{o}}$$

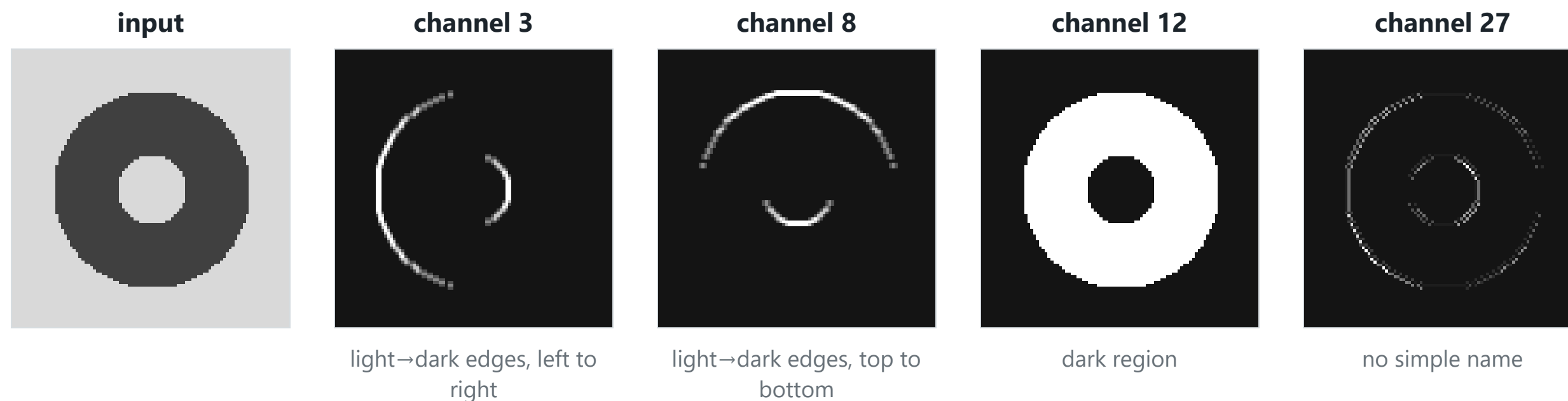
262 k

≈ 1 MB per image (float32),
stored for backprop

A Feature Map Is Not an Image

Part 3 · Building a CNN

- Each output channel is the response map of one learned detector.
- Some channels have a simple meaning (edges, blobs); many do not.
- Their meaning emerges together with the layers that read them.

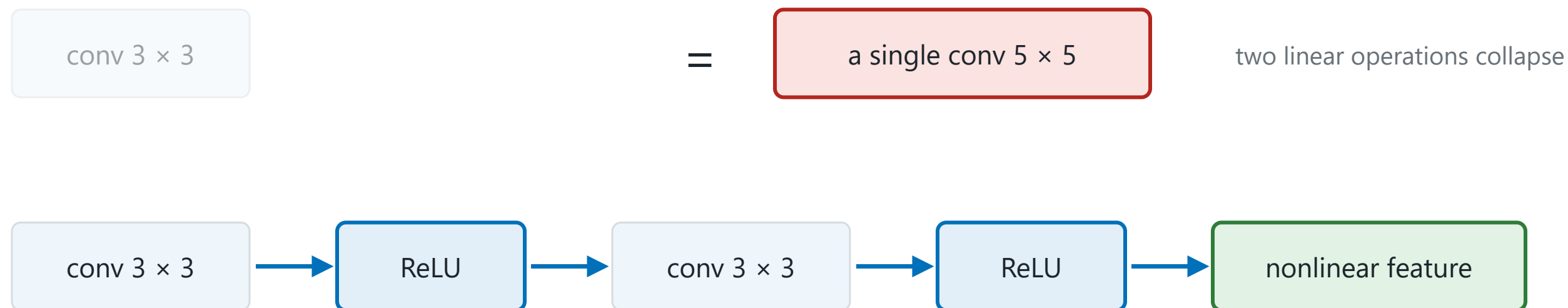


bright = strong response of that channel's filter (after ReLU); computed live in the browser

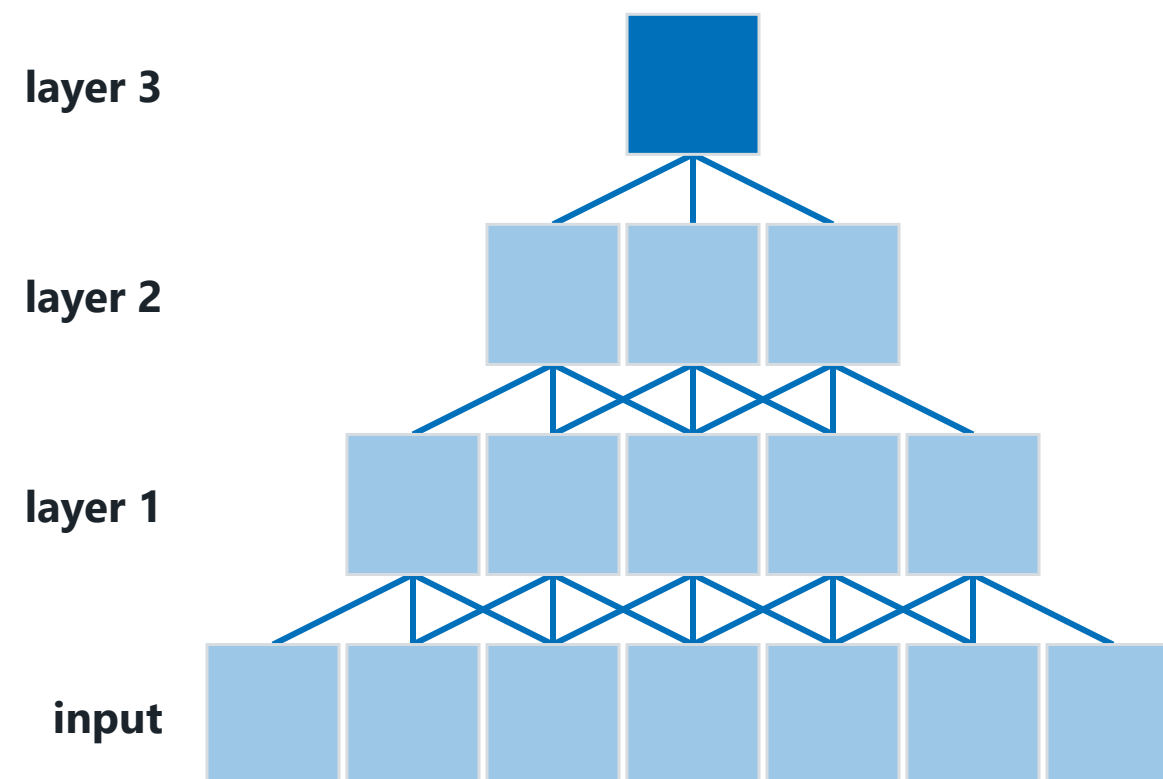
Stacking Layers Needs Nonlinearity

Part 3 · Building a CNN

- Convolution is linear: two convolutions in a row equal one larger convolution.
- A ReLU after each convolution prevents this collapse, exactly as for neurons.
- The standard block: convolution → normalization → ReLU.



- The receptive field is the input region that can influence one output element.
- One 3×3 layer sees 3 pixels per axis; two layers see 5; three see 7.
- Deep layers can therefore react to larger structures than any single kernel.

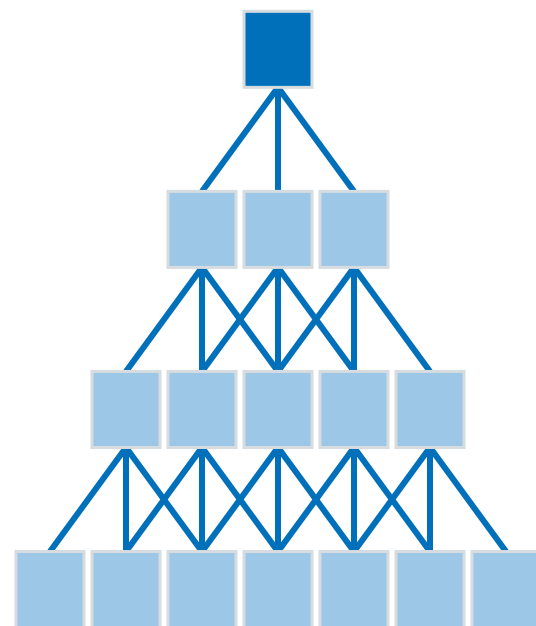


one layer-3 unit sees
7 input pixels

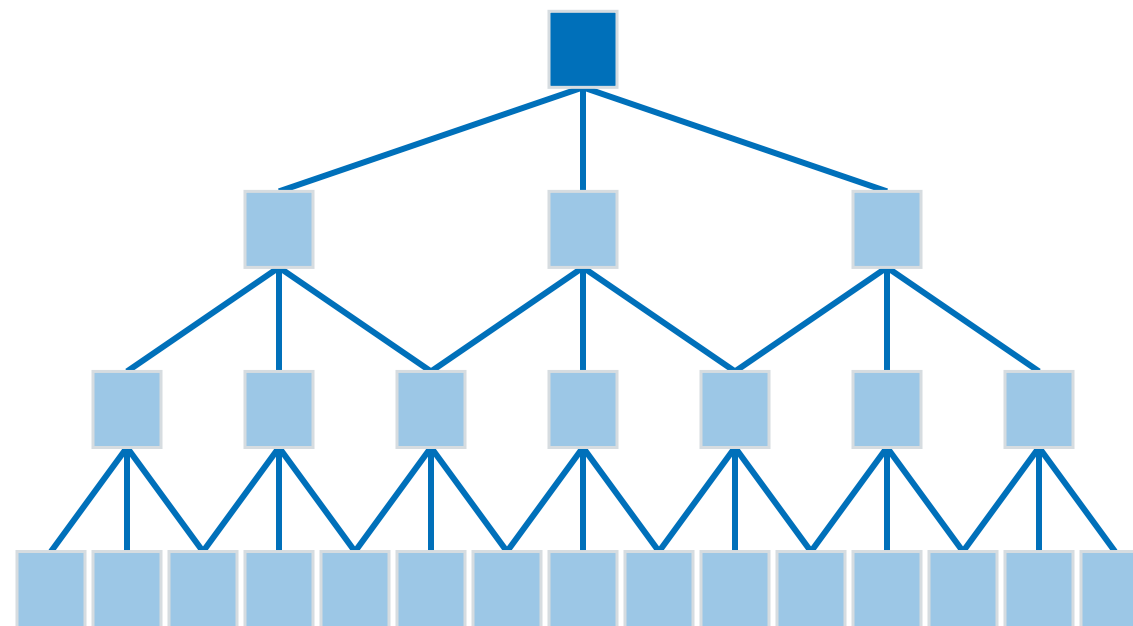
How Fast the Receptive Field Grows

Part 3 · Building a CNN

- Stride and pooling make later layers take bigger steps in the input.
- The receptive field then grows multiplicatively, not just by $K - 1$ per layer.
- The theoretical field is larger than the region that effectively matters.



three 3×3 , stride 1: RF = 7



three 3×3 , stride 2: RF = 15

$$r_l = r_{l-1} + (K_l - 1) \cdot j_{l-1}$$

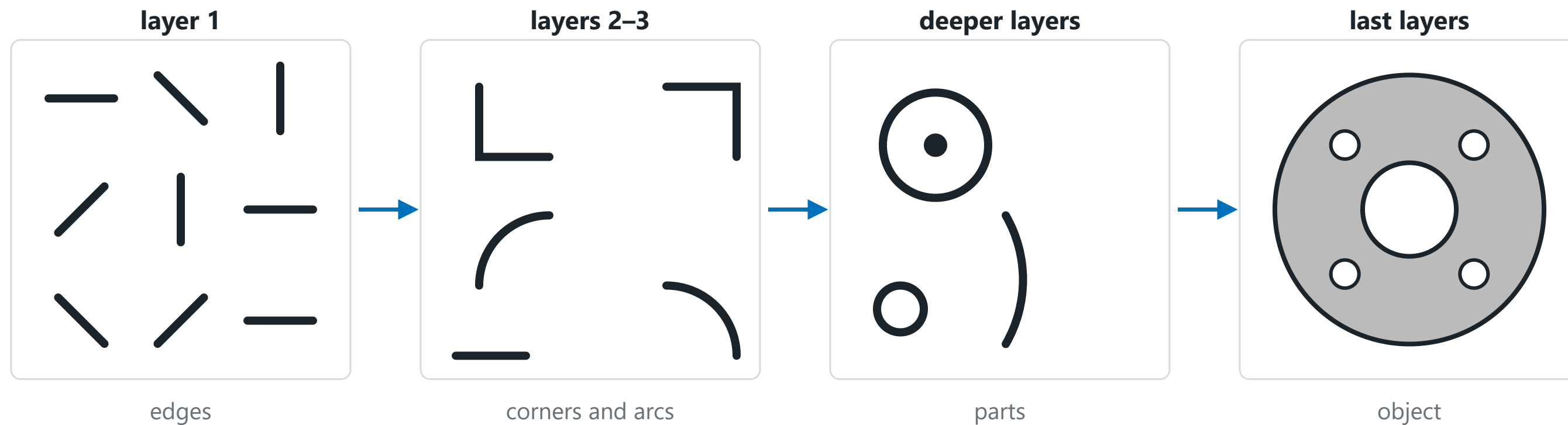
$$j_l = j_{l-1} \cdot S_l$$

j = step between neighbouring units, in input pixels

A Hierarchy of Features

Part 3 · Building a CNN

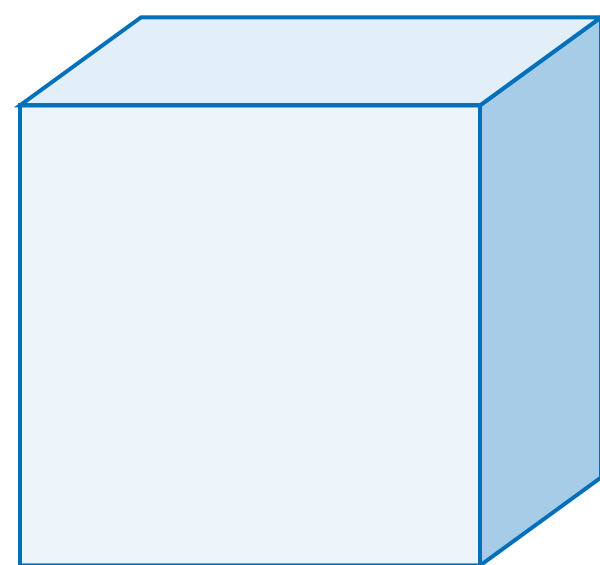
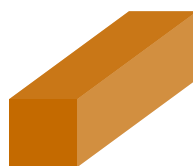
- Early layers: small receptive field → edges and colours.
- Middle layers combine them into corners, arcs, and textures, then into parts.
- Late layers carry object- and task-specific information.



1 × 1 Convolution

Part 3 · Building a CNN

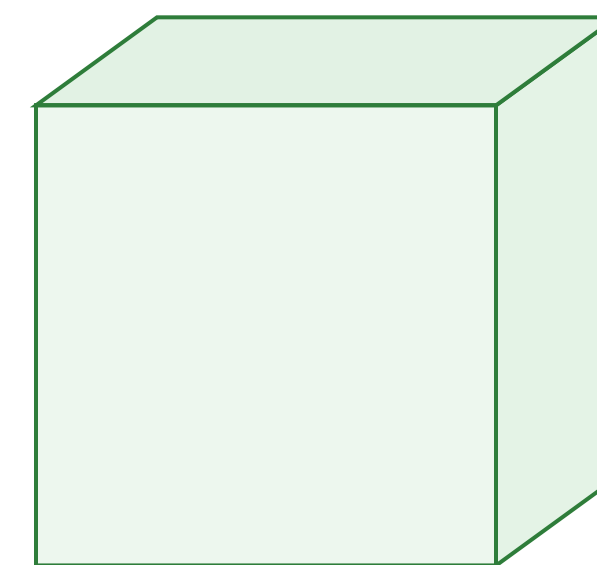
- A 1 × 1 kernel looks at a single pixel, but across all its channels.
- It is a small dense layer applied identically at every pixel.
- Used to reduce or increase the number of channels cheaply (bottlenecks).



$C_{\text{in}} \times H \times W$

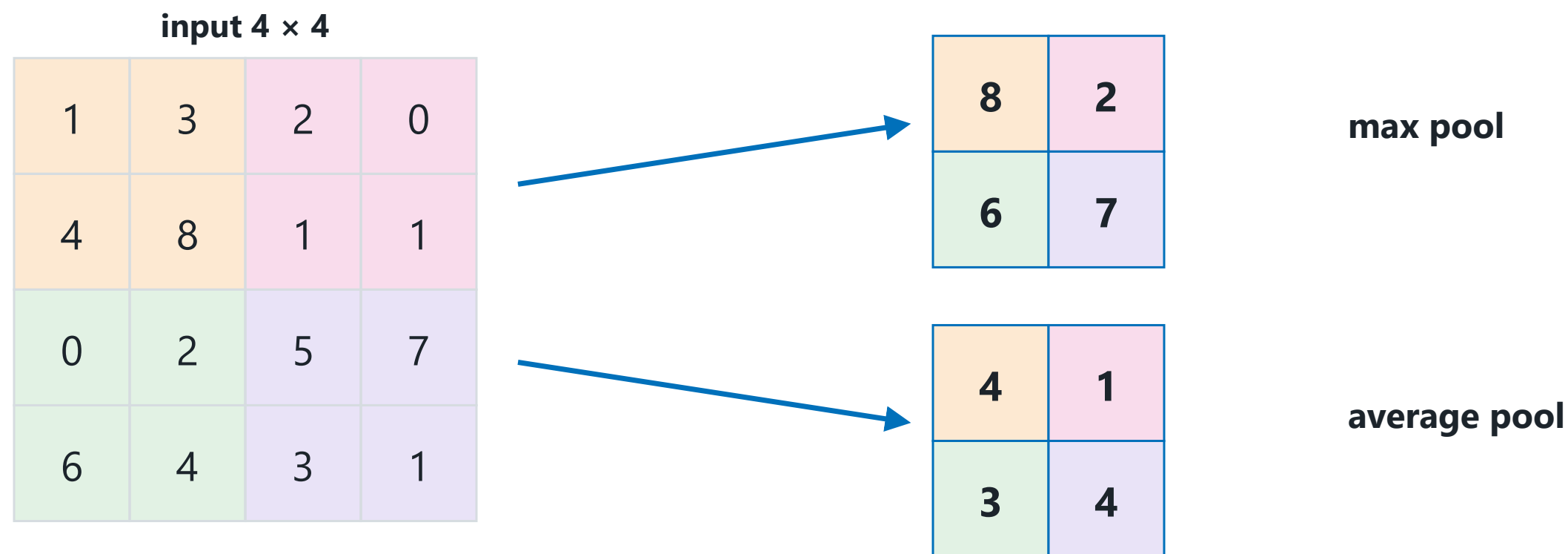
$$\begin{matrix} x_{h,w} \\ C_{\text{in}} \text{ values} \end{matrix} \times \begin{matrix} W \\ C_{\text{out}} \times C_{\text{in}} \end{matrix} = \begin{matrix} y_{h,w} \\ C_{\text{out}} \end{matrix}$$

the same small matrix at every pixel; no neighbouring pixels are used



$C_{\text{out}} \times H \times W$

- Pooling slides a window like convolution but uses a fixed rule instead of weights.
- Max pooling keeps the strongest response; average pooling keeps the mean.
- It halves the resolution and adds tolerance to small shifts.

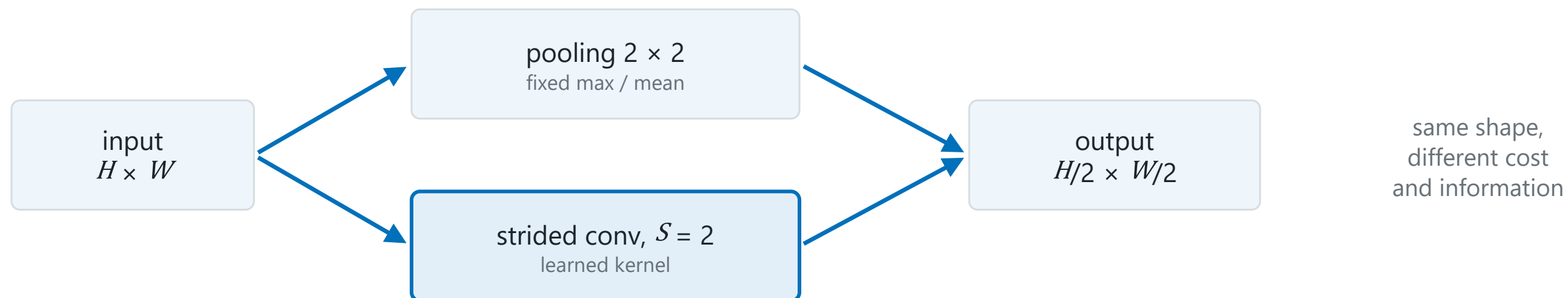


no learnable weights
each channel separately
 $H \times W \rightarrow H/2 \times W/2$

Pooling versus Strided Convolution

Part 3 · Building a CNN

- Both halve the resolution.
- Pooling has no learnable weights; a strided convolution learns what to keep.
- The choice affects cost, aliasing, and localization accuracy.

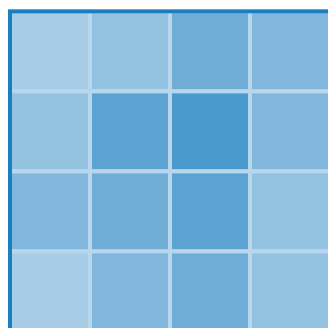


Global Average Pooling

Part 3 · Building a CNN

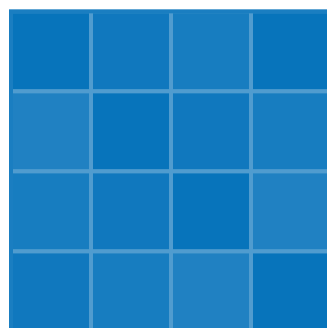
- Each channel is averaged over all positions into a single number.
- The result is a vector of length C , regardless of the input resolution.
- It replaces a large flatten layer before classification, but discards location.

channel 1



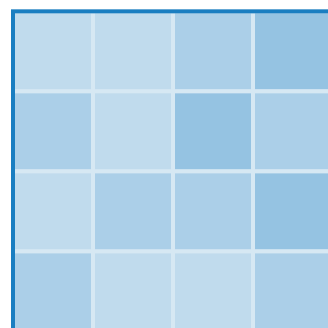
0.31

channel 2



0.77

channel 3



0.18

mean over $H \times W$,
separately for each channel

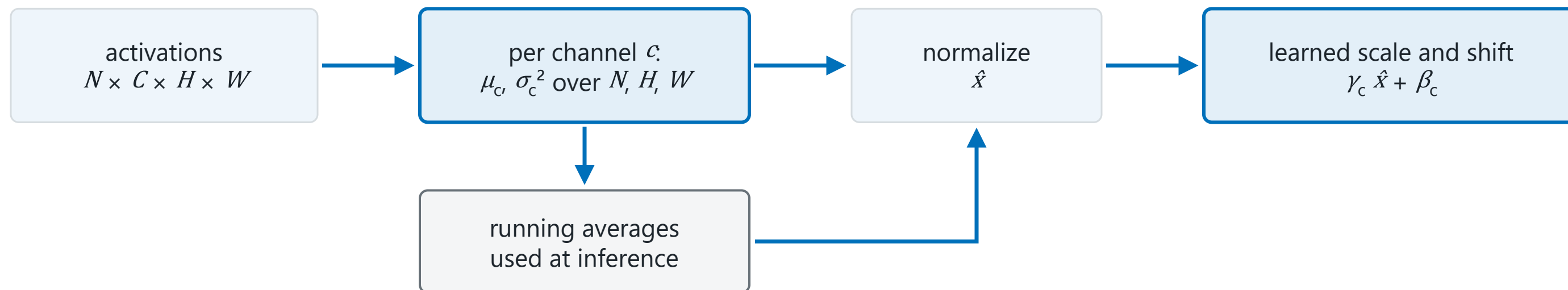
=

vector of length C
no position information

- Subtract the per-channel mean and divide by the standard deviation.
- Use the statistics of the pretraining data (e.g. ImageNet) or of the training set.
- Store them with the model; inference must use exactly the same values.



- The same idea applied inside the network, to the activations of each channel.
- During training, statistics come from the current minibatch; γ , β are learned.
- At inference, stored running averages are used instead.

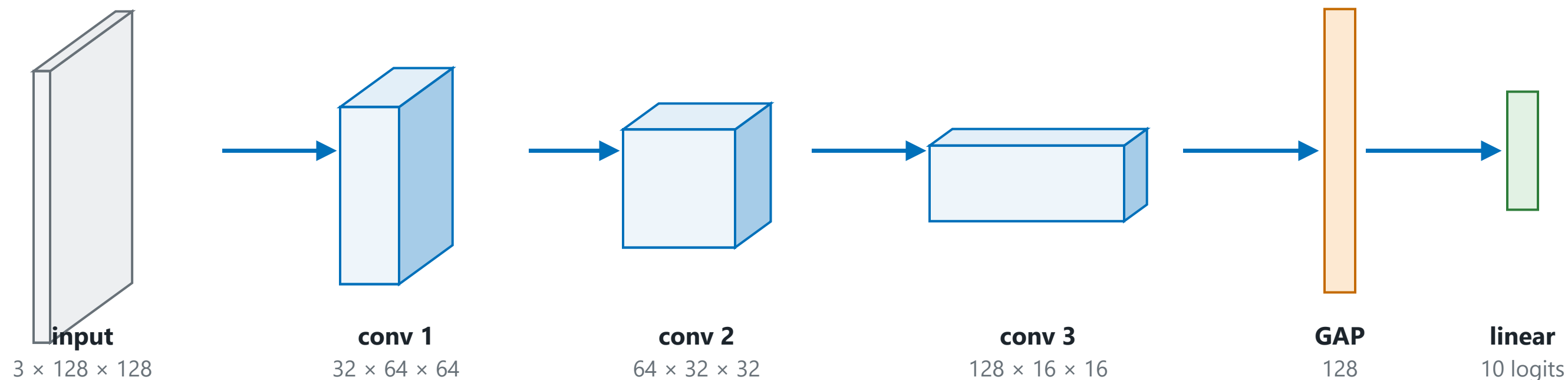


A Minimal CNN

Part 3 · Building a CNN

- Three conv blocks halve the resolution and increase the number of channels.
- Global average pooling turns the last feature map into a vector.
- A linear layer maps it to one score per class.

each conv block: 3×3 conv, stride 2, padding 1 $\rightarrow$ BatchNorm $\rightarrow$ ReLU



A Minimal CNN in Numbers

Part 3 · Building a CNN

layer	output shape	parameters	receptive field	jump
input	$3 \times 128 \times 128$	–	1	1
conv $3 \times 3, S = 2$	$32 \times 64 \times 64$	$3 \cdot 9 \cdot 32 + 32 = 896$	3	2
conv $3 \times 3, S = 2$	$64 \times 32 \times 32$	$32 \cdot 9 \cdot 64 + 64 = 18\,496$	7	4
conv $3 \times 3, S = 2$	$128 \times 16 \times 16$	$64 \cdot 9 \cdot 128 + 128 = 73\,856$	15	8
global average pool	128	–	whole image	
linear	10	$128 \cdot 10 + 10 = 1\,290$		
total		94 538		

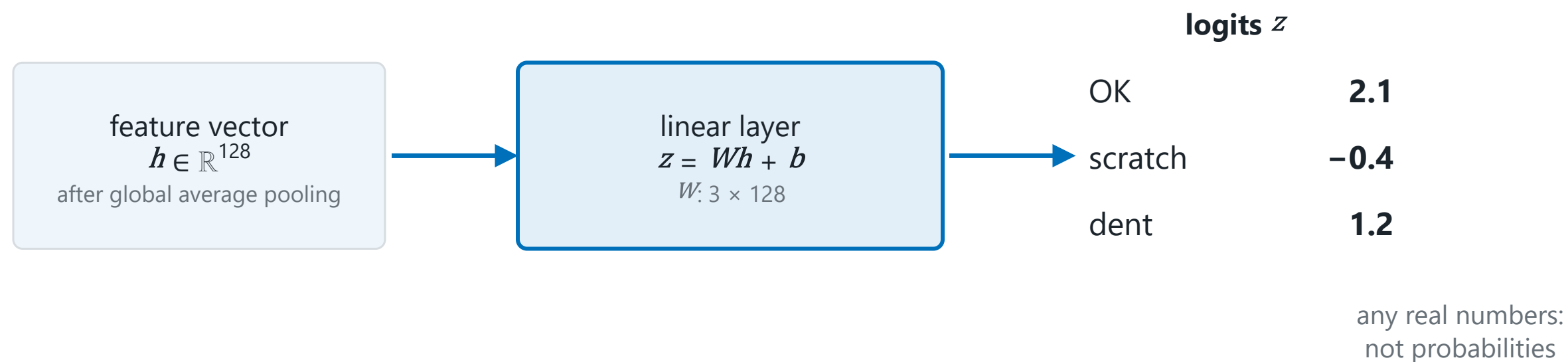
Every number follows from the formulas of Part 2. BatchNorm adds $2\mathcal{C}$ learnable values per block (omitted).

Why does this network accept a 256×256 image without any change?

Convolutions work at any resolution; global average pooling always returns a vector of length 128, so the linear layer still fits.



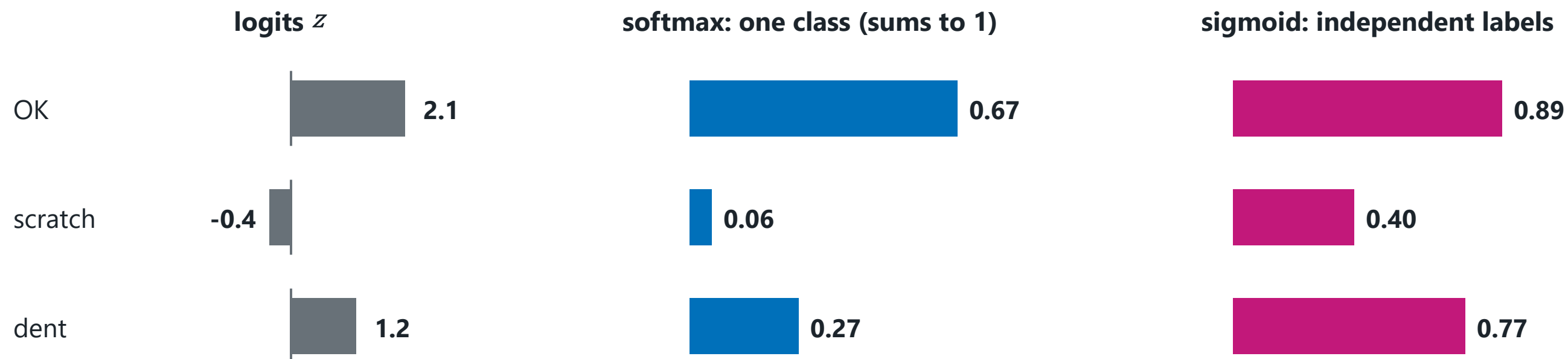
- The last linear layer produces one real number per class: the *logits*.
- Larger logit = more evidence for that class.
- Logits are not probabilities yet.



Softmax and Sigmoid

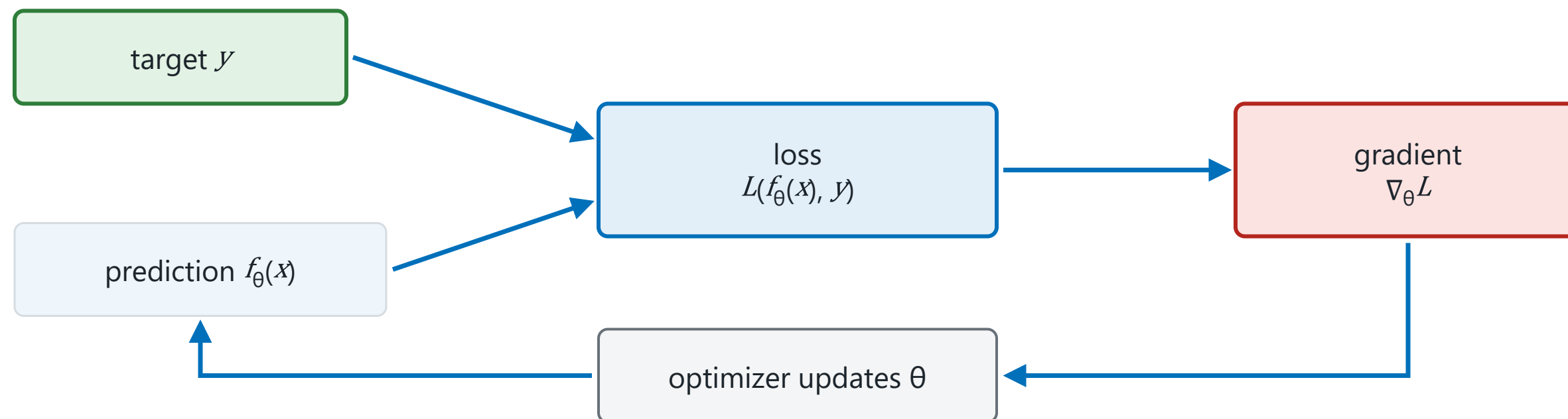
Part 4 · Training

- Softmax turns logits into a distribution over mutually exclusive classes.
- Sigmoid squashes each logit separately; use it for independent labels.
- Move the logits: softmax always sums to one, sigmoid outputs do not.



softmax sums to 1.00; sigmoid outputs sum to 2.06

- A loss compares the prediction with the target and returns one number.
- Its gradient tells the optimizer how to change the weights.
- It must match the output type; a reporting metric need not be a good loss.



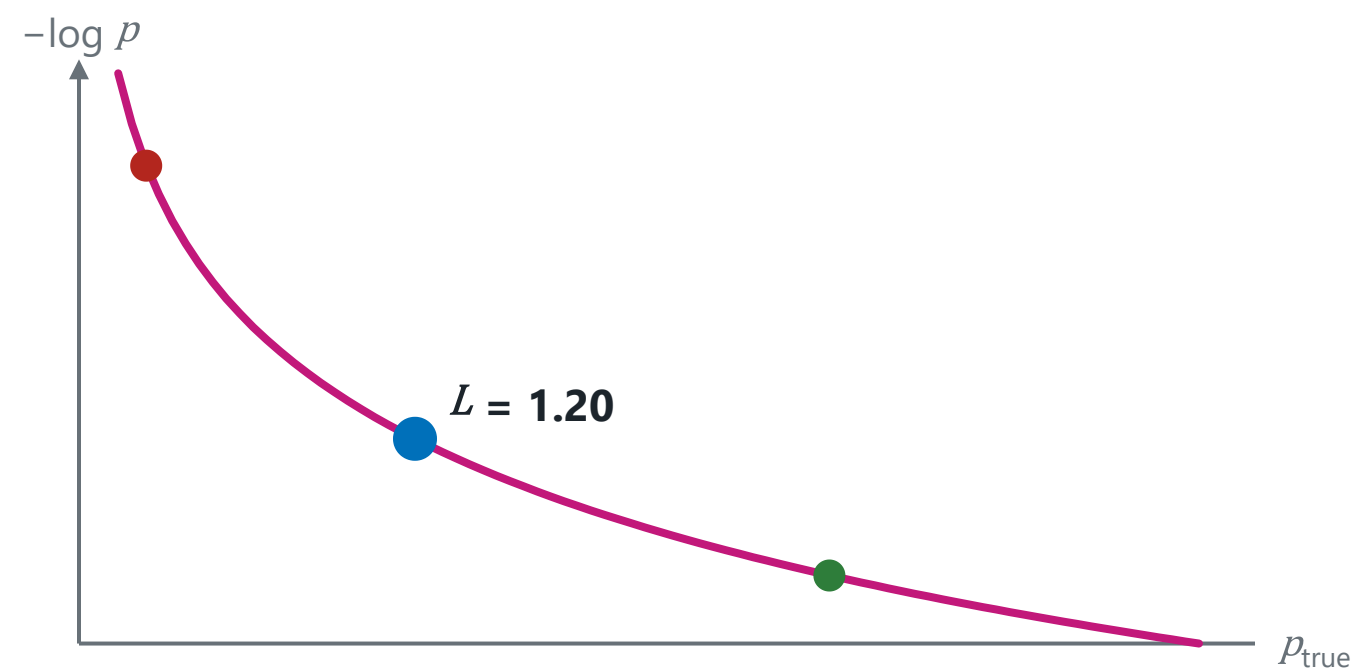
- Classification loss: minus the log-probability of the correct class.
- Confident correct predictions cost little; confident mistakes cost a lot.
- Most libraries take logits directly; do not apply softmax twice.

$$L_{\text{CE}} = -\sum_c y_c \log p_c = -\log p_{\text{true}}$$

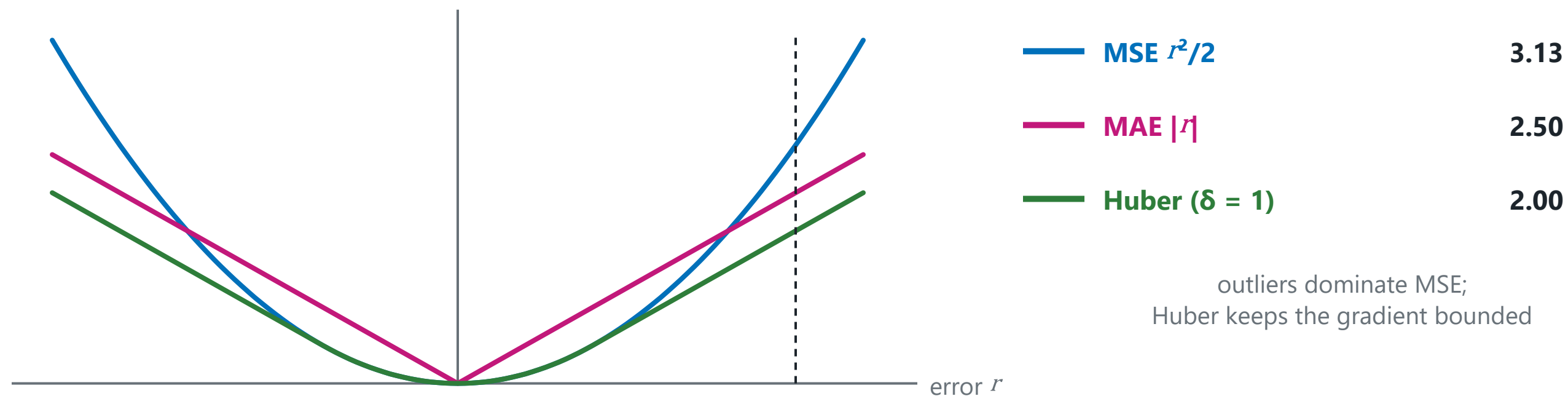
one-hot target: only the true class contributes

true class OK, $p = 0.67 \rightarrow L = 0.40$

true class scratch, $p = 0.06 \rightarrow L = 2.81$

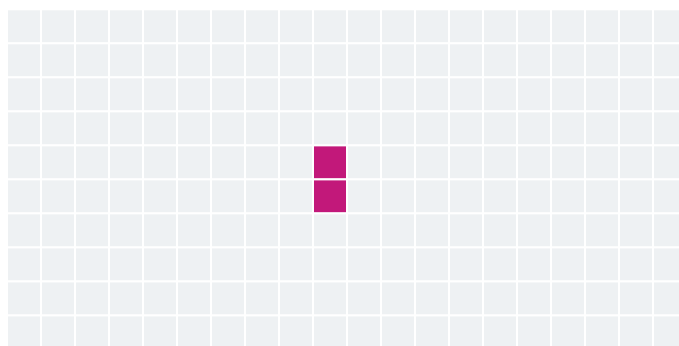


- MSE penalizes large errors quadratically, so outliers dominate.
- MAE grows linearly: robust, but its gradient magnitude never shrinks.
- Huber is quadratic near zero and linear for large errors.



- Pixel-wise cross-entropy is classification at every pixel.
- When the object covers few pixels, the background dominates the loss.
- Dice loss measures region overlap and is insensitive to this imbalance; either loss works on its own.

defect = 1 % of pixels



pixel-wise CE: 99 % of the loss
comes from background

pixel-wise cross-entropy
 $-(1/HW) \sum \log p_{\text{true}}$
every pixel counts equally
→ dominated by background

or

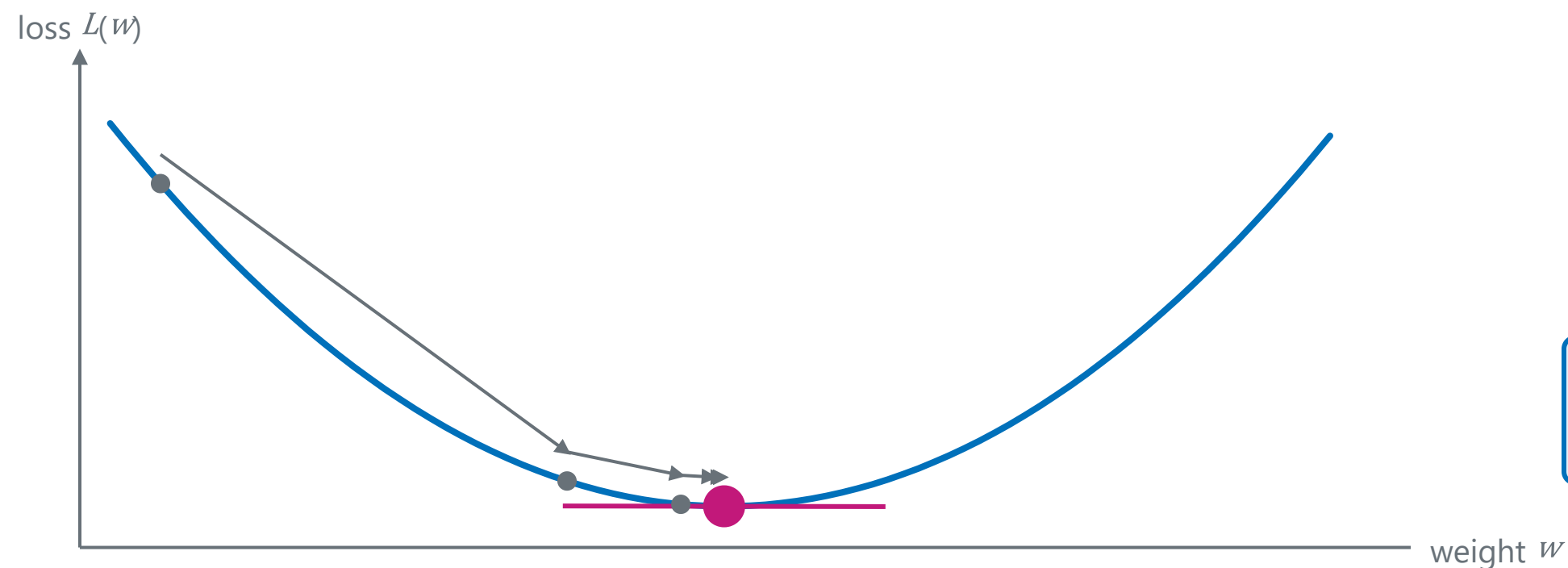
Dice loss
 $1 - 2|P \cap Q| / (|P| + |Q|)$
overlap of the whole region
→ insensitive to object size

two independent choices: each loss can be used on its own

Gradient Descent in One Dimension

Part 4 · Training

- Imagine the loss as a function of a single weight w .
- The derivative is the slope; stepping against it lowers the loss.
- The learning rate η sets the size of the step. Press $\rightarrow$ to take a step.



step 5: $w = 3.20$

slope = -0.00

$$w \leftarrow w - \eta \partial L / \partial w$$

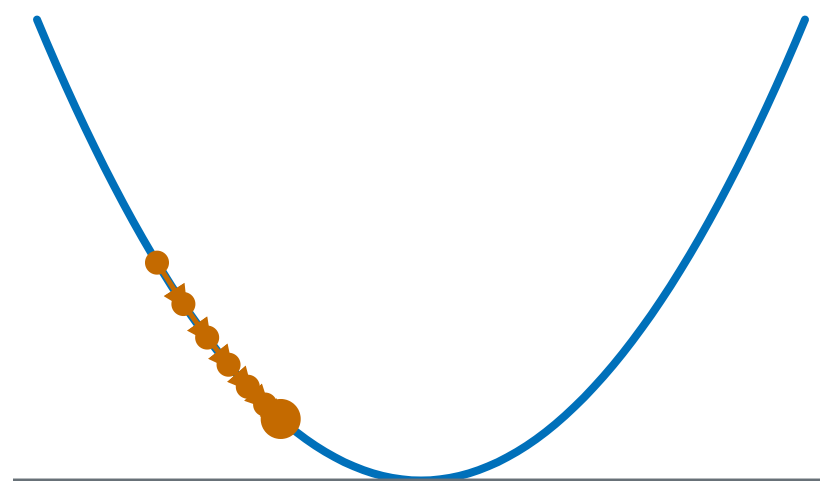
negative slope $\Rightarrow$ increase w
positive slope $\Rightarrow$ decrease w

Choosing the Learning Rate

Part 4 · Training

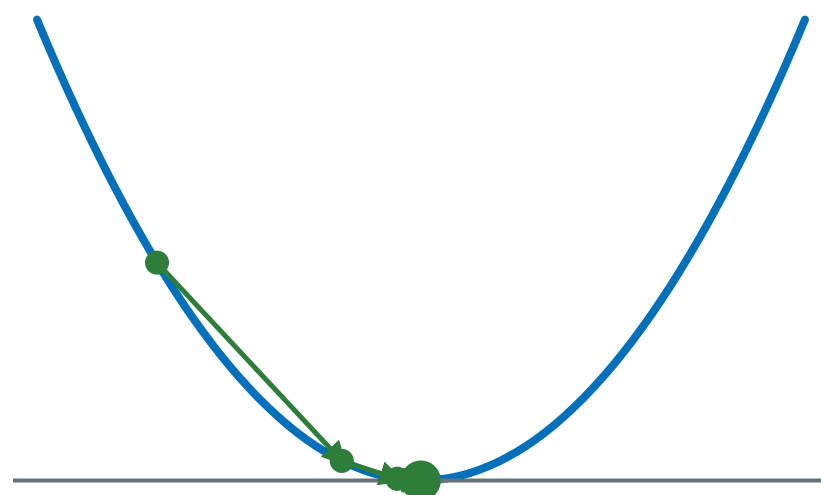
- Too small: training is slow and may stall.
- Too large: steps overshoot the minimum, the loss oscillates or becomes NaN.
- In practice: try values on a log scale (10^{-4} , 10^{-3} , 10^{-2}) and watch the curve.

η too small



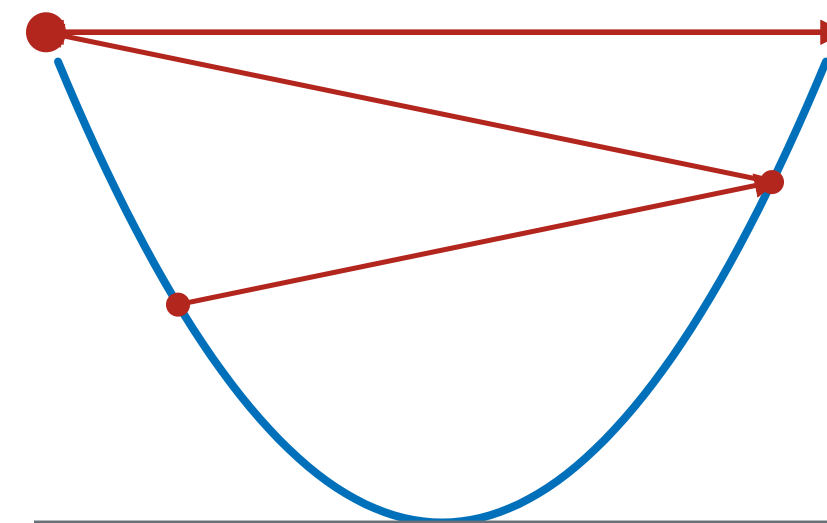
slow progress

η suitable



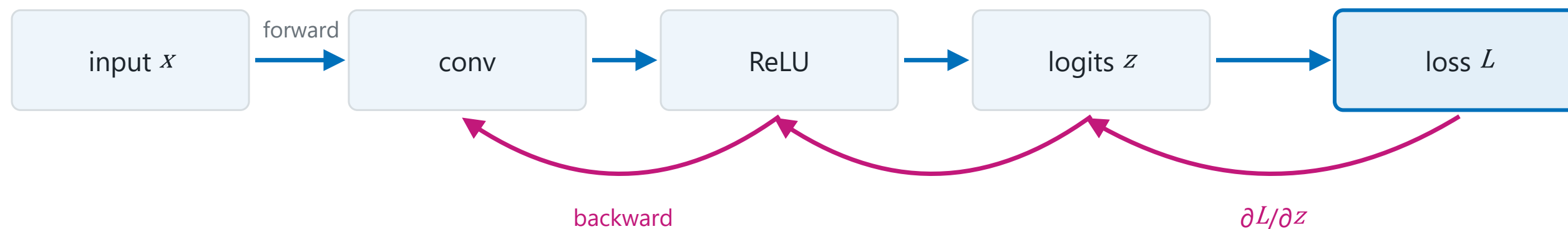
fast and stable

η too large



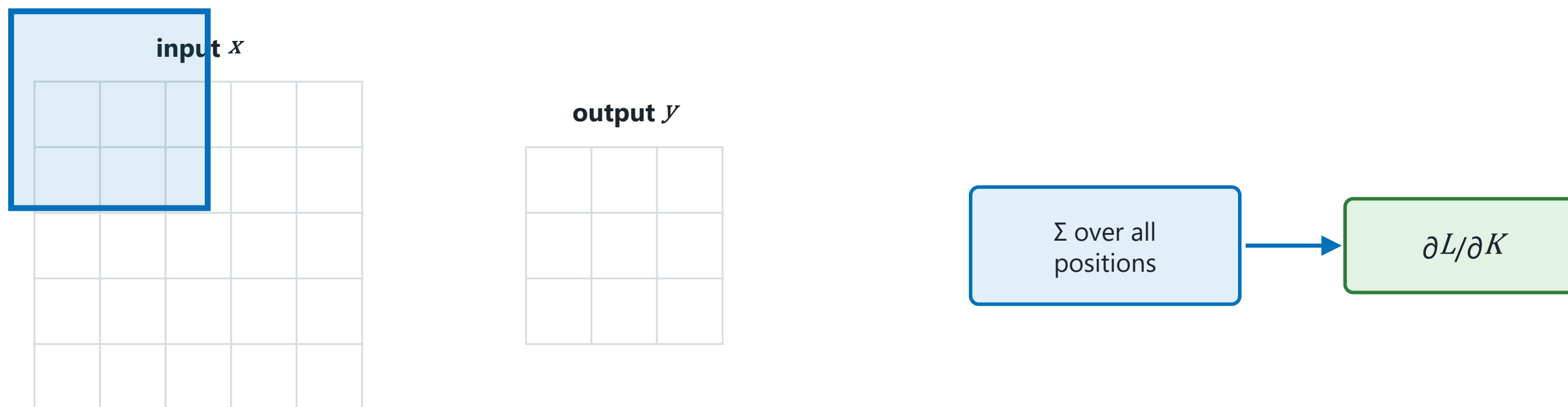
oscillates, diverges $\rightarrow$ NaN

- The forward pass computes the prediction and stores intermediate values.
- The backward pass applies the chain rule from the loss back to every weight.
- One backward pass gives the gradient for all millions of weights at once.



chain rule: $\partial L / \partial w = \partial L / \partial z \cdot \partial z / \partial a \cdot \partial a / \partial w$ — each layer multiplies by its local derivative

- The same kernel produced every output value.
- Its gradient is therefore a sum of contributions from all positions.
- A pattern that matters anywhere in the image updates the same weights.

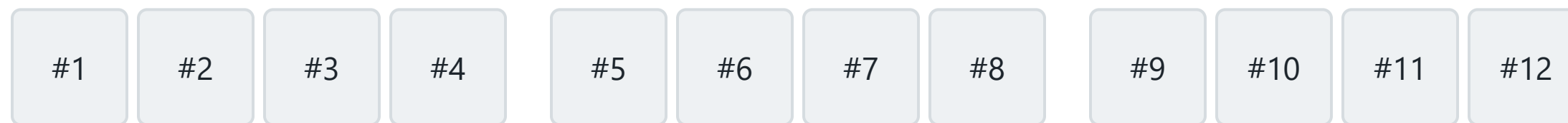


$$\partial L / \partial k_{u,v} = \sum_{i,j} \partial L / \partial y_{i,j} \cdot x_{i+u, j+v} \text{ — every output position contributes to the same kernel}$$

Batch, Iteration, and Epoch

Part 4 · Training

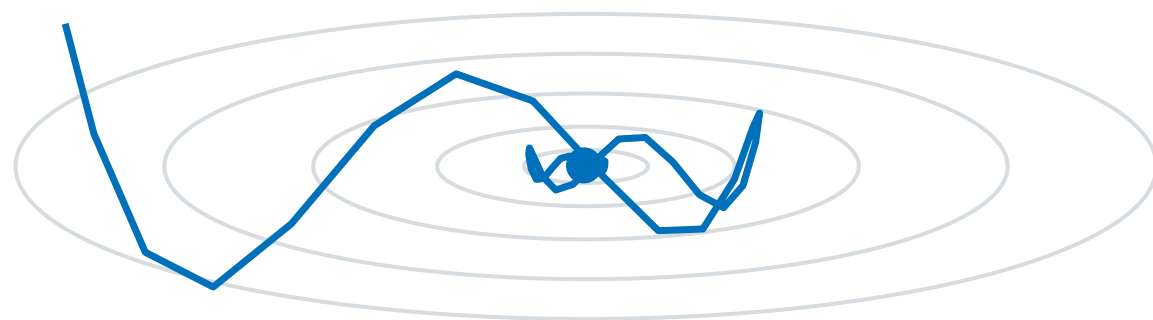
- The gradient is estimated from a minibatch of B images, not from the whole set.
- One iteration = one minibatch = one weight update; one epoch = one pass over the data.
- The number of updates matters more than the epoch count alone.



one epoch = all 12 images; minibatch $B = 4 \Rightarrow 12 / 4 = 3$ iterations per epoch

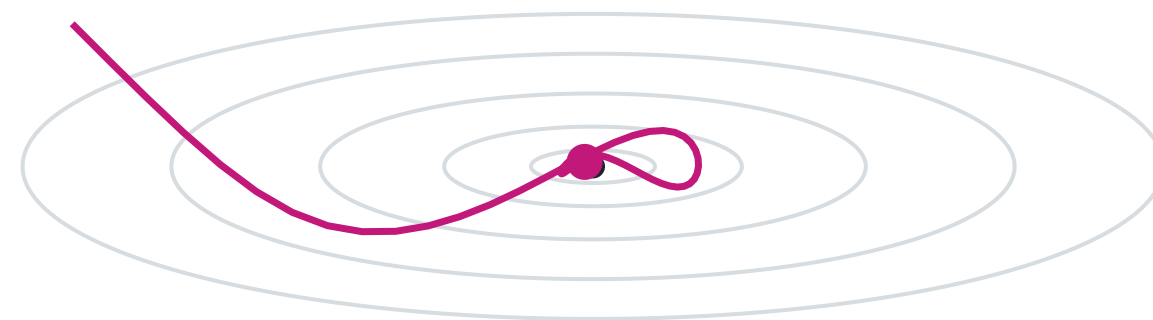
each epoch uses a new random order

- Minibatch gradients are noisy; momentum averages them over recent steps.
- AdamW additionally adapts the step size to each parameter.
- Both paths below are computed live on the same elongated loss surface.



SGD + momentum

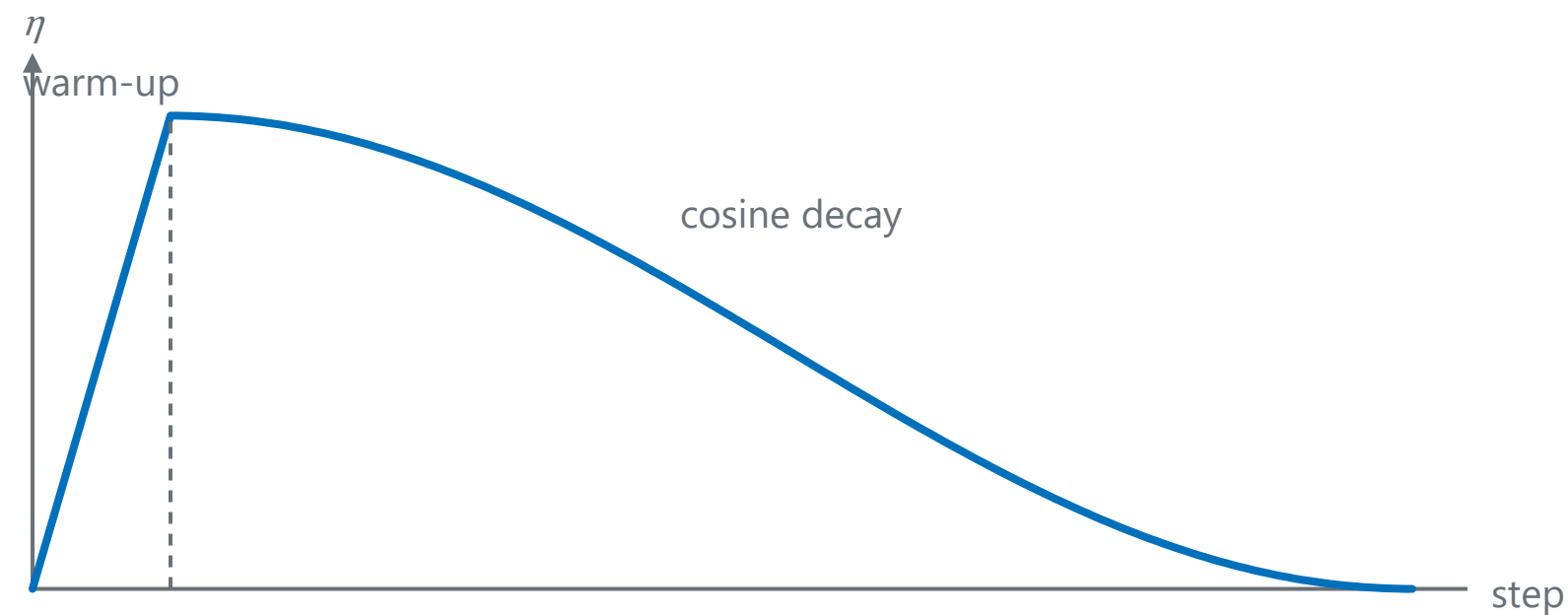
averages past gradients, damps the zig-zag



AdamW

per-parameter step size + decoupled weight decay

- Warm-up protects early training from steps that are too large.
- Cosine or step decay refines the final stage with smaller steps.
- Recheck the learning rate whenever the batch size changes.



- Train is used to learn parameters.
- Validation guides model and hyperparameter selection.
- Test is used only once, for the final independent estimate.



- Images of the same product must not appear in more than one split.
- Fit preprocessing statistics only on the training set.
- Temporal and spatial correlation can inflate the measured performance.

wrong: photos of one item in both splits

TRAIN: A1, B1, C1
TEST: A2, B2, C2

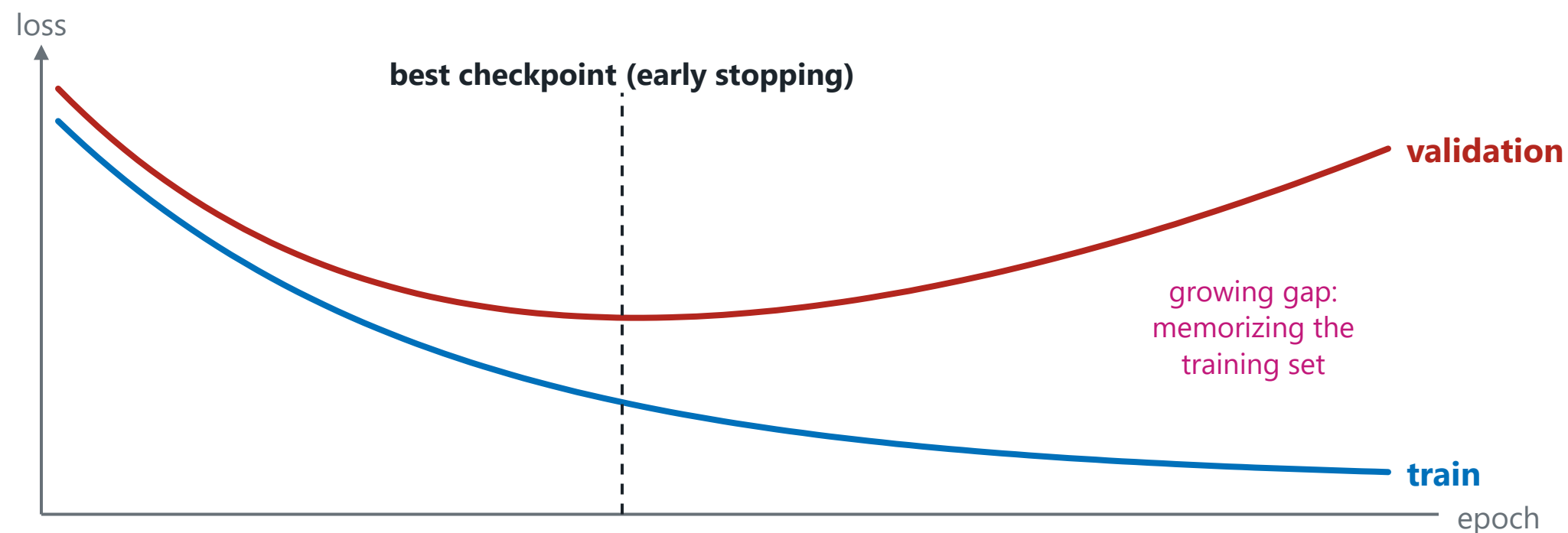
group by item

correct: split by item

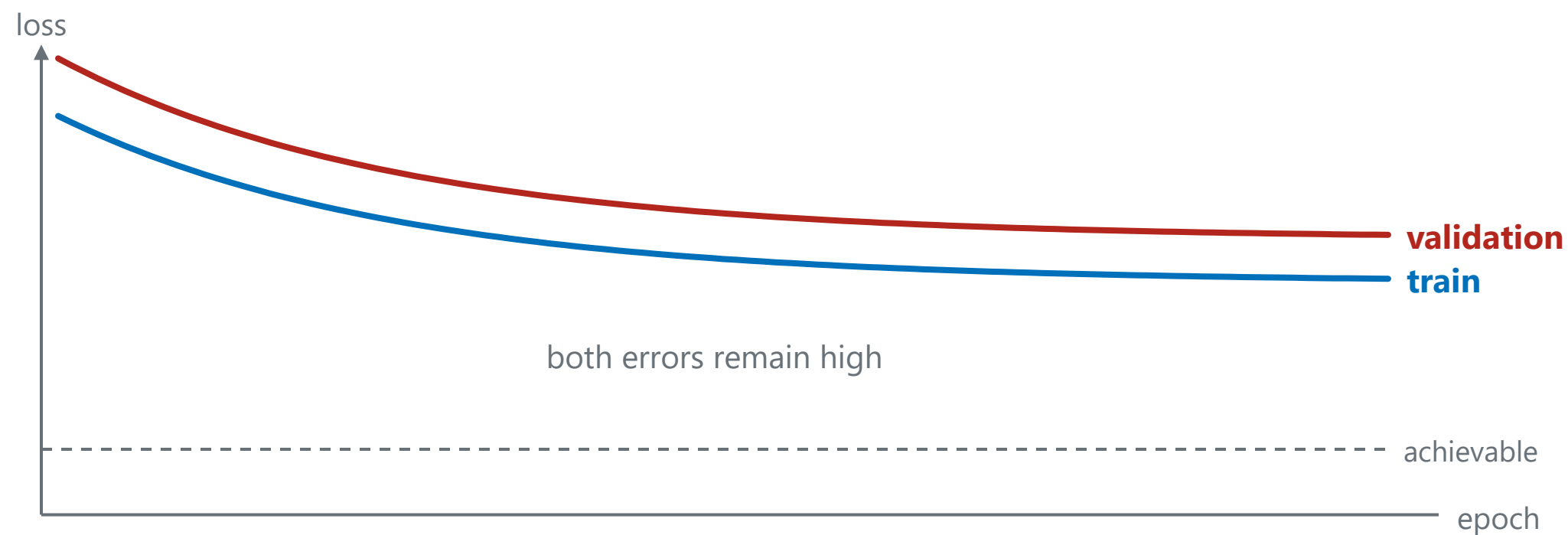
TRAIN: A1, A2, B1, B2
TEST: C1, C2

A1, A2 = two photos of item A

- Training loss keeps falling while validation loss starts to rise.
- The model memorizes random or source-specific details of the training set.
- Keep the checkpoint with the best validation result (early stopping).

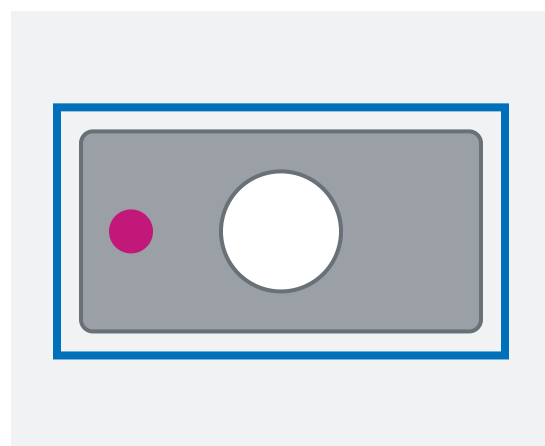


- Both training and validation results are poor.
- Causes: too small a model, too short training, a bad learning rate, or wrong inputs.
- Stronger regularization does not solve underfitting.



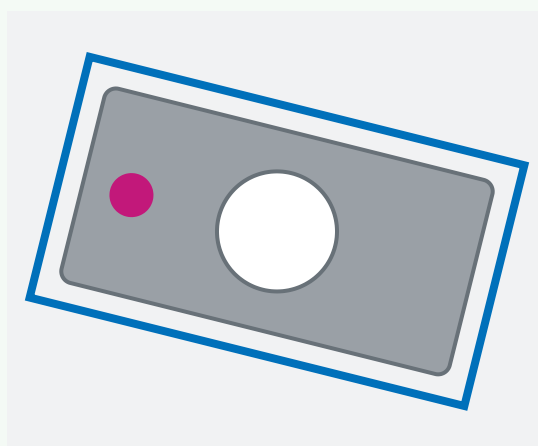
- Random transformations create new training samples; the label must stay correct.
- Apply geometry consistently to the image, mask, and boxes.
- Photometric changes simulate the camera and illumination.

ORIGINAL



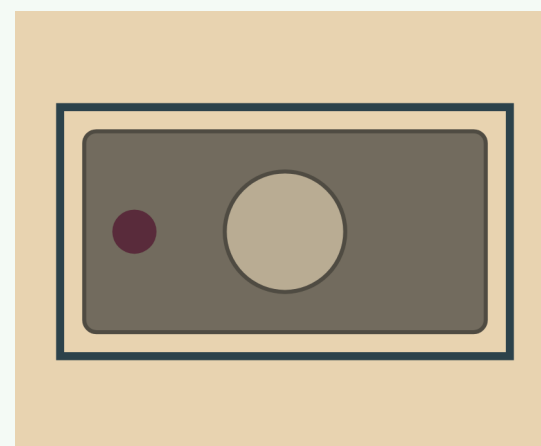
image, box and mask
describe one object

✓ GEOMETRIC



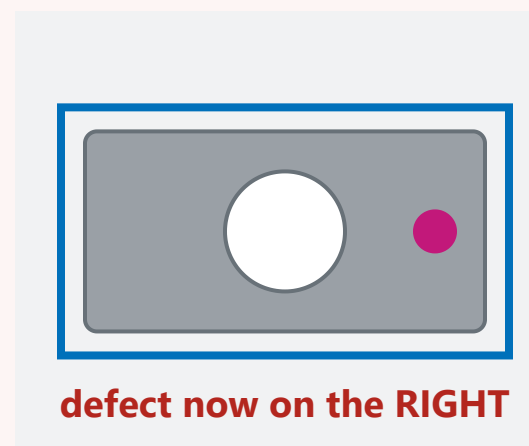
rotate image, box and mask
by the same transform

✓ PHOTOMETRIC



change appearance;
geometry stays fixed

✗ LABEL CHANGES



defect now on the RIGHT

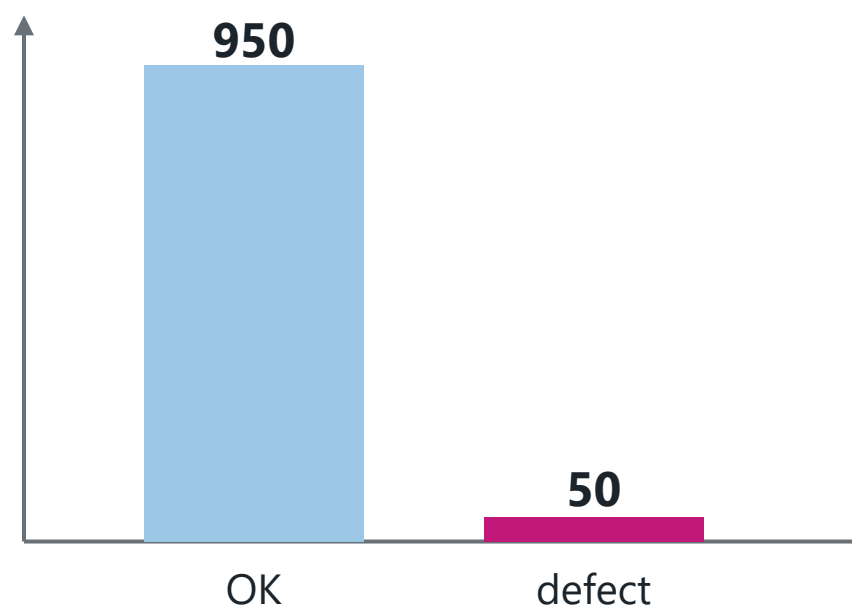
a horizontal flip changes
a left/right label

Is a horizontal flip suitable when reading digits or detecting a defect on the left?

No: augmentation encodes invariances and must not change the meaning of the target.



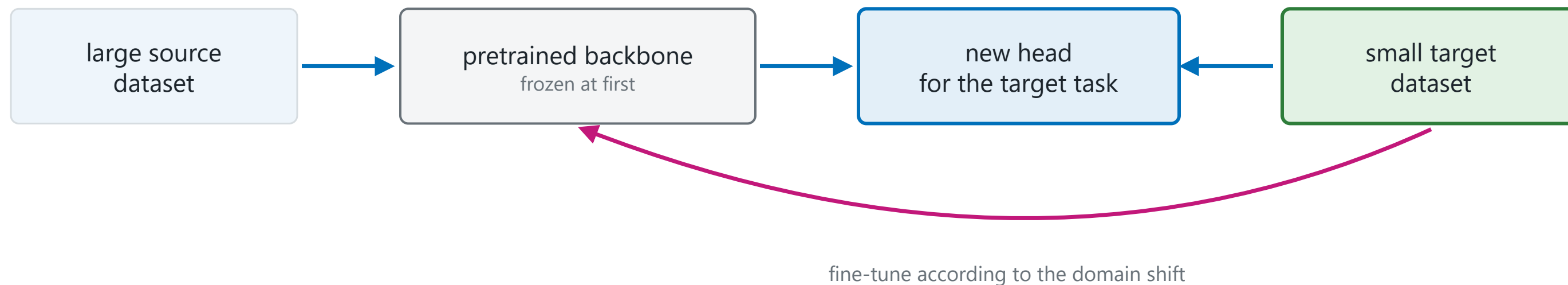
- Accuracy can hide complete failure on a rare class.
- Use per-class recall, precision, and a confusion matrix.
- Sampling, a weighted loss, or focal loss may help.



always predict "OK" $\Rightarrow$ accuracy 95 %
yet zero recall for defects

report per-class precision, recall,
and the confusion matrix

- A pretrained backbone already provides general visual features.
- Train the new head first, then carefully unfreeze the backbone.
- The domain shift determines how many layers should be adapted.



Weight Decay and Dropout

Part 5 · Data and diagnosis

- Weight decay limits the magnitude of weights.
- Dropout randomly disables activations during training.
- For CNNs, sound augmentation often matters more than a high dropout rate.

weight decay
 $w \leftarrow (1 - \eta\lambda)w$

limits weight magnitude

dropout
random units switched off in training



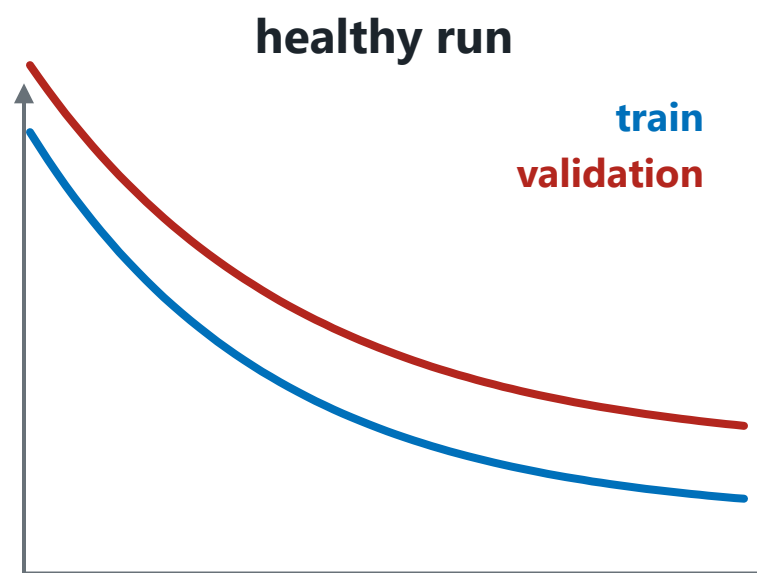
augmentation
safe data invariances

often the most effective for CNNs

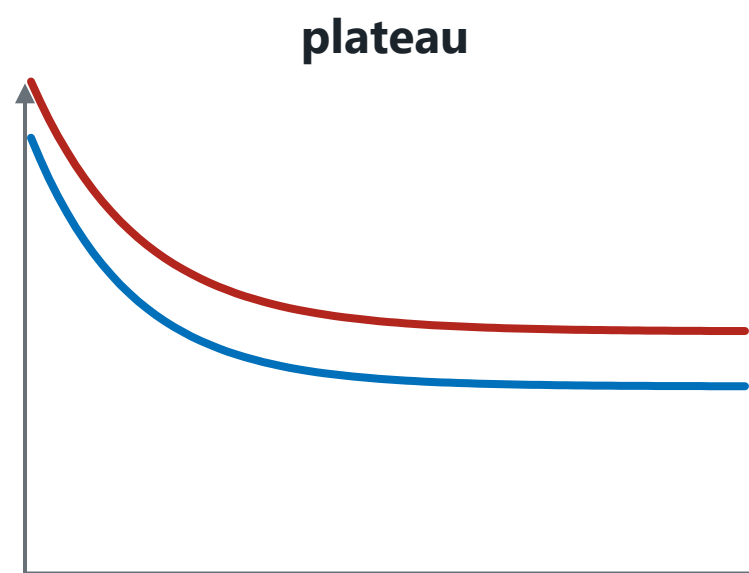
Diagnosing Training Curves

Part 5 · Data and diagnosis

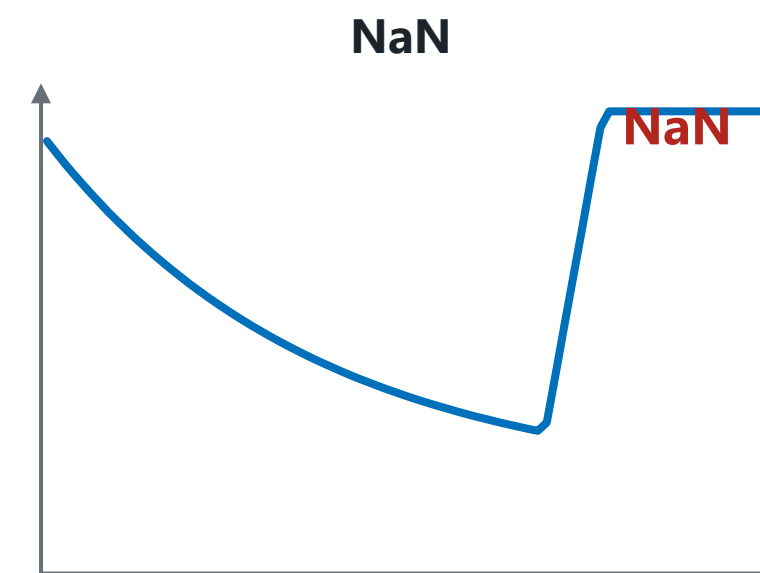
- Track training and validation loss as well as task metrics.
- A plateau may expose a limit in data, representation, or optimization.
- A sudden NaN indicates a numerical error or an excessive step size.



keep going

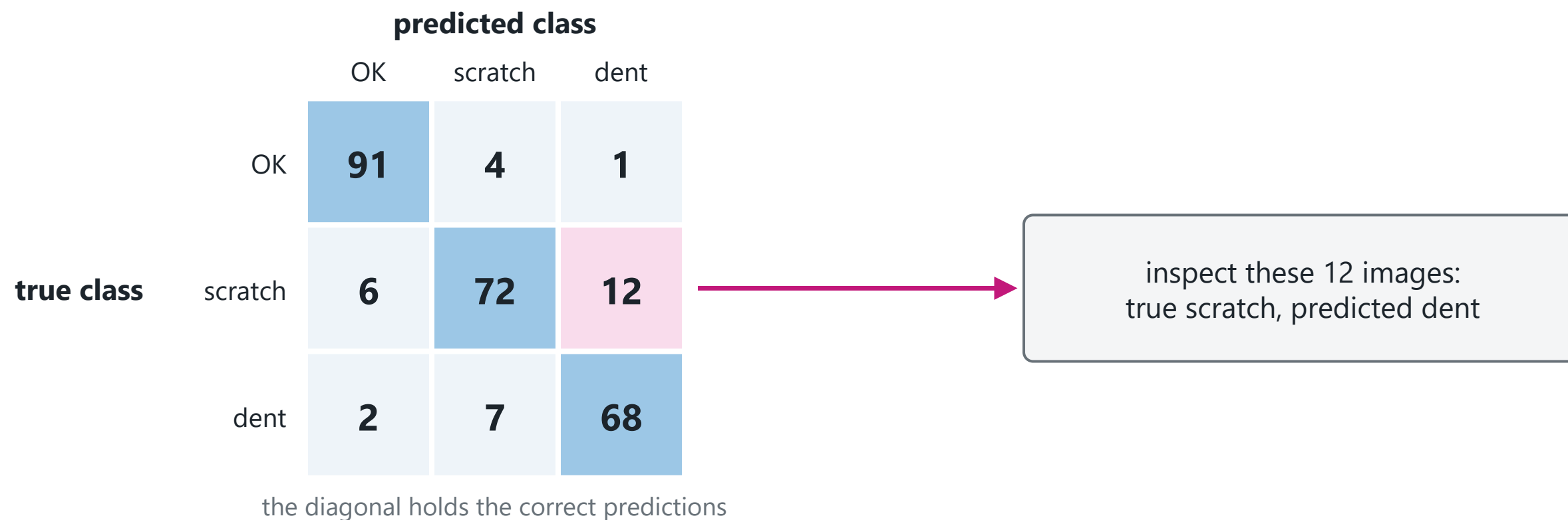


check data, capacity, η

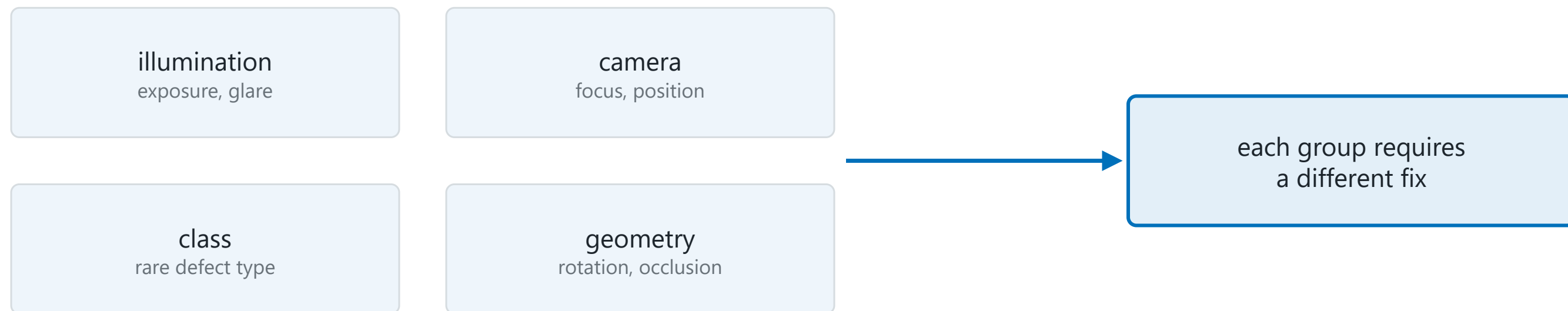


lower η , check inputs

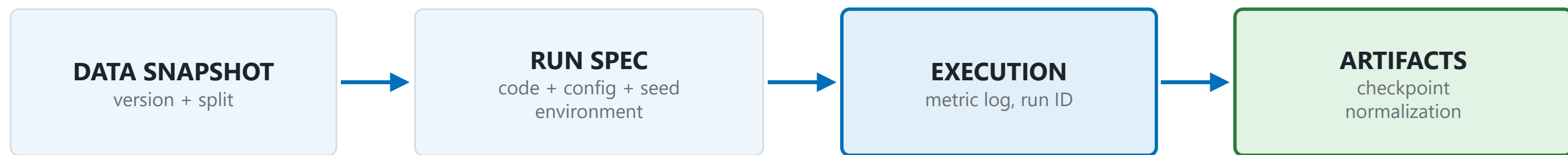
- Rows are true classes, columns are predictions; off-diagonal cells are errors.
- A row-normalized matrix helps when classes are imbalanced.
- Inspect the actual images behind the largest confusions.



- Group errors by illumination, camera, class, and geometry.
- Each group leads to a different pipeline correction.
- One aggregate number cannot explain operational risk.

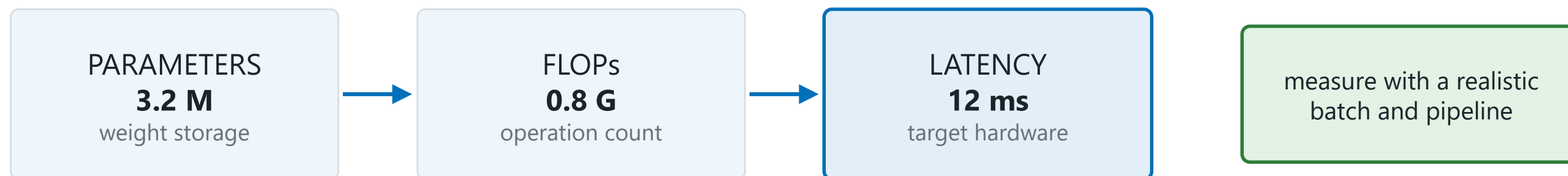


- Store the seed, split, dataset version, and configuration.
- A checkpoint must contain both weights and normalization.
- Report variation across runs when the dataset is small.



same snapshot + same specification $\Rightarrow$ comparable rerun

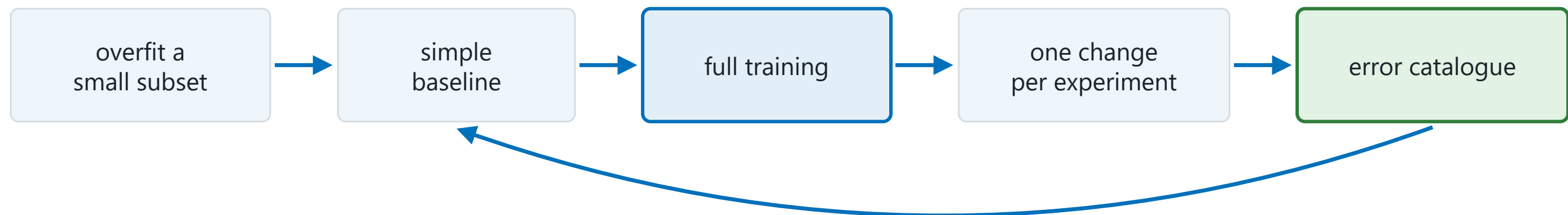
- Parameters determine model size, not latency.
- FLOPs ignore memory transfers and hardware behaviour.
- Measure latency on the target device with a realistic batch size.



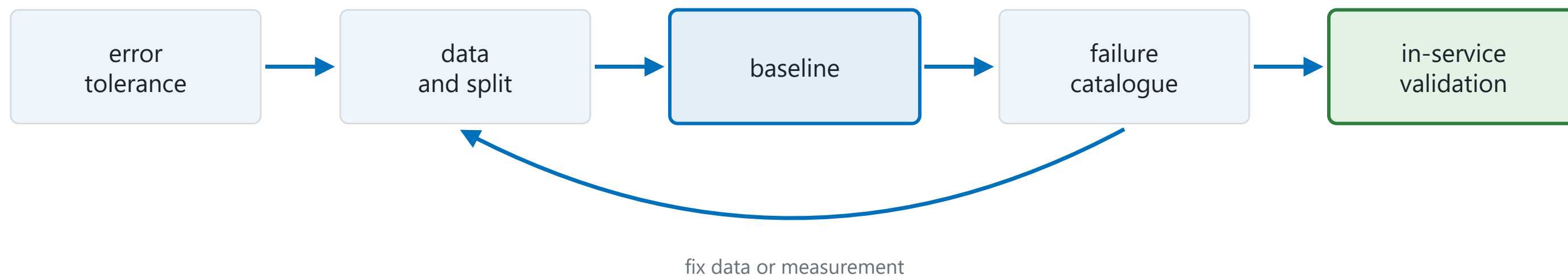
A Sensible First Experiment

Part 5 · Data and diagnosis

- Start with a simple pretrained backbone and a short training run.
- Sanity checks: one batch must be fit almost perfectly; random labels must not generalize.
- Look at augmented inputs before training; then change one thing at a time.



- Define the error tolerance and the unit used for data splitting.
- Build a baseline and a catalogue of failures.
- Iterate on data and measurement first, then on architecture.

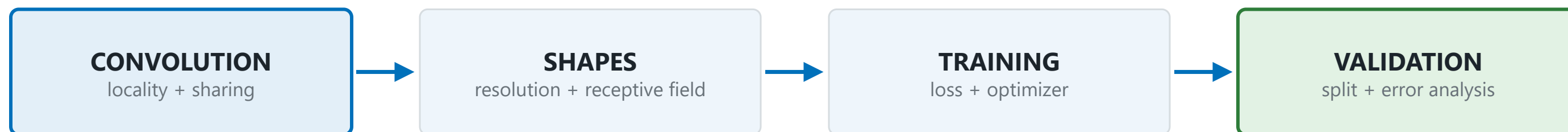


Design a CNN-based defect inspection: input, output, loss, and two metrics.

Justify the split unit and identify which invariances are safe.



- Convolution slides one small learned detector over the whole image.
- Output shape, parameters, and receptive field can be computed before training.
- A good experiment is measurable, reproducible, and analyzes its errors.



- You can calculate the shape and parameters of a convolutional layer.
- You can distinguish a logit, activation, loss, and metric.
- Next, we will combine a backbone, neck, and head for specific tasks.

