

# CNN Architectures

Backbones, Necks, and Heads for Machine Vision

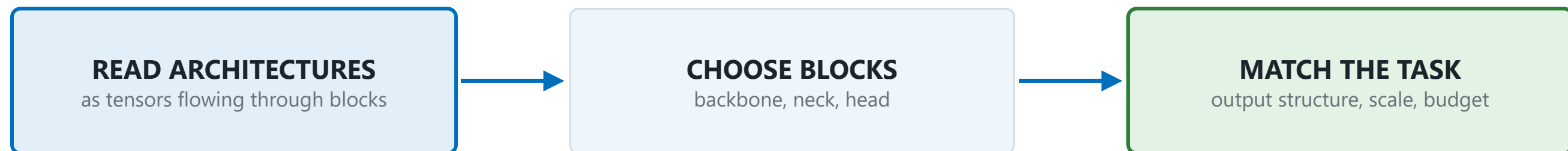
Matouš Cejnek

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

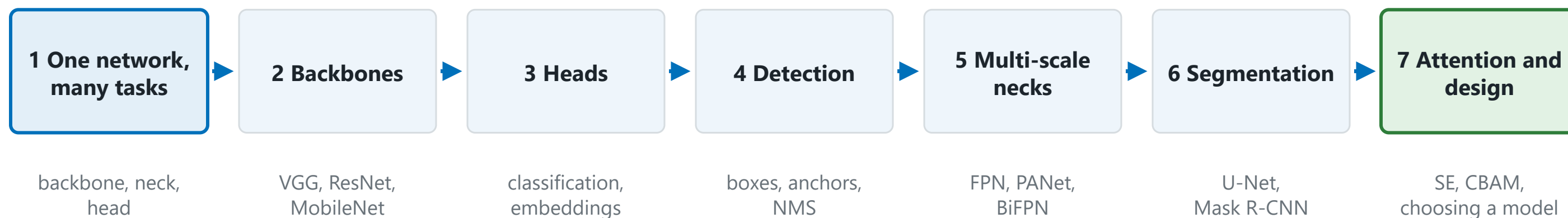
# Why This Lecture

Part 1 · One network, many tasks

- Lecture 1 built one small CNN for one task. Real systems reuse a few proven designs.
- The goal is to read an architecture as tensors flowing through named blocks.
- By the end, you can choose a backbone, neck, and head for a given vision problem.



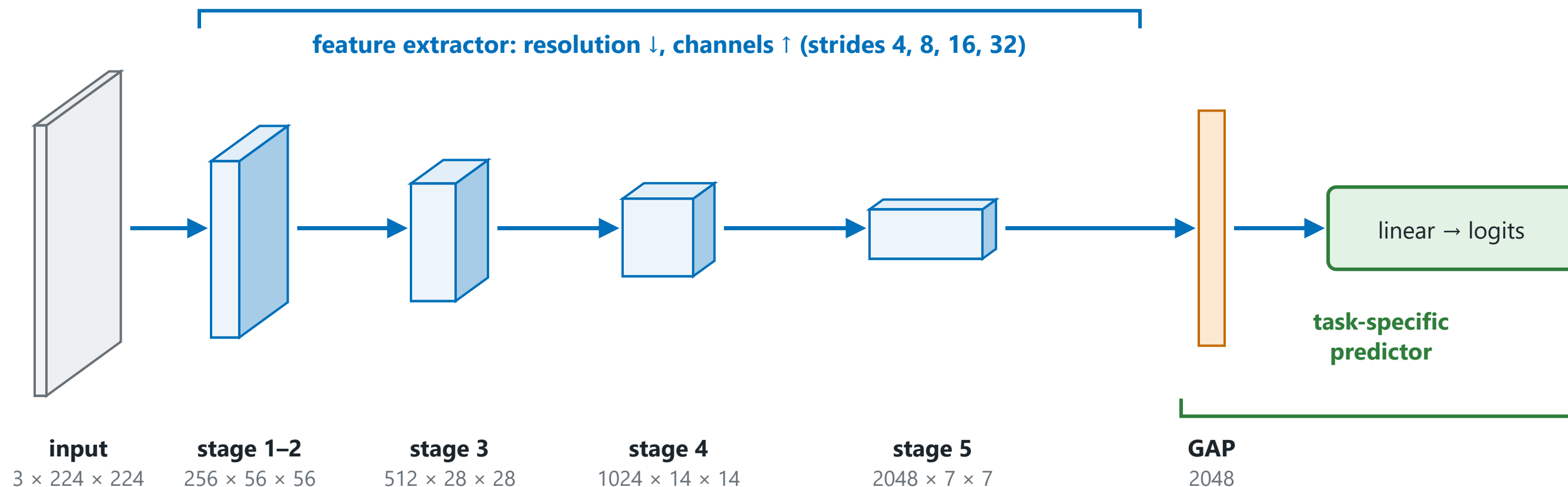
- Seven parts: the system view first, then each building block, then design decisions.
- Every block is introduced through the problem it solves.
- Check questions close each part.



# Recap: What a CNN Already Does

Part 1 · One network, many tasks

- Stacked conv blocks lower the resolution and raise the number of channels.
- Strides 4, 8, 16, 32 give feature maps at several scales — but we used only the last one.
- A small head (GAP + linear) turned the last map into class scores.



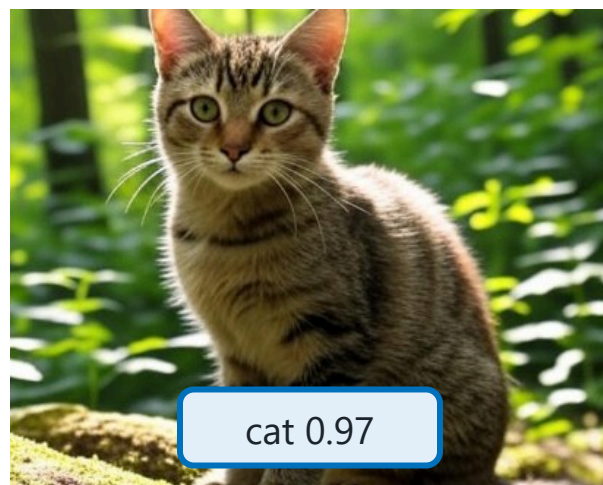
# Four Tasks, Four Output Shapes

Part 1 · One network, many tasks

- The same image can be asked four different questions.
- Each answer has a different tensor shape — detection even a variable length.
- The architecture must produce exactly that shape.

## CLASSIFICATION

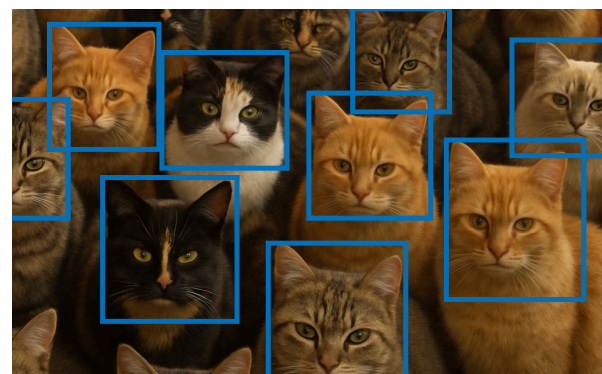
Is this a cat?



$C$  class scores

## DETECTION

Where are the cats?



$N \times (4 \text{ box} + 1 \text{ score} + C)$   
 $N$  varies per image

## SEGMENTATION

Which pixels are apple?



$C \times H \times W$  logits

## REGRESSION

How many cats?



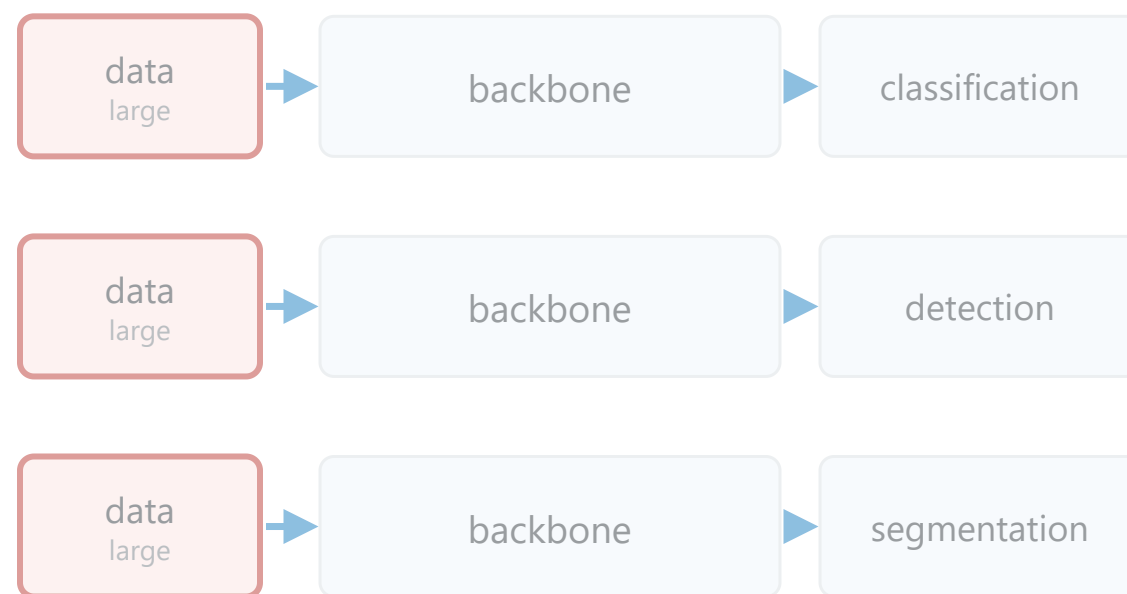
one number

# Do Not Train Everything From Scratch

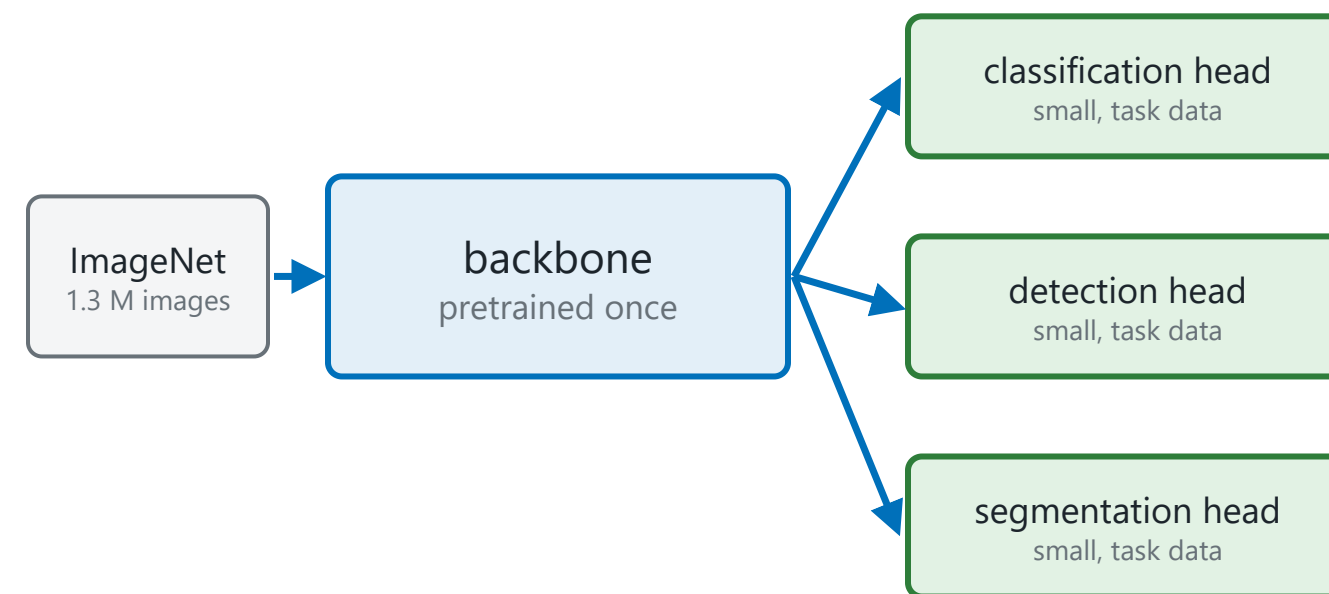
Part 1 · One network, many tasks

- A deep feature extractor needs a lot of data and compute to train well.
- Good features (edges, textures, parts) are useful for almost every visual task.
- So we train it once on a large dataset and reuse it; only the task-specific part is new.

train everything from scratch, per task



reuse one pretrained backbone

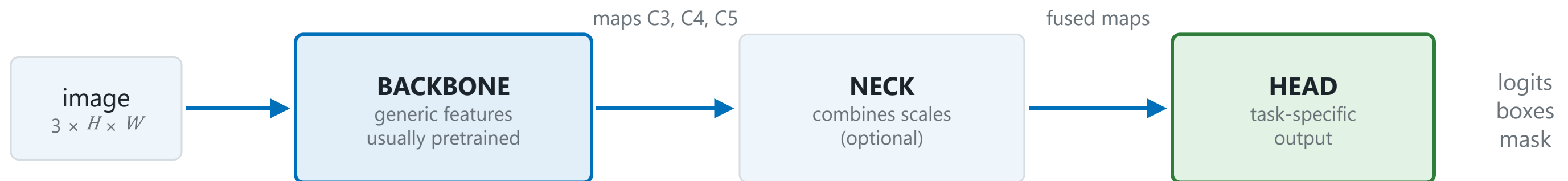


only the small head must be learned from the (small) task dataset

# Backbone, Neck, Head

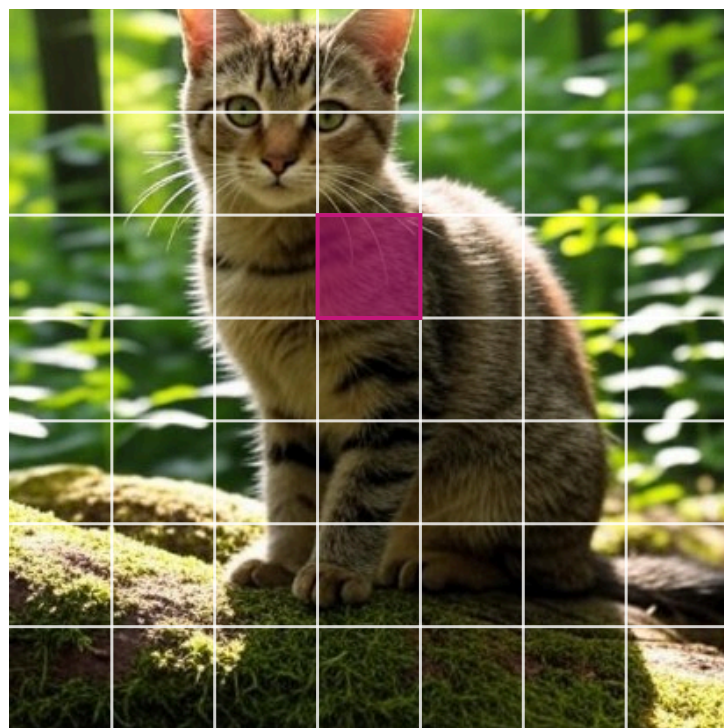
Part 1 · One network, many tasks

- Backbone: generic feature extractor, usually pretrained.
- Neck: optionally fuses maps of different scales.
- Head: turns features into the required output structure.



interfaces between blocks are tensors: their shapes, strides, and meaning

- A backbone exposes several maps, named by stage: C2 ... C5.
- Stride = how many input pixels one cell of the map corresponds to.
- Fine levels keep small details; coarse levels see large context.



**ResNet-50 feature maps for a  $224 \times 224$  input**

| level     | stride | size           | channels |
|-----------|--------|----------------|----------|
| <b>C2</b> | 4      | $56 \times 56$ | 256      |
| <b>C3</b> | 8      | $28 \times 28$ | 512      |
| <b>C4</b> | 16     | $14 \times 14$ | 1024     |
| <b>C5</b> | 32     | $7 \times 7$   | 2048     |

one C5 cell (magenta) summarizes a  $32 \times 32$  pixel patch

# Check question

Think about blocks

**You switch from classifying parts to segmenting their defects. Which blocks change?**

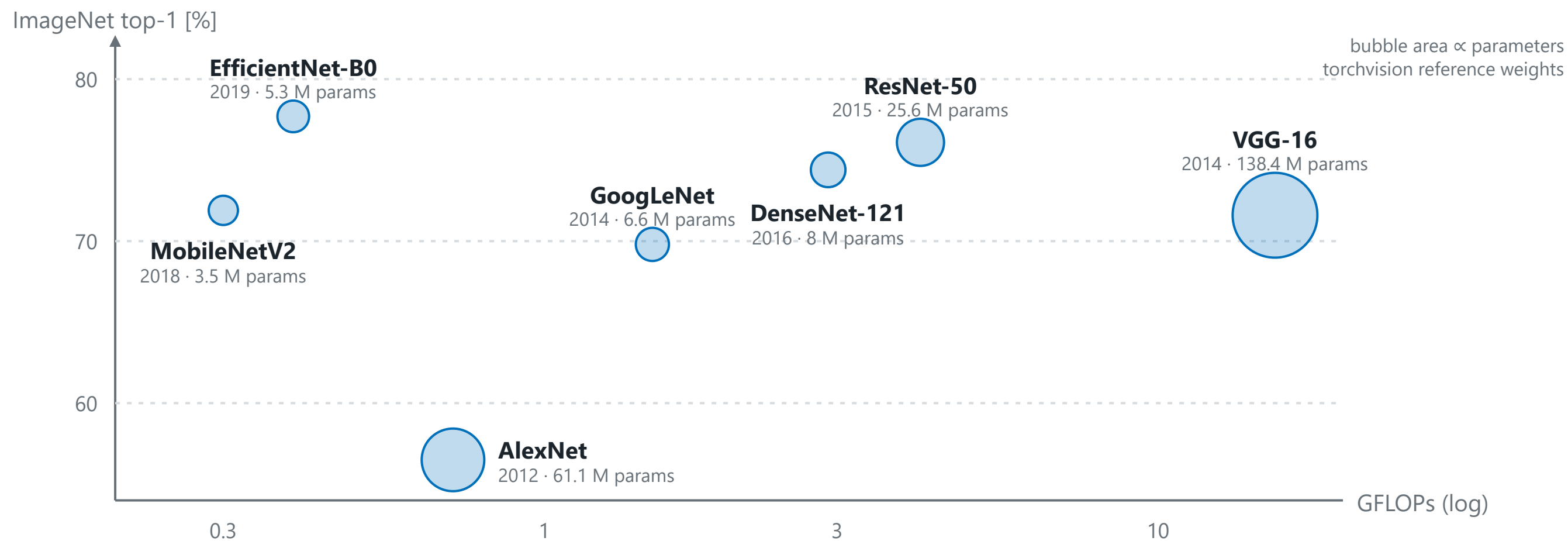
The head certainly (scores  $\rightarrow$  per-pixel mask), usually also the neck (fine levels are needed). The pretrained backbone can stay.



# Seven Years of Backbones

## Part 2 · Backbones

- Accuracy rose by 20 points; compute first exploded, then dropped below AlexNet.
- Each step solved a concrete problem: depth, optimization, cost.
- We follow that story: VGG → Inception → ResNet → DenseNet → MobileNet → EfficientNet.

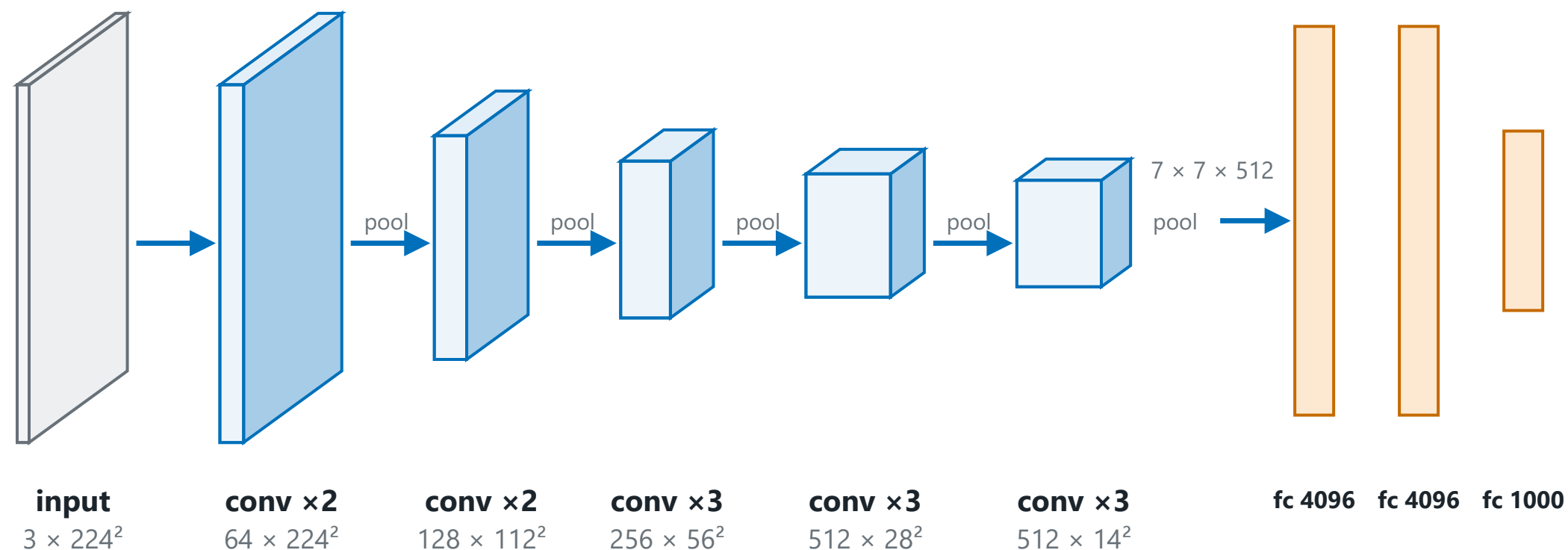


# VGG: One Block, Repeated

Part 2 · Backbones

- Five stages of identical  $3 \times 3$  convolutions, each followed by  $2 \times 2$  max pooling.
- Very regular and easy to read — still a good mental model.
- Its weakness is the end: three huge fully connected layers.

each stage: same  $3 \times 3$  convolutions, then  $2 \times 2$  max pooling → resolution  $\frac{1}{2}$ , channels  $\times 2$

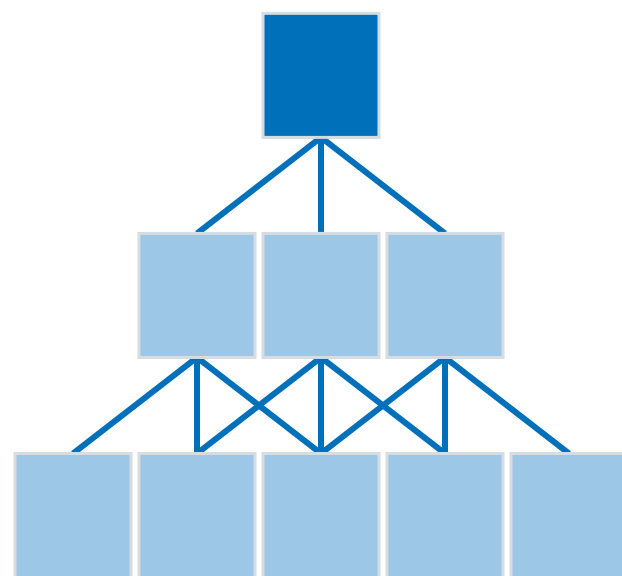


three dense layers:  
124 M of 138 M  
parameters

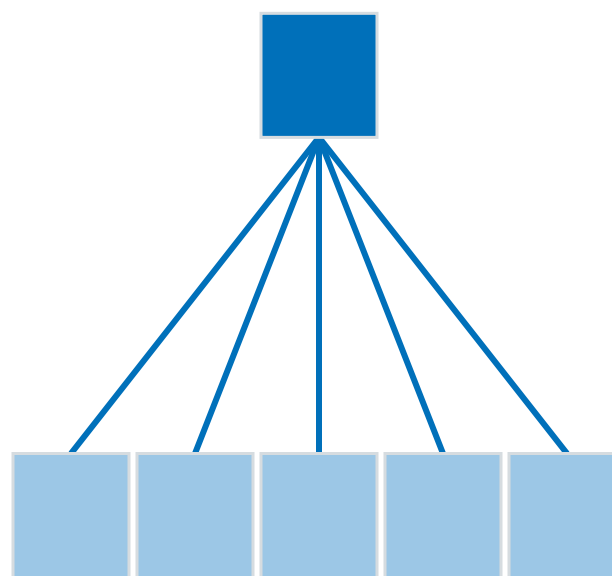
# Why Only $3 \times 3$ Kernels?

Part 2 · Backbones

- Two stacked  $3 \times 3$  layers see the same  $5 \times 5$  region as one  $5 \times 5$  layer (lecture 1).
- They need 28 % fewer weights and add a second nonlinearity.
- Three  $3 \times 3$  layers replace a  $7 \times 7$  in the same way.



two  $3 \times 3$  layers: field 5



one  $5 \times 5$  layer: field 5

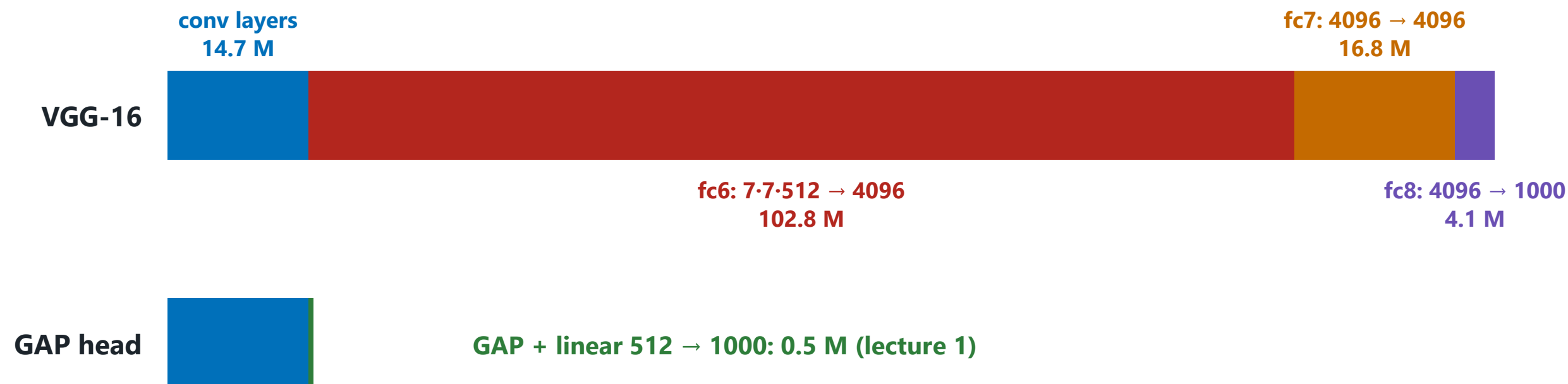
same view, different cost

|                                                       |                             |
|-------------------------------------------------------|-----------------------------|
| weights ( $C \rightarrow C$ )                         |                             |
| $2 \cdot 9 \mathcal{C}^2 = \mathbf{18 \mathcal{C}^2}$ | $\mathbf{25 \mathcal{C}^2}$ |
| nonlinearities                                        |                             |
| $\mathbf{2}$                                          | $\mathbf{1}$                |
| $3 \times 3 + 3 \times 3$                             | $5 \times 5$                |

# Where VGG's Parameters Are

## Part 2 · Backbones

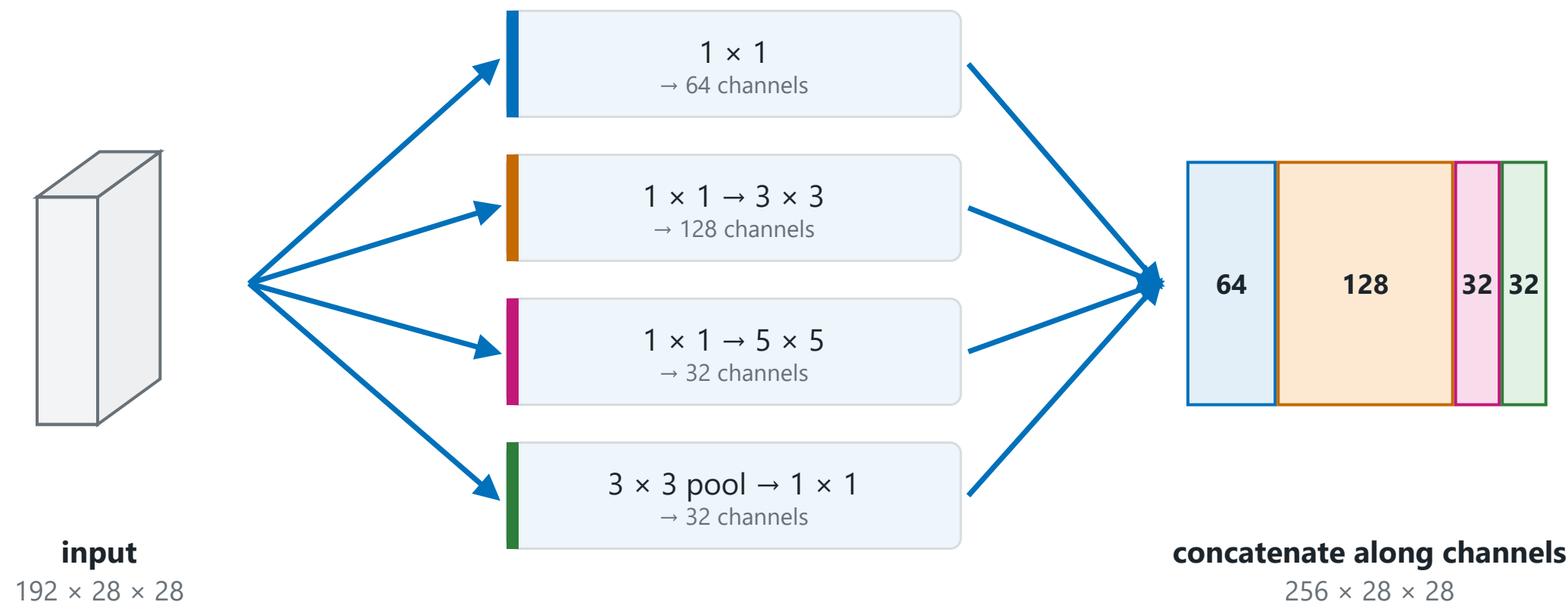
- The convolutions hold only 15 M of 138 M parameters.
- The first dense layer alone ( $7 \cdot 7 \cdot 512 \rightarrow 4096$ ) holds 103 M.
- Later networks replace it with global average pooling — the head from lecture 1.



# Inception: Several Kernel Sizes at Once

Part 2 · Backbones

- Objects appear at different sizes; which kernel size is right is not known in advance.
- Inception runs  $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$ , and pooling branches in parallel on the same input.
- The results are concatenated along the channel axis.



each branch sees  
the same input at a  
different scale

# The $1 \times 1$ Reduction Trick

Part 2 · Backbones

- A  $5 \times 5$  convolution on 192 channels is expensive.
- A  $1 \times 1$  convolution first reduces the channels (lecture 1: it mixes channels cheaply).
- The same output costs about ten times less.

input  $192 \times 28 \times 28 \rightarrow 32$  channels with a  $5 \times 5$  convolution

direct  $5 \times 5$



120.4 M multiply-adds

$1 \times 1$  to 16, then  $5 \times 5$



$2.4 \text{ M} + 10.0 \text{ M} = 12.4 \text{ M}$  ( $\approx 10\times$  cheaper)

$$28 \cdot 28 \cdot 25 \cdot 192 \cdot 32 = 120.4 \text{ M} \text{ vs } 28 \cdot 28 \cdot 192 \cdot 16 + 28 \cdot 28 \cdot 25 \cdot 16 \cdot 32 = 12.4 \text{ M}$$

## When does processing several kernel sizes in parallel pay off?

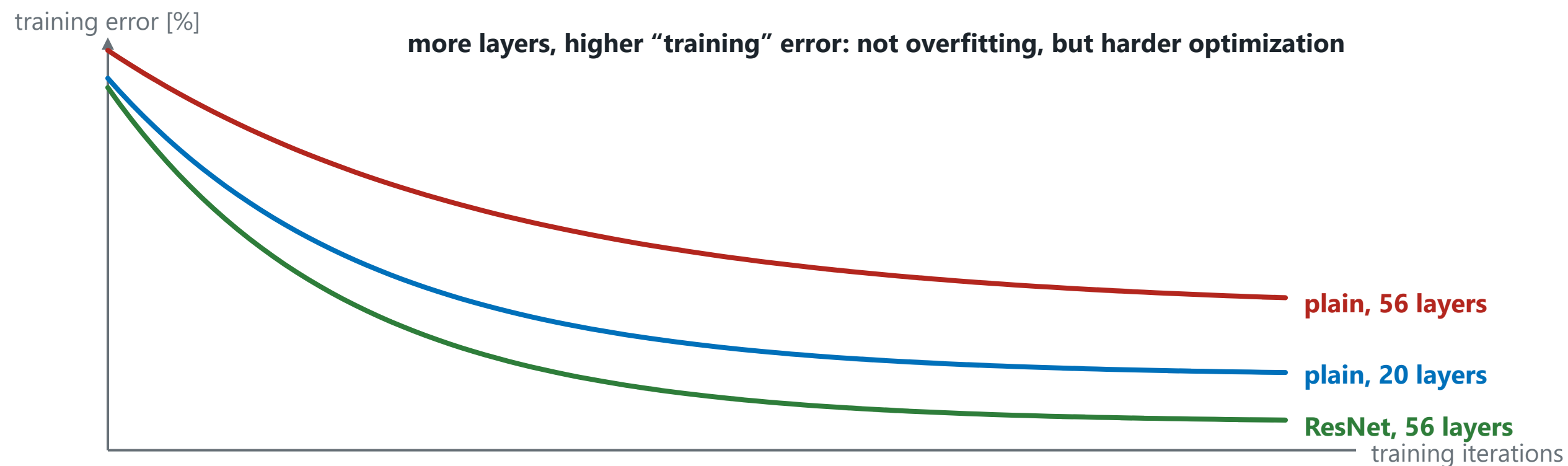
When relevant structures have very different sizes within one image and a single kernel size would cover them poorly.



# Deeper Is Not Automatically Better

Part 2 · Backbones

- Stacking more plain layers made even the *training* error worse.
- This is not overfitting: the deeper model could copy the shallow one, but the optimizer does not find it.
- We need a way to make very deep networks trainable.



schematic, after He et al. 2016 (CIFAR-10)

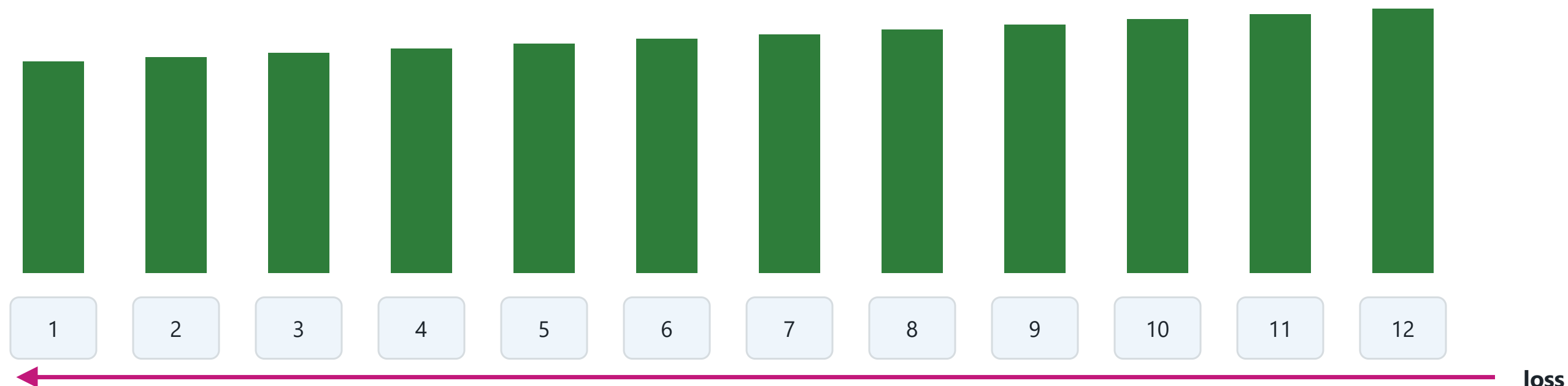
# Why Plain Deep Networks Are Hard to Train

Part 2 · Backbones

- Backpropagation multiplies local derivatives layer by layer (lecture 1).
- If each factor is a bit below one, the gradient fades before it reaches the early layers.
- A shortcut that passes the gradient unchanged keeps all layers learning.

**gradient magnitude reaching each layer (the loss is on the right)**

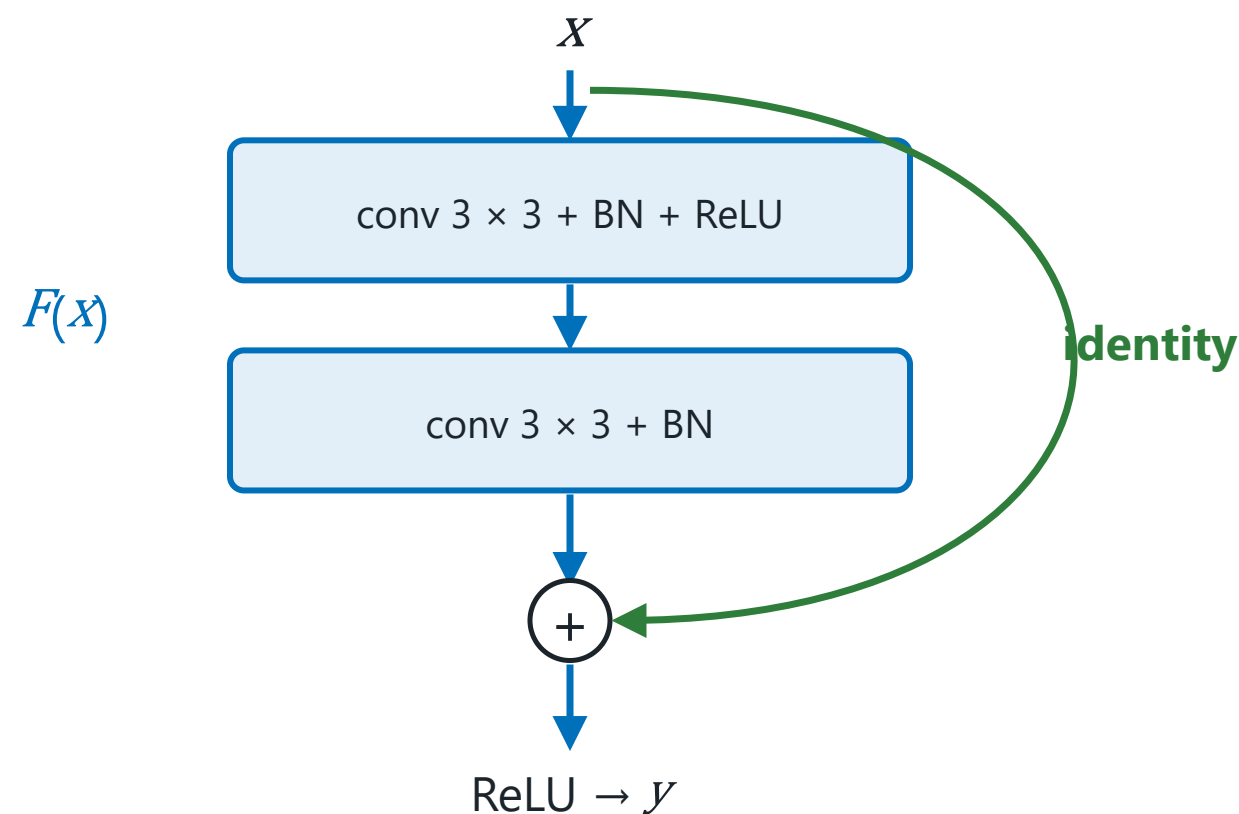
residual: the identity path passes the gradient on almost unchanged



# The Residual Block

## Part 2 · Backbones

- The block adds its input to the output of two convolutions.
- The layers only learn what to change; doing nothing is easy.
- This single idea enabled networks with 50, 100, and more layers.



$$y = F(x) + x$$

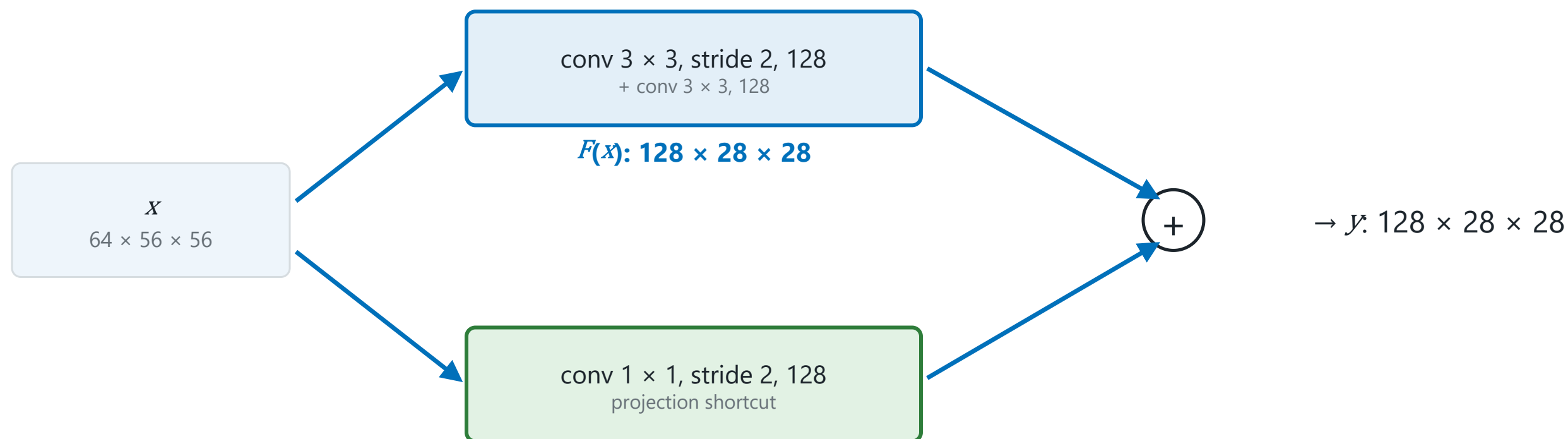
the block only has to learn  
the correction  $F(x) = y - x$

if  $F \rightarrow 0$ , the block is the identity:  
extra depth cannot hurt the start

# When Shapes Do Not Match

Part 2 · Backbones

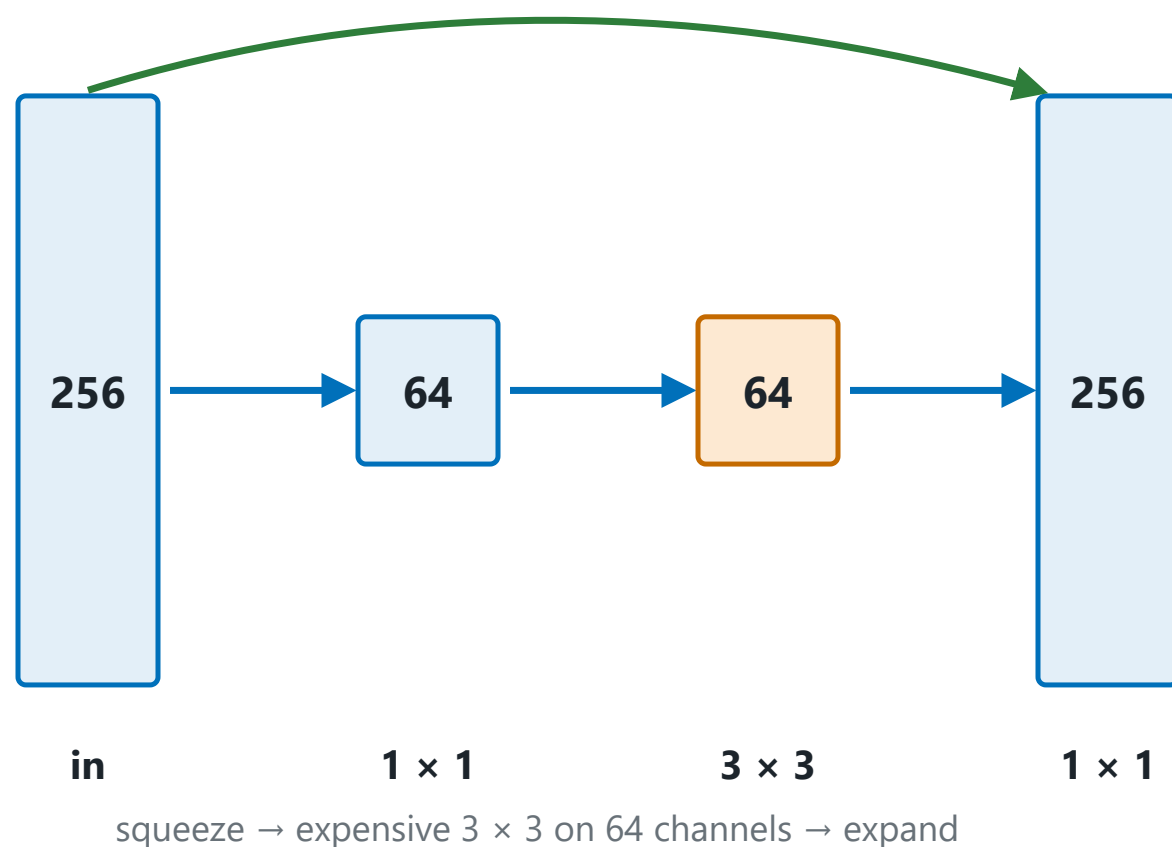
- Adding requires identical shapes (lecture 1: shapes are part of the design).
- At a stage boundary, the main path halves the resolution and doubles the channels.
- A  $1 \times 1$  convolution with stride 2 on the shortcut adapts the input.



# The Bottleneck Block

## Part 2 · Backbones

- Deep ResNets have 256 or more channels; a  $3 \times 3$  there is costly.
- $1 \times 1$  squeezes to 64 channels, the  $3 \times 3$  works there,  $1 \times 1$  expands back.
- $17 \times$  fewer weights per block — this is what ResNet-50 and deeper use.



### weights per block (256 channels)

two  $3 \times 3$

$$2 \cdot 9 \cdot 256^2 = 1\,179\,648$$

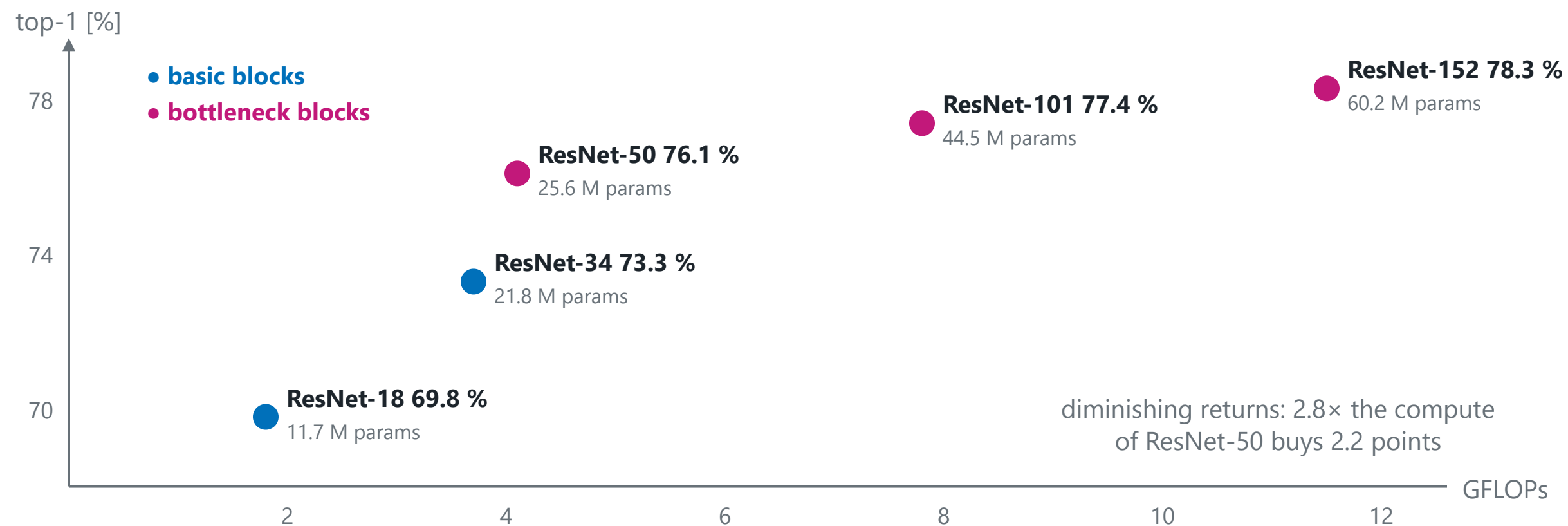
bottleneck

$$256 \cdot 64 + 9 \cdot 64^2 + 64 \cdot 256 = 69\,632 \text{ (17} \times \text{ fewer)}$$

# The ResNet Family

## Part 2 · Backbones

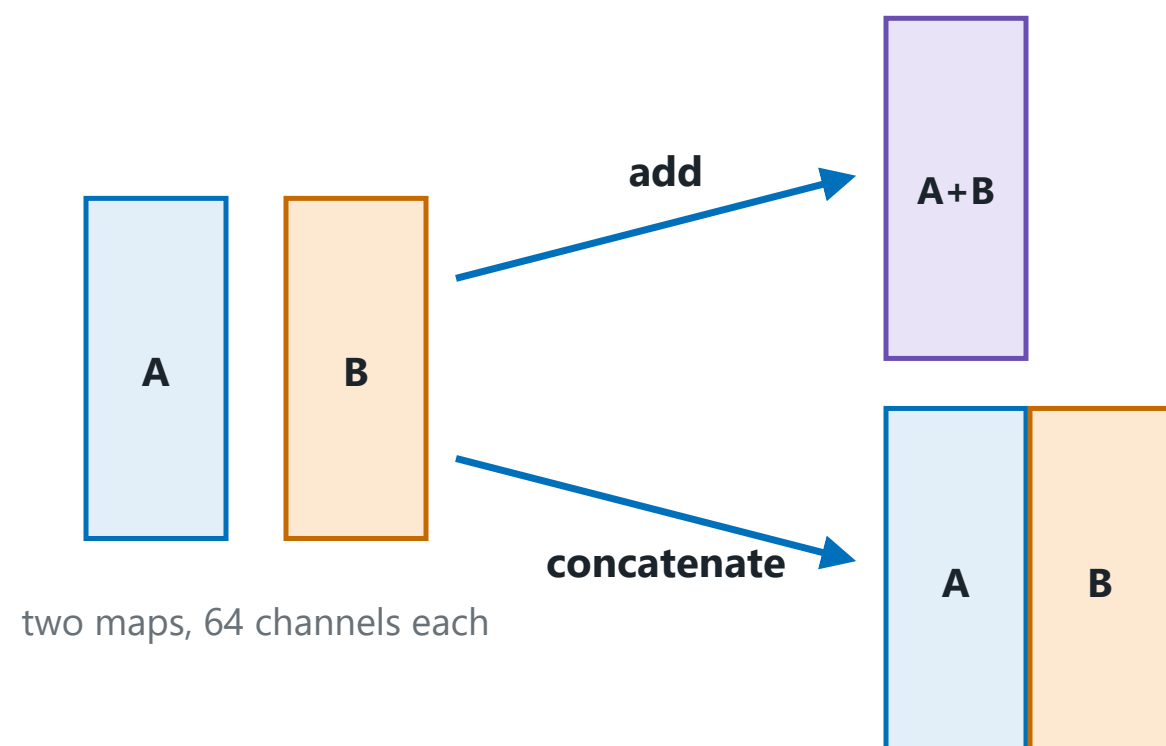
- ResNet-18/34 use basic blocks; ResNet-50/101/152 use bottlenecks.
- Accuracy grows with depth, but each step costs more for less gain.
- ResNet-50 is the most common default backbone to start with.



# Two Ways to Merge Tensors

Part 2 · Backbones

- Adding keeps the number of channels and requires identical shapes.
- Concatenating stacks the channels and only requires the same height and width.
- Every architecture diagram is built from these two merges — look for them.



64 channels — shapes must match exactly  
used in ResNet, FPN

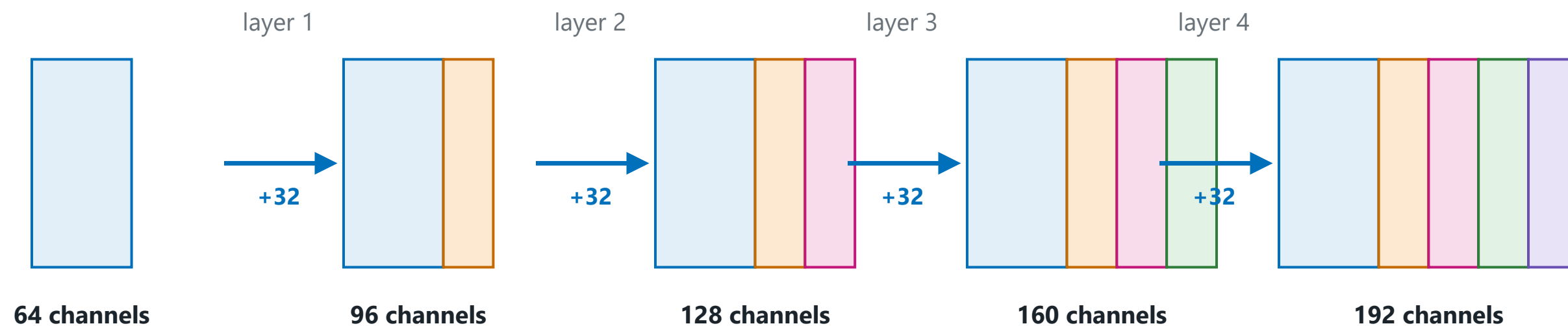
128 channels — only  $H \times W$  must match  
used in Inception, DenseNet, U-Net

# DenseNet: Concatenate Everything

## Part 2 · Backbones

- Inside a block, each layer receives the concatenated outputs of all earlier layers.
- It adds only a few new channels (growth rate  $k$ ), so it stays small.
- Few parameters, but the growing concatenations cost memory bandwidth.

**each layer sees all previous maps (concatenated) and adds  $k = 32$  new channels**

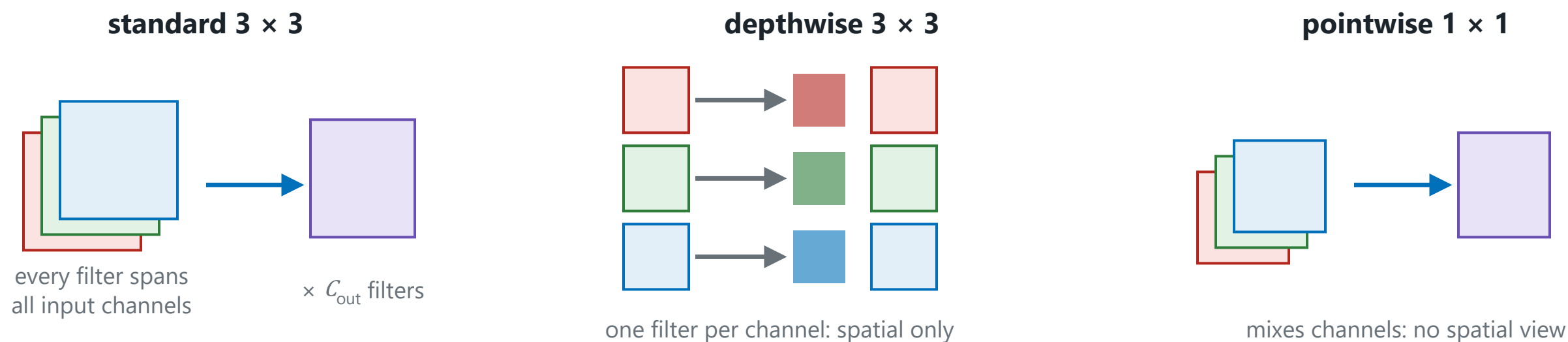


features are reused instead of re-learned · cost: many concatenations, high memory traffic

# Depthwise Separable Convolution

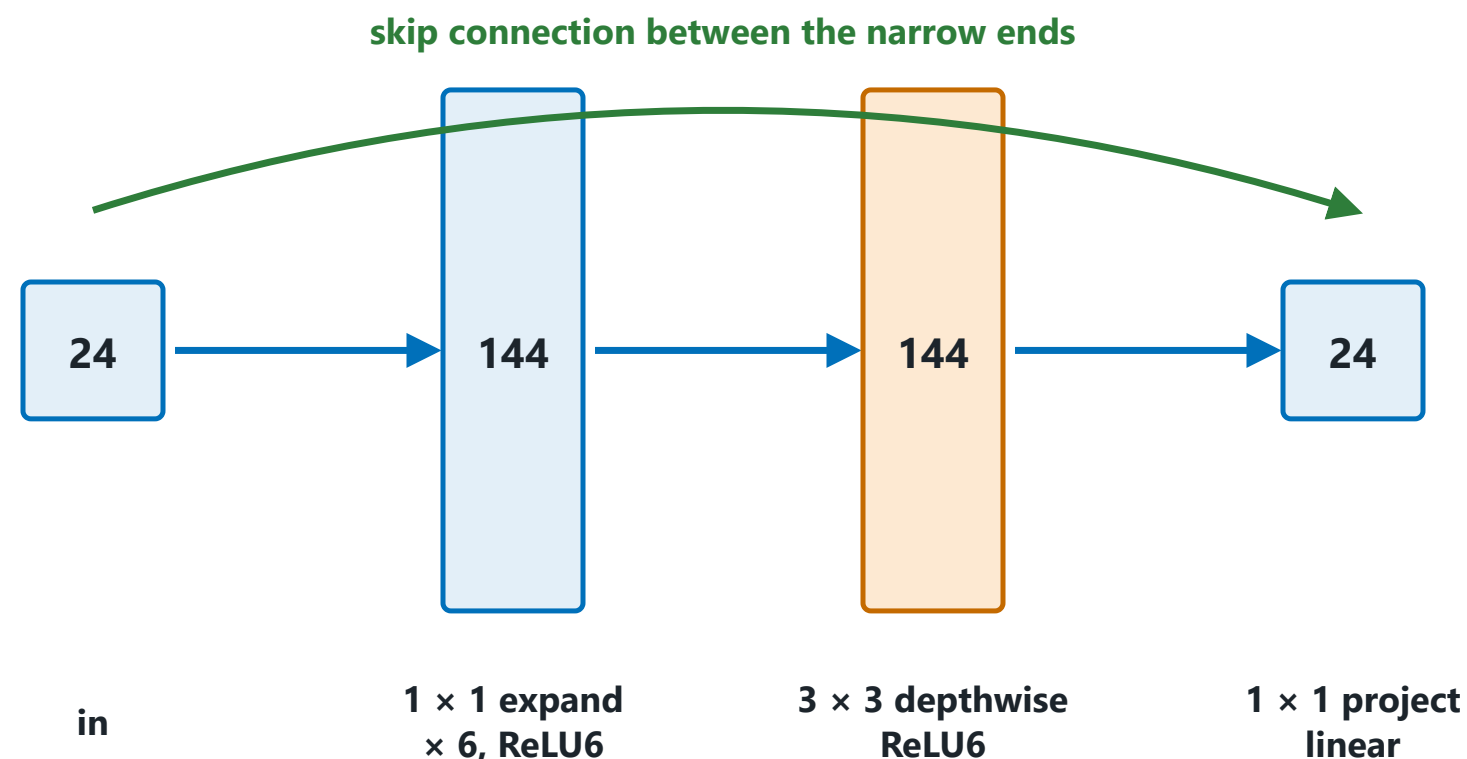
Part 2 · Backbones

- A standard convolution filters space and mixes channels in one expensive step.
- Split it: a depthwise  $3 \times 3$  per channel, then a pointwise  $1 \times 1$  to mix channels.
- Almost the same expressive power for about 1/8 of the cost.



|           |                                             |                                                   |
|-----------|---------------------------------------------|---------------------------------------------------|
| standard  | <b>462 M</b>                                |                                                   |
| separable | <b>3.6 M + 51.4 M = 55 M (8.4× cheaper)</b> | example: 128 → 128<br>channels, 56 × 56<br>output |

- Expand a narrow tensor, filter it with a cheap depthwise conv, project it back.
- The skip connection joins the narrow ends — the reverse of a bottleneck.
- Built for phones and embedded devices.



inverted residual (MobileNetV2):  
narrow → wide → narrow

the expensive part is depthwise,  
so the wide middle is cheap

no ReLU after projection:  
a narrow layer would lose information

# Two Knobs: Width and Resolution

## Part 2 · Backbones

- Width  $\alpha$  scales all channel counts; cost and parameters scale with  $\alpha^2$ .
- Input resolution scales cost quadratically but not the parameters.
- Try it: which knob would you turn for small defects, which for a tight latency budget?

MobileNetV2 at width  $\alpha$  and input resolution  $r$  (reference:  $\alpha = 1$ ,  $r = 224$ : 300 M multiply-adds, 3.5 M parameters)

multiply-adds  $\propto \alpha^2 r^2$



**300 M MACs**

parameters  $\propto \alpha^2$

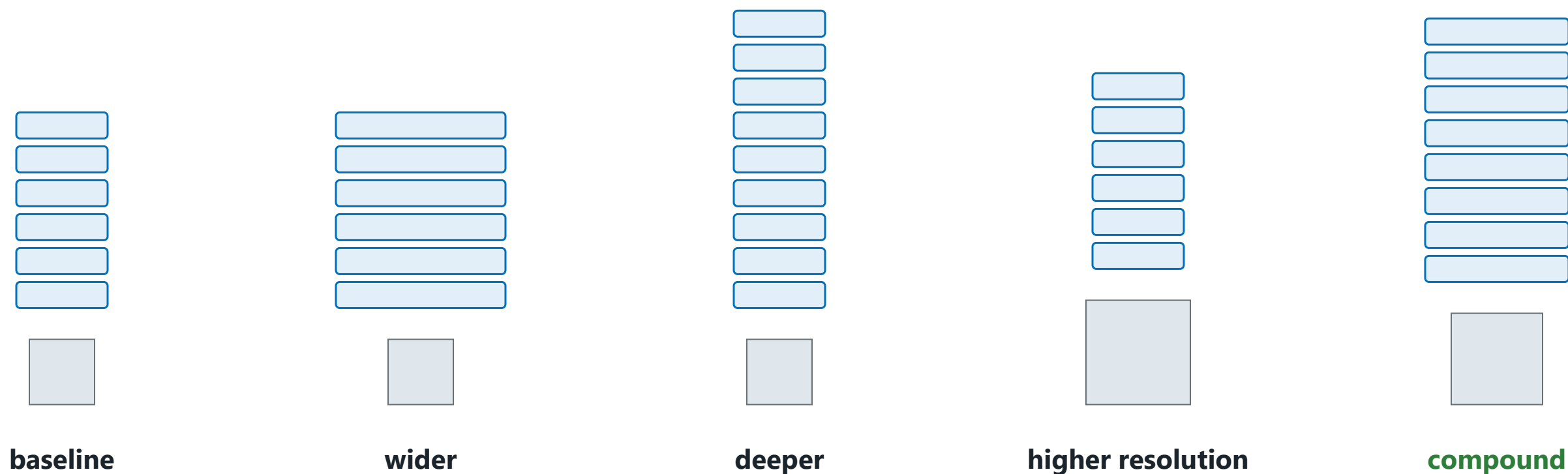


**3.50 M params**

# EfficientNet: Scale Everything Together

## Part 2 · Backbones

- Making a network only wider, only deeper, or only higher-resolution saturates quickly.
- EfficientNet grows all three together with fixed ratios (compound scaling).
- B0 ... B7 form one family from small to large.

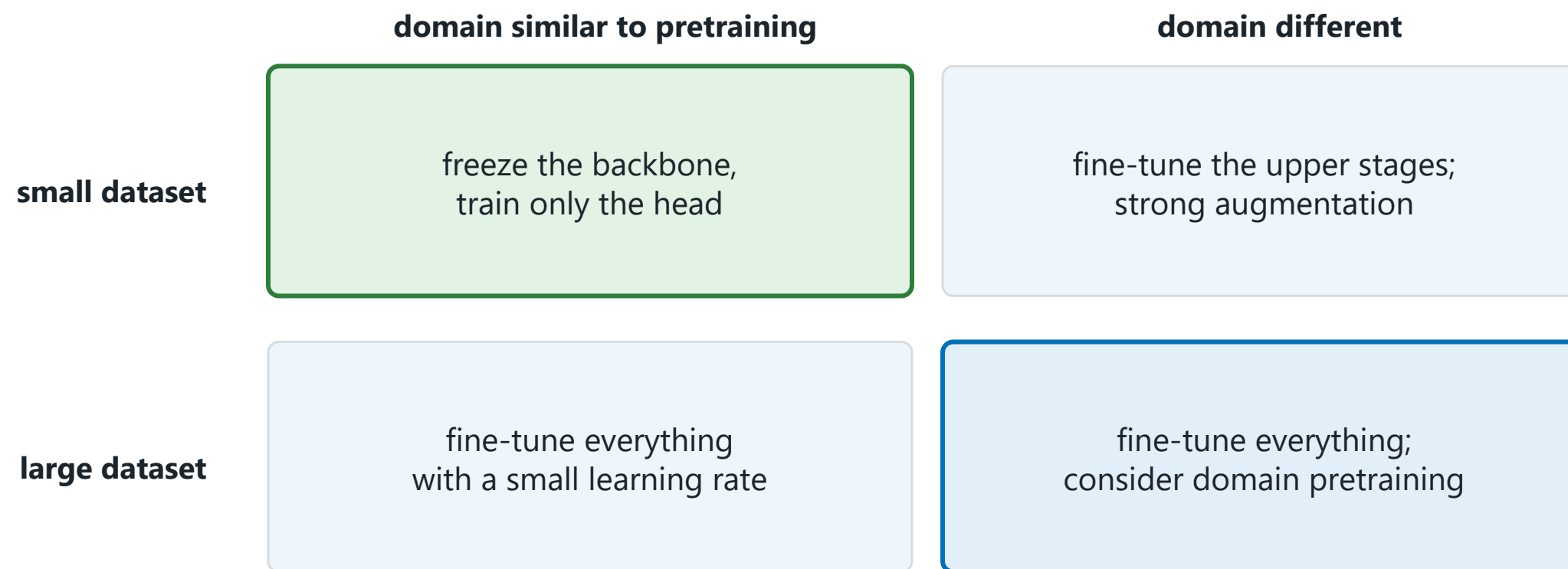


EfficientNet: depth  $\propto 1.2^\varphi$ , width  $\propto 1.1^\varphi$ , resolution  $\propto 1.15^\varphi$  for a chosen  $\varphi$

# Reusing a Pretrained Backbone

## Part 2 · Backbones

- How much to fine-tune depends on dataset size and on how close the domain is to ImageNet.
- Industrial images (grayscale, X-ray, microscopy) are often far from it.
- Always compare with a frozen backbone: it is the cheapest baseline.



early layers transfer best (edges, textures); late layers are most task-specific

# Check question

Choose a backbone

**An inspection camera runs on an embedded board without a GPU. Which backbone family do you try first, and what do you measure?**

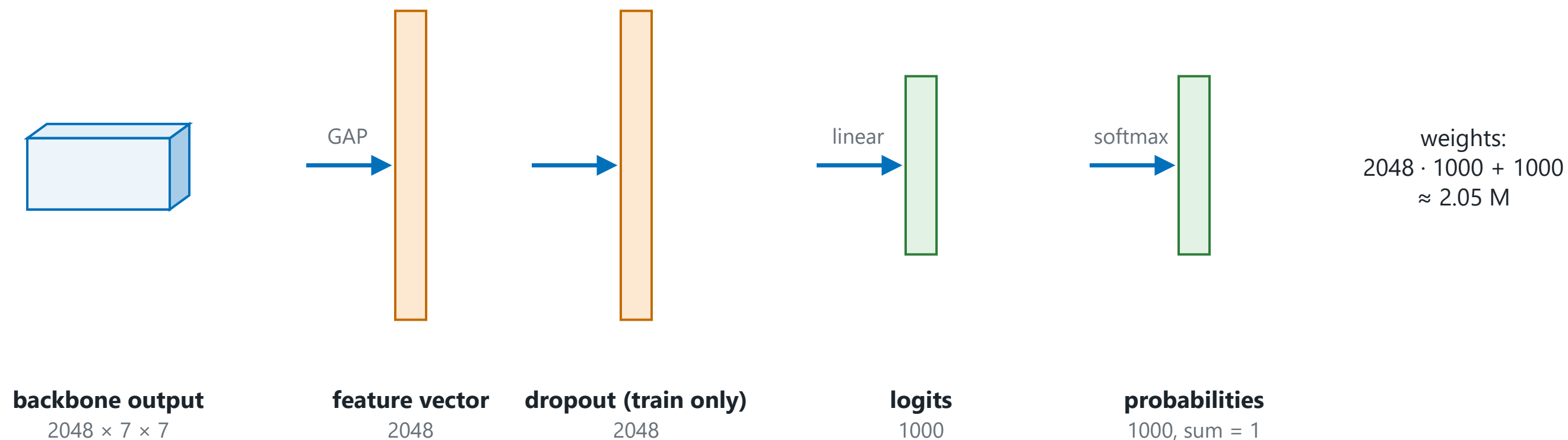
MobileNet or EfficientNet-B0 (depthwise separable). Measure end-to-end latency on the board itself, not FLOPs on a PC.



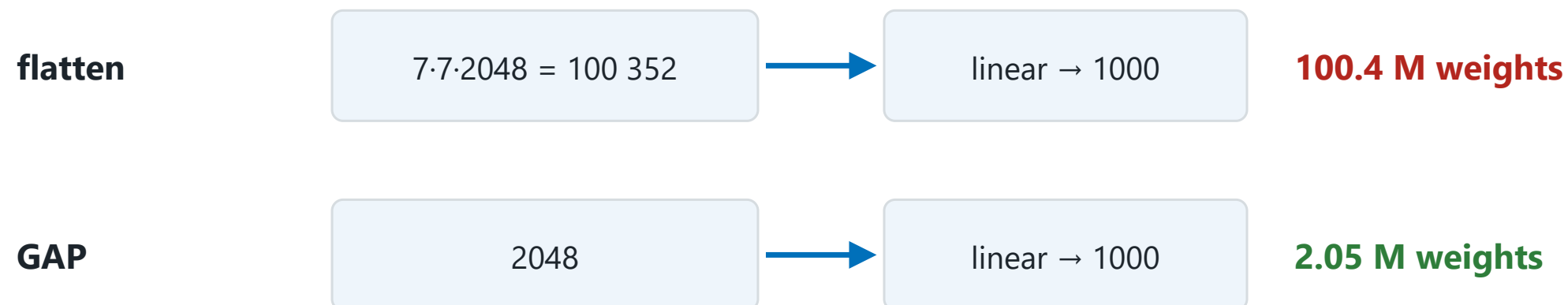
# The Classification Head

## Part 3 · Heads

- Global average pooling turns the last map into one vector per image.
- A linear layer maps it to class logits; softmax gives probabilities.
- Dropout before the linear layer helps on small datasets.



- Flattening keeps position but makes the head  $50 \times$  larger.
- GAP discards position and works for any input resolution.
- For classification GAP is the standard; for localization neither is enough.



**now feed a  $448 \times 448$  image: the last map becomes  $2048 \times 14 \times 14$**

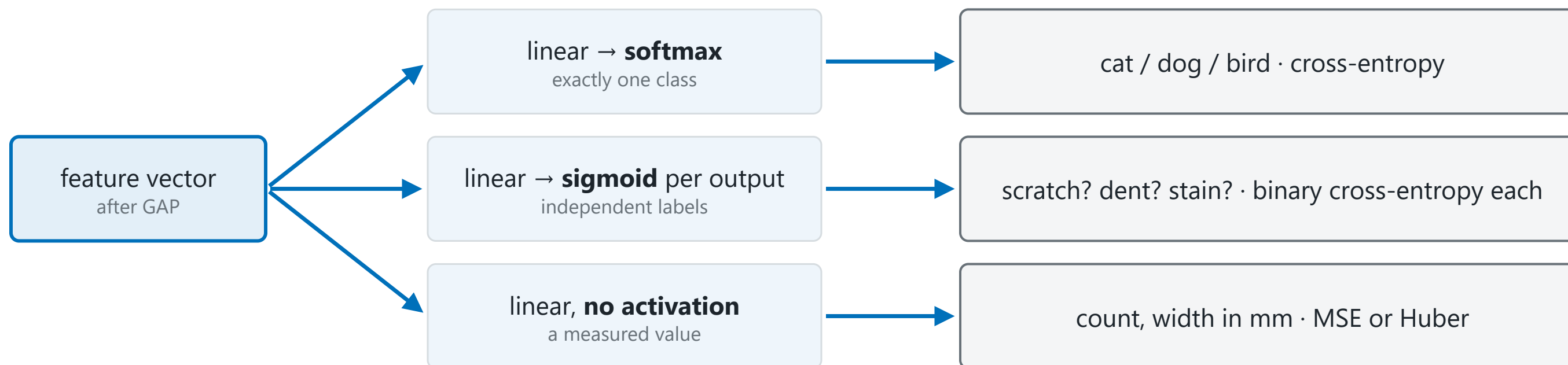
flatten  $\rightarrow 401\,408$  values  $\neq 100\,352$ : the linear layer no longer fits

GAP  $\rightarrow$  still 2048 values: the same head works at any resolution

# One Feature Vector, Several Head Types

## Part 3 · Heads

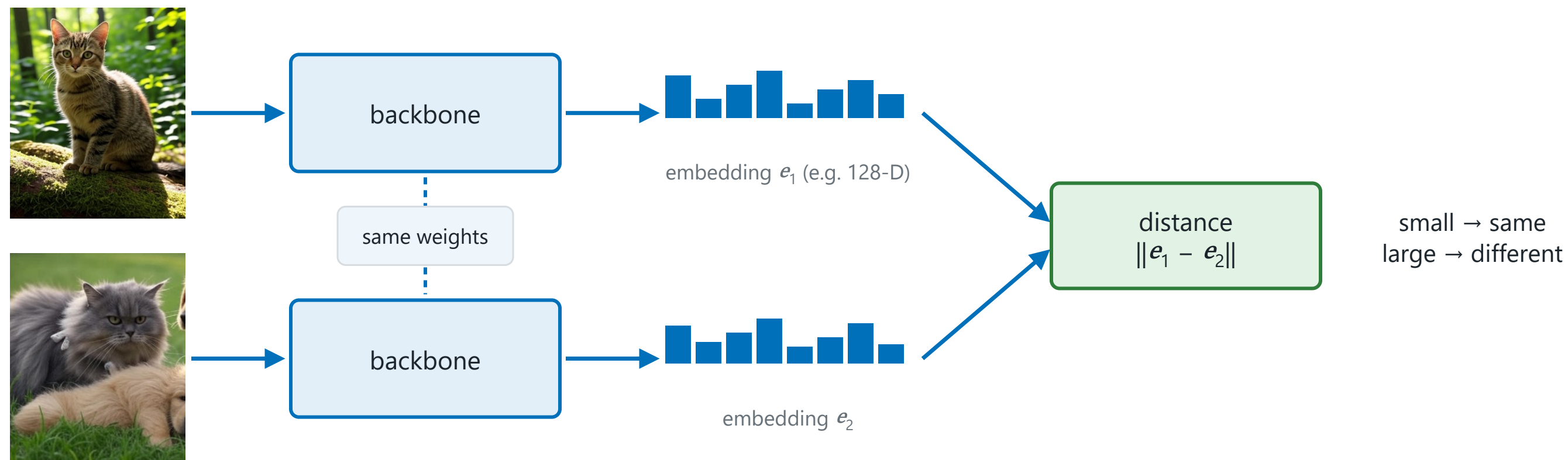
- The output activation and the loss must match what the label means (lecture 1).
- Several heads can share one backbone (multi-task learning).
- The head is small: changing it is cheap.



# Embedding Heads: Learning Similarity

## Part 3 · Heads

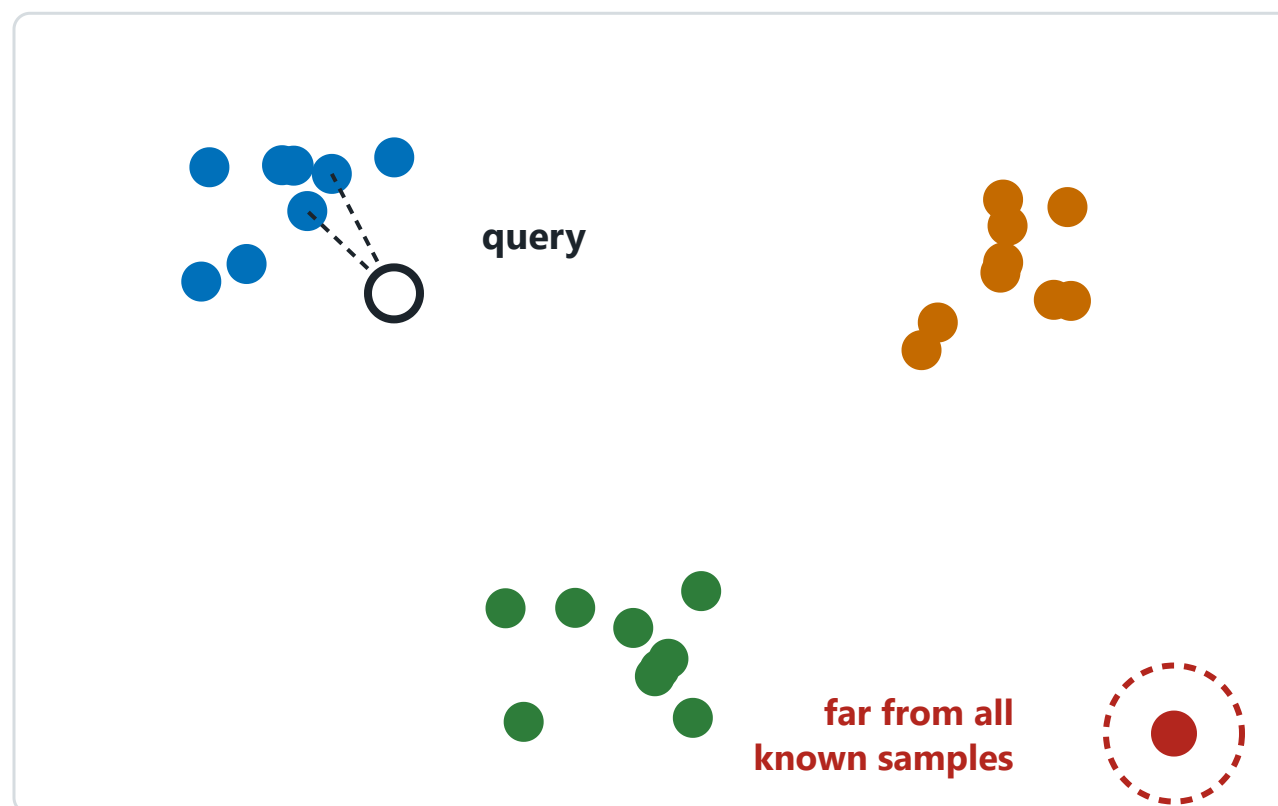
- Instead of class scores, the head outputs a vector (embedding).
- Two images pass through the same backbone; their distance measures similarity.
- New classes need no retraining: just compare with stored examples.



# What an Embedding Space Is Good For

Part 3 · Heads

- Training arranges the space so that similar images are close.
- Then search, clustering, and anomaly detection are just distance computations.
- Useful in inspection: only “good” parts are needed to flag unusual ones.



**anomaly detection: distance to normal samples**

triplet loss (anchor  $a$ , positive  $p$ , negative  $n$ ):  
 $\max(0, d(a, p) - d(a, n) + \text{margin})$

uses: retrieval, clustering,  
re-identification, few-shot,  
anomaly detection

# Check question

Pick a head

**A line produces 300 part types and new types appear every month. Classification head or embedding head?**

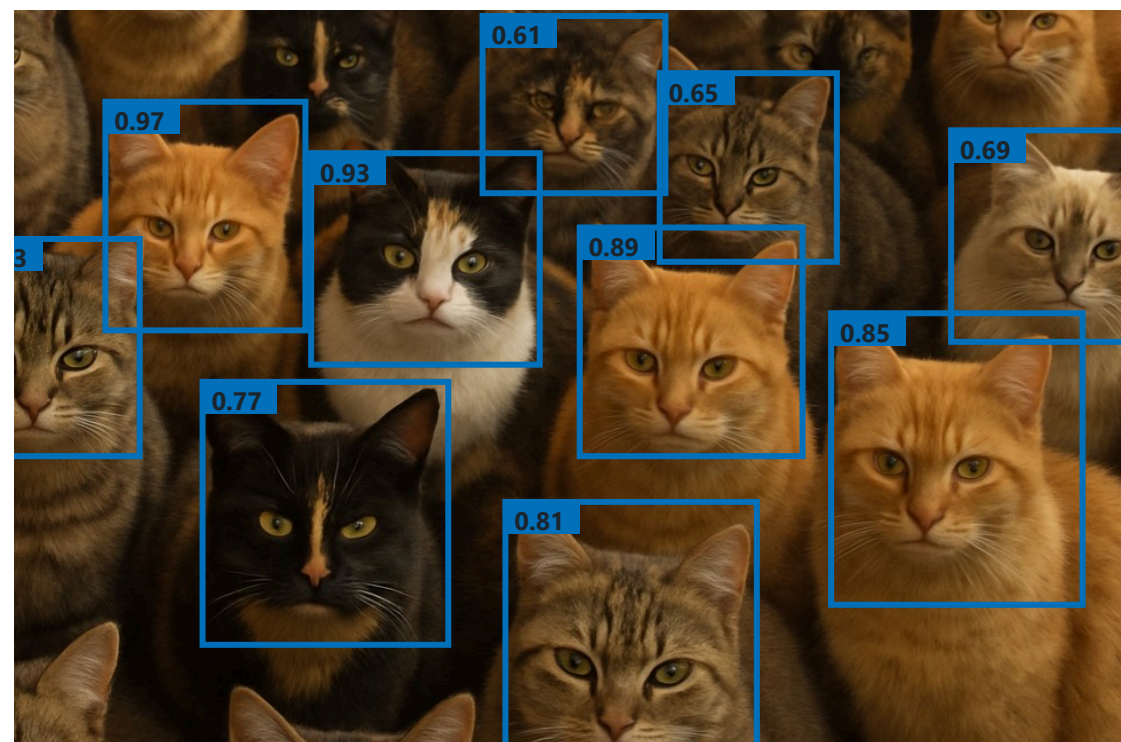
Embedding head: a new type only needs reference images in the database, not a new output neuron and retraining.



# Detection Changes the Output

## Part 4 · Detection

- The number of objects is unknown and different in every image.
- Each object needs a class, a confidence score, and four box coordinates.
- A fixed-size output layer cannot produce this directly.

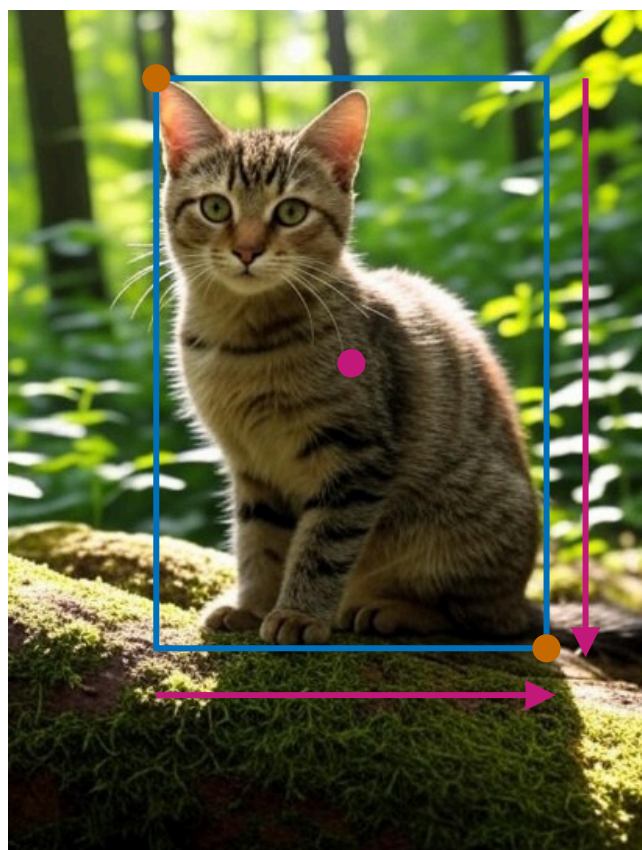


the output is a list whose length  $N$  varies

| class | score | $x_1$ | $y_1$ | $x_2$ | $y_2$ |
|-------|-------|-------|-------|-------|-------|
| cat   | 0.97  | 120   | 460   | 295   | 660   |
| cat   | 0.93  | 300   | 505   | 500   | 690   |
| cat   | 0.89  | 535   | 570   | 730   | 770   |
| cat   | 0.85  | 755   | 645   | 975   | 900   |
| cat   | 0.81  | 470   | 810   | 690   | 1024  |
| cat   | 0.77  | 205   | 705   | 420   | 935   |
| cat   | 0.73  | 0     | 580   | 150   | 770   |
| cat   | 0.69  | 860   | 485   | 1024  | 670   |
| cat   | 0.65  | 605   | 435   | 760   | 600   |
| cat   | 0.61  | 450   | 385   | 610   | 540   |

$N = 10$ : each object = class + score + 4 box numbers

- Two corners, or a centre plus width and height: the same box.
- Datasets and libraries differ (pixels vs. 0–1, corners vs. centre).
- A wrong format is a common, silent bug.



**corners**

$$(x_1, y_1, x_2, y_2) = (95, 45, 345, 410)$$

**centre + size**

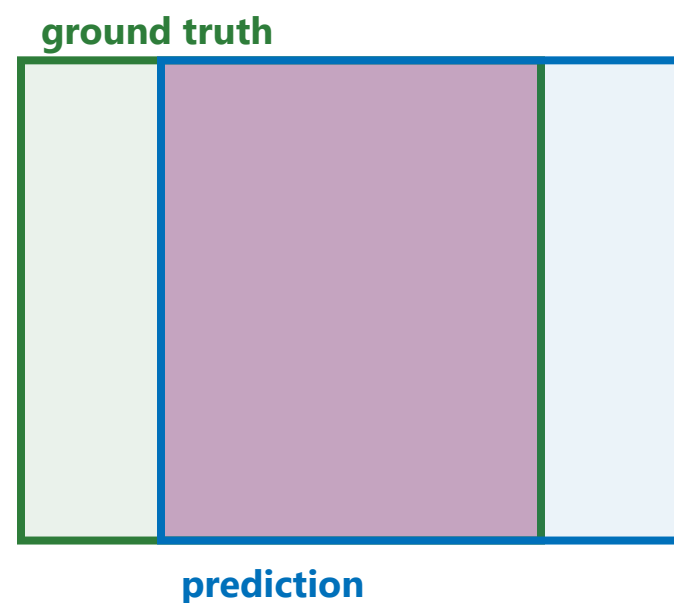
$$(c_x, c_y, w, h) = (220, 227.5, 250, 365)$$

same box, different numbers: always check the format  
and whether coordinates are in pixels or normalized to 0–1

# Intersection over Union

Part 4 · Detection

- IoU measures how well a predicted box overlaps the true one: 0 = disjoint, 1 = identical.
- It decides whether a detection counts as correct, and drives duplicate removal.
- Move the prediction and watch how fast IoU drops.



$$\text{IoU} = 0.58$$

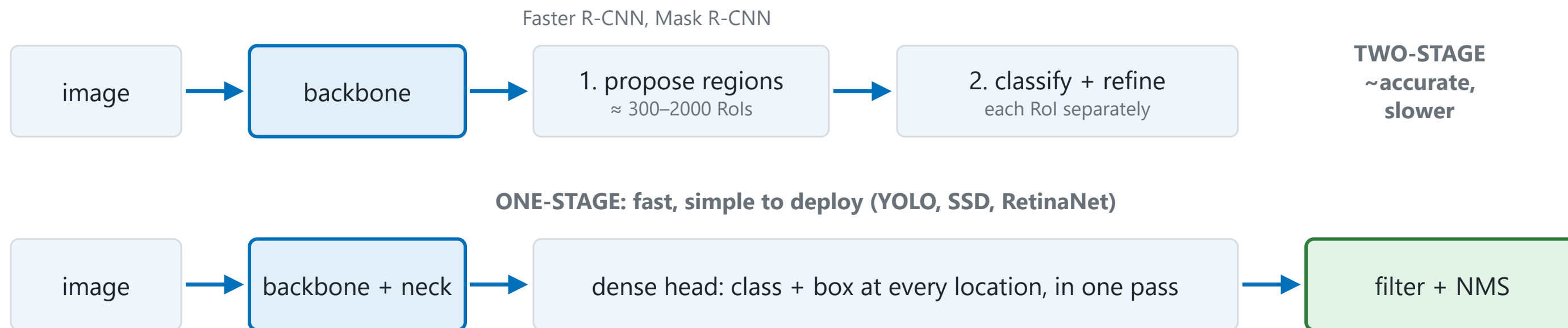
**counts as a correct detection**

$\text{IoU} = \text{area}(\cap) / \text{area}(\cup)$   
match if  $\text{IoU} \geq 0.5$  (a common choice)

# Two Families of Detectors

## Part 4 · Detection

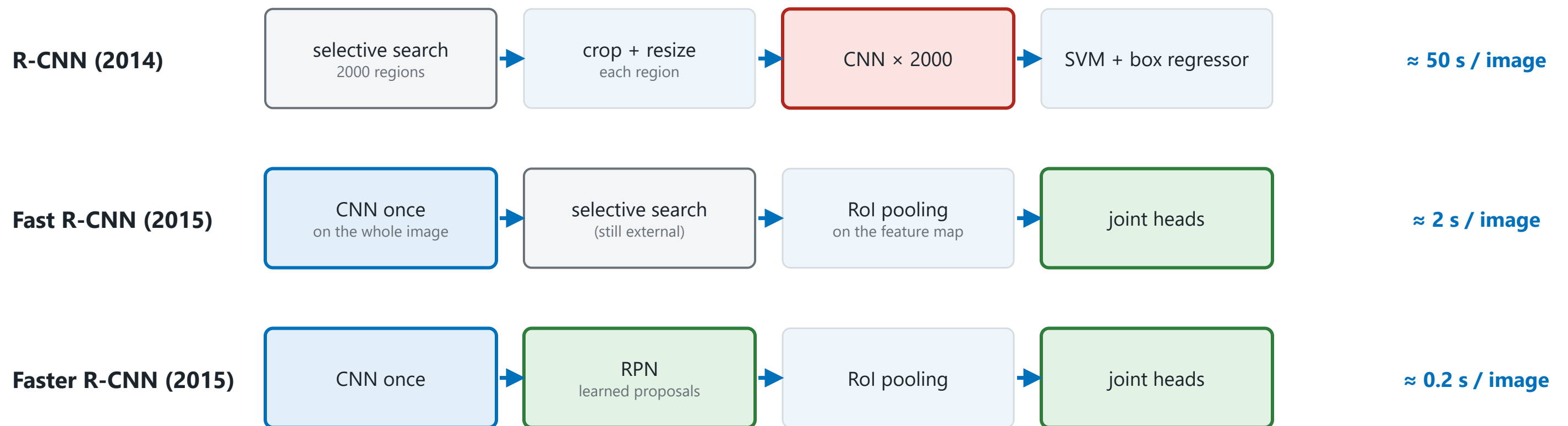
- Two-stage: first propose likely regions, then classify each one.
- One-stage: predict boxes densely at every location in a single pass.
- Both need a final filtering step.



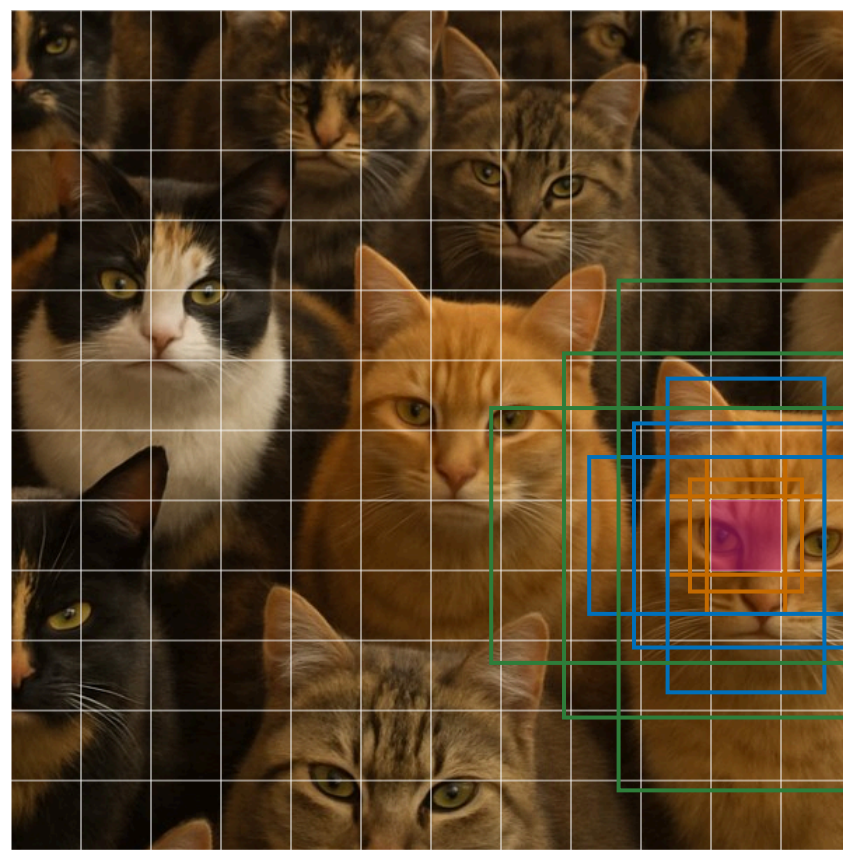
# From R-CNN to Faster R-CNN

## Part 4 · Detection

- R-CNN ran a whole CNN on 2000 crops per image.
- Fast R-CNN ran the CNN once and cropped the feature map instead.
- Faster R-CNN also learned the proposals: fully end to end.



- A small conv network slides over the feature map.
- At each location it scores a set of reference boxes (anchors) and refines them.
- The best few hundred become proposals for the second stage.



## Region Proposal Network

at every feature-map location:

$k = 9$  anchors (3 sizes  $\times$  3 aspect ratios)

outputs per location:

$2k$  objectness scores +  $4k$  box offsets

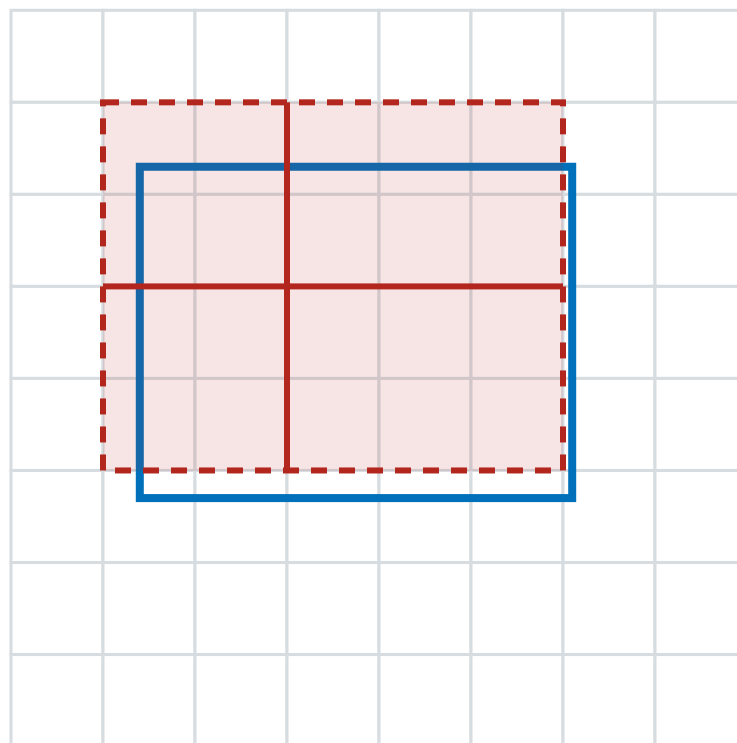
keep the  $\approx 300$  best proposals after NMS

# Rol Pooling and Rol Align

Part 4 · Detection

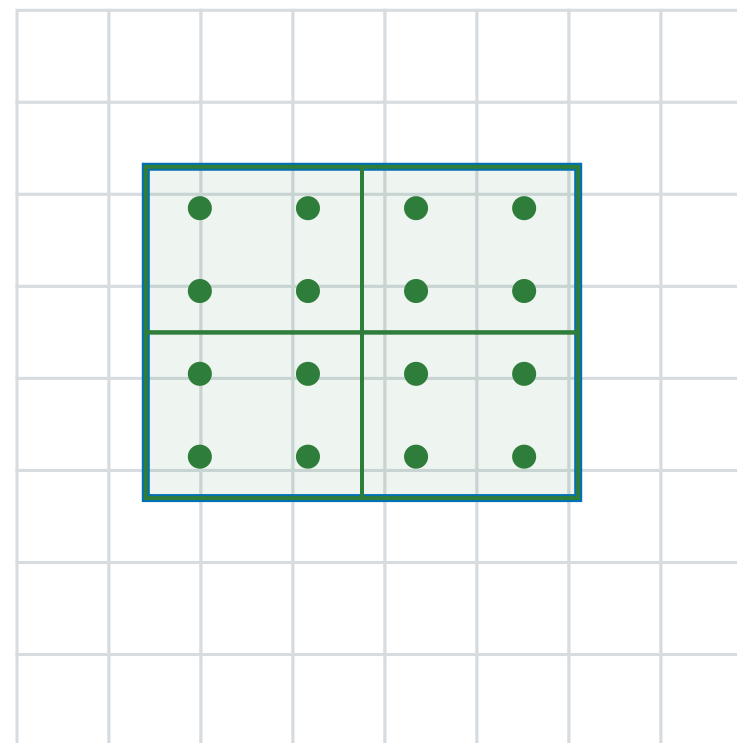
- Proposals have different sizes, but the second-stage head needs a fixed-size input.
- Rol pooling snaps the box to the grid; Rol Align samples it exactly.
- The difference matters for masks and small objects.

coordinates rounded to cells: boundaries shift



**Rol pooling**

exact bins, values sampled by bilinear interpolation



**Rol Align**

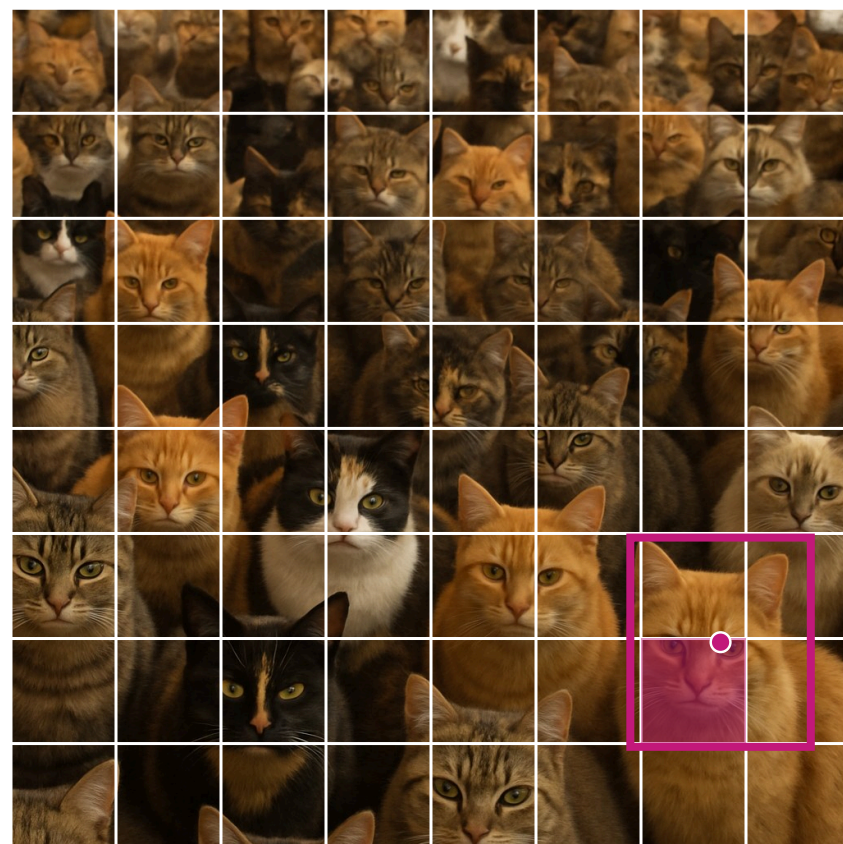
both turn a box of any size  
into a fixed  $7 \times 7$  (here  $2 \times 2$ )  
feature grid for the head

the misalignment of Rol pooling  
hurts pixel-accurate masks

# One-Stage Detection: Predict on a Grid

## Part 4 · Detection

- The final feature map is a grid; each cell is responsible for objects centred in it.
- Each cell predicts boxes, confidences, and class scores directly.
- One forward pass, no proposals: YOLO, SSD.



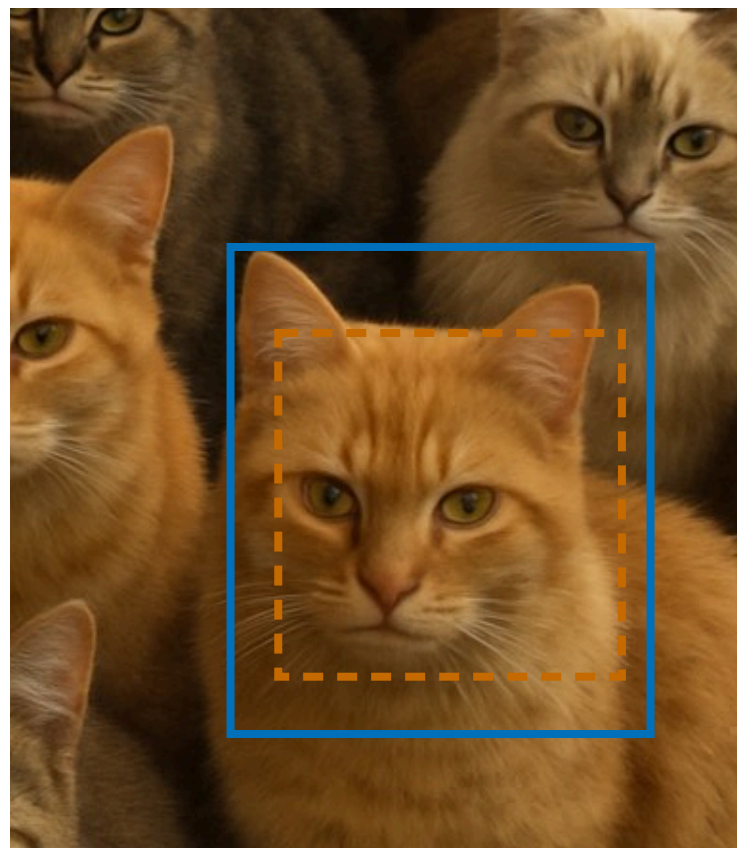
split the image into an  $S \times S$  grid (here  $8 \times 8$ )

the cell containing the object's centre is responsible for it

that cell predicts the box, a confidence, and class scores

**output tensor:  $S \times S \times (B \cdot 5 + C)$**   
**YOLOv1:  $7 \times 7 \times (2 \cdot 5 + 20) = 7 \times 7 \times 30$**

- Each cell has a few reference boxes of typical sizes and aspect ratios.
- The network predicts small offsets from the best-matching anchor.
- Small, normalized targets are much easier to learn than raw coordinates.



## anchor (prior)

the network predicts small corrections  $(t_x, t_y, t_w, t_h)$ :

$$x = x_a + t_x \cdot w_a \quad y = y_a + t_y \cdot h_a$$

$$w = w_a \cdot \exp(t_w) \quad h = h_a \cdot \exp(t_h)$$

## predicted box

small, well-scaled targets are easy to regress;  
anchor-free detectors (FCOS, CenterNet, YOLOv8)  
predict distances from a point instead

- Most candidate boxes contain no object.
- YOLO adds an objectness score; SSD and RetinaNet add a background class.
- Either way, a score threshold removes most candidates.



**every candidate box gets**

YOLO: objectness  $\in (0, 1) \times$  class probabilities

score =  $P(\text{object}) \cdot P(\text{class} \mid \text{object})$

SSD / RetinaNet: one extra "background" class

(or per-class sigmoids where background = all low)

**either way: most candidates are background  
and must be discarded by a score threshold**

# From Ten Thousand Candidates to a Few Boxes

## Part 4 · Detection

- Dense detectors score every anchor at every location of several maps.
- A score threshold keeps the confident ones.
- Several boxes still cover the same object — they must be merged.

**YOLOv3 at  $416 \times 416$ :  $(13^2 + 26^2 + 52^2) \cdot 3$  anchors = 10 647 candidate boxes per image**

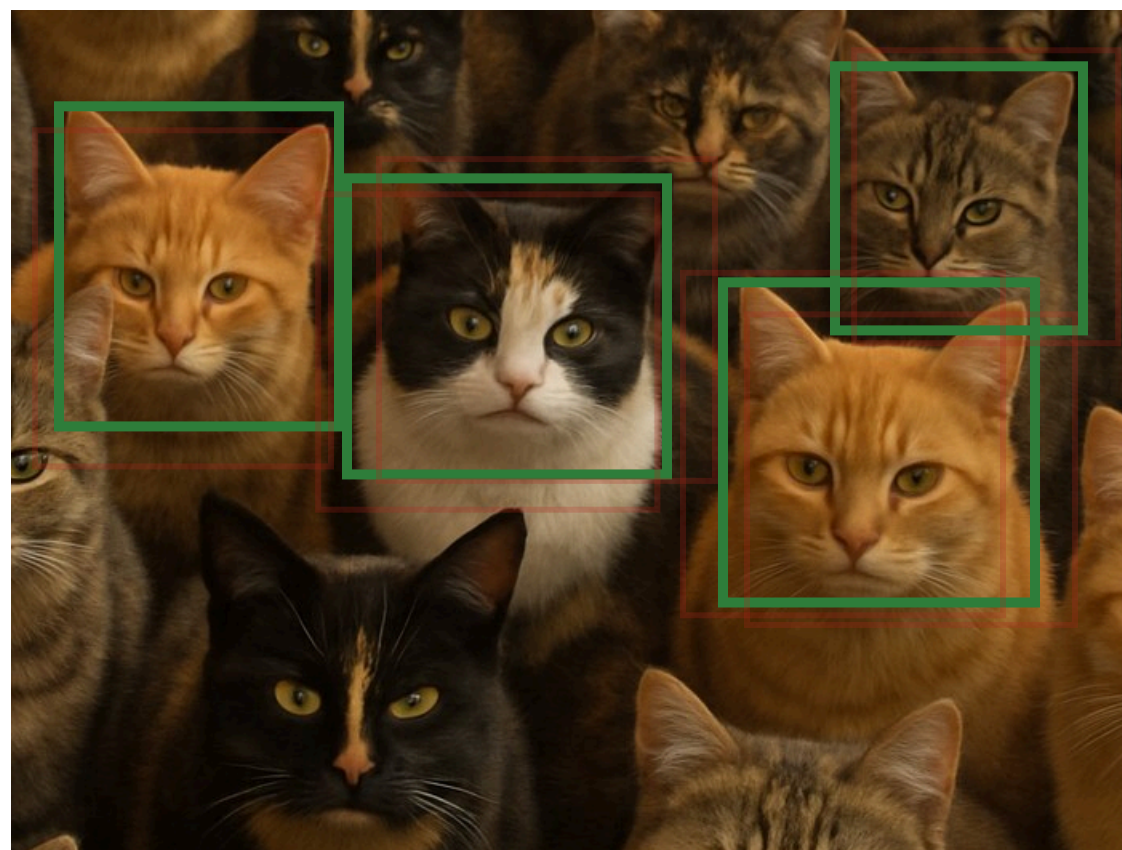
**all candidates: 10 647**

**score > 0.25: 63**

**after NMS: 11**

numbers after the first row are illustrative; width on a log scale

- Take the highest-scoring box and keep it.
- Remove all remaining boxes that overlap it with IoU above a threshold.
- Repeat until no boxes are left. Press → to run it.



## candidates sorted by score

|     |            |                        |
|-----|------------|------------------------|
| #1  | score 0.95 | keep                   |
| #2  | score 0.92 | keep                   |
| #3  | score 0.90 | keep                   |
| #4  | score 0.88 | keep                   |
| #5  | score 0.81 | suppressed (IoU > 0.5) |
| #6  | score 0.74 | suppressed (IoU > 0.5) |
| #7  | score 0.70 | suppressed (IoU > 0.5) |
| #8  | score 0.66 | suppressed (IoU > 0.5) |
| #9  | score 0.62 | suppressed (IoU > 0.5) |
| #10 | score 0.58 | suppressed (IoU > 0.5) |

done: 4 boxes kept from 10

# When NMS Removes Real Objects

Part 4 · Detection

- Neighbouring objects of the same class can overlap strongly.
- NMS cannot tell a duplicate from a neighbour — only the IoU threshold decides.
- Move the threshold and watch the second cat.



**two real cats overlap: IoU of their boxes = 0.11**

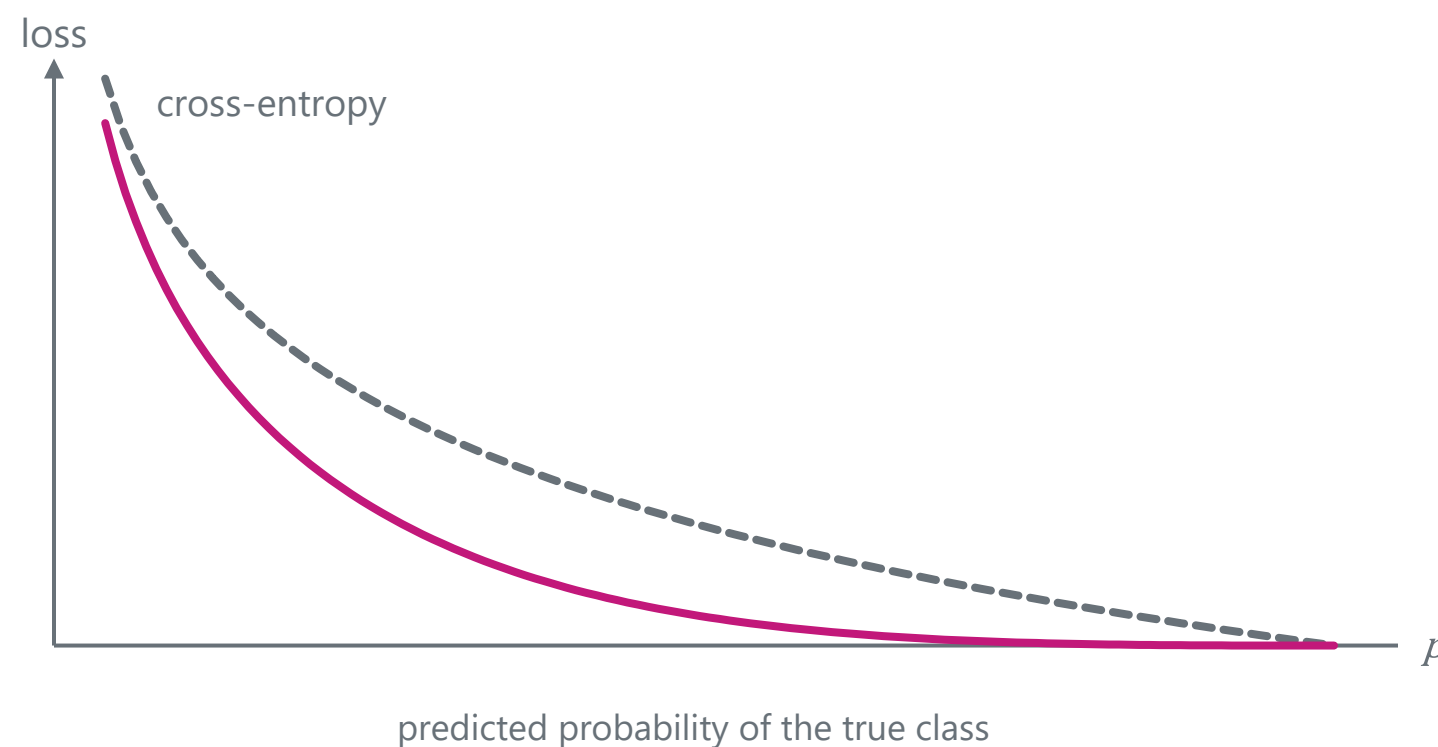
**threshold  $0.50 \geq \text{IoU} \rightarrow$  both cats kept ✓**

a low NMS threshold removes true neighbours (crowds, piles of parts);  
a high one keeps duplicates — tune it on validation data,  
or use class-aware / soft NMS

# Focal Loss: Too Many Easy Negatives

Part 4 · Detection

- A dense detector sees thousands of easy background boxes per image.
- Their small losses add up and dominate cross-entropy (lecture 1: class imbalance).
- Focal loss down-weights examples the model already gets right.



$$\text{focal loss: } -(1 - p)^{\gamma} \log p$$

easy box ( $p = 0.95$ ): weight  $2.5e-3$

hard box ( $p = 0.2$ ): weight 0.64

10 000 easy background boxes no longer  
drown the few hard, informative ones

# Measuring Detection: Precision, Recall, AP

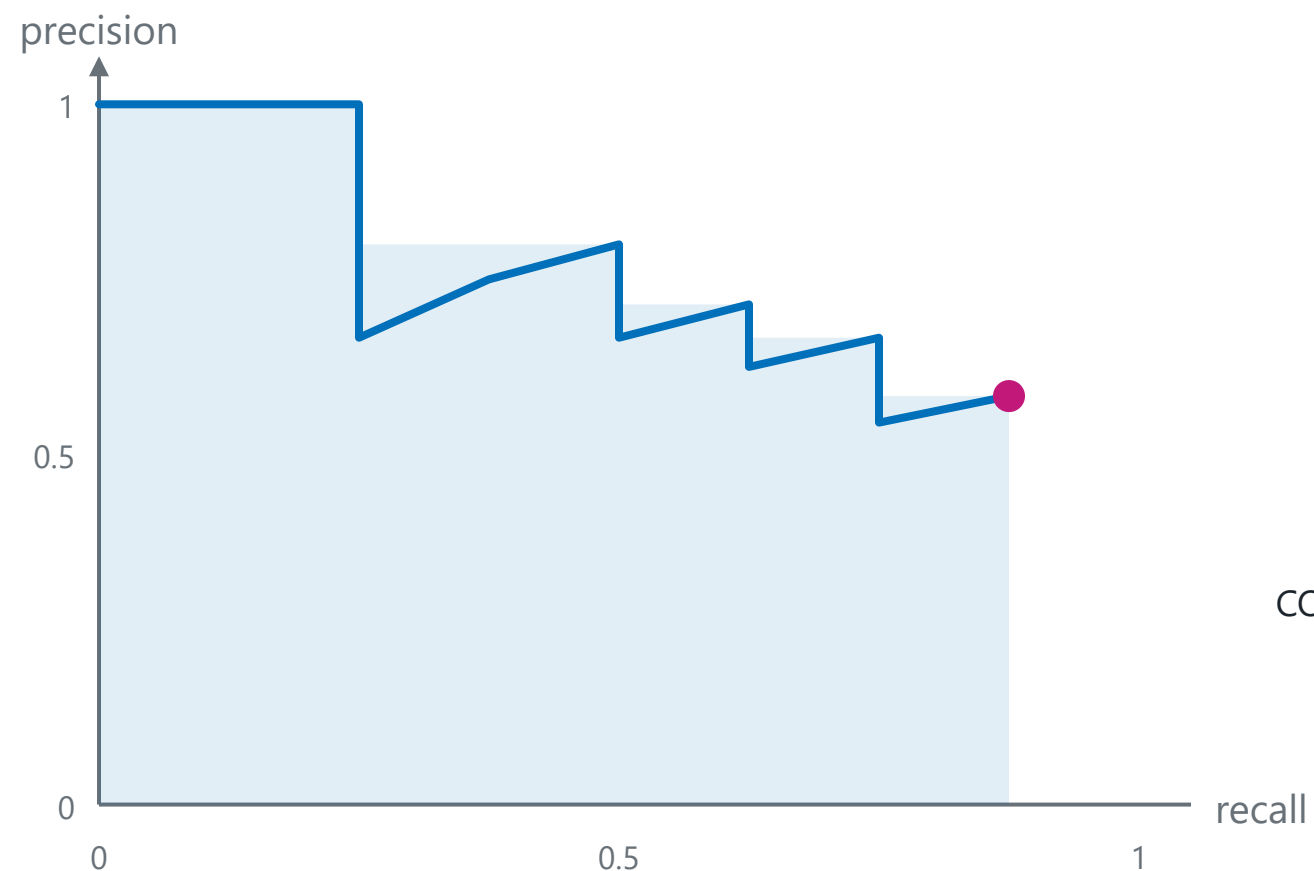
## Part 4 · Detection

- A detection is a true positive if it matches an unmatched object with  $\text{IoU} \geq 0.5$ .
- Sweeping the score threshold traces a precision–recall curve.
- Average precision (AP) summarizes it; mAP averages over classes.

### detections sorted by score

|      |    |
|------|----|
| 0.97 | TP |
| 0.93 | TP |
| 0.90 | FP |
| 0.86 | TP |
| 0.81 | TP |
| 0.77 | FP |
| 0.70 | TP |
| 0.64 | FP |
| 0.58 | TP |
| 0.51 | FP |
| 0.44 | FP |
| 0.38 | TP |

8 ground-truth objects



**AP = 0.70**

lower the threshold:  
recall ↑, precision usually ↓

AP = area under the  
(envelope of the) PR curve

mAP = mean over classes;  
COCO also averages IoU 0.5–0.95

**Your detector misses 10-pixel defects. Deeper backbone, or a higher-resolution feature map?**

Higher resolution: at stride 32 the defect is a fraction of one cell and is averaged away. Depth adds semantics, not detail.

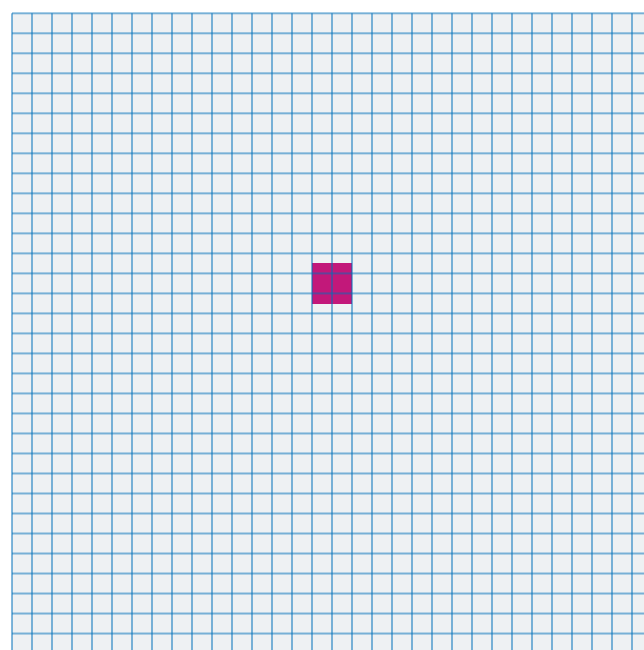


# Why Small Objects Get Lost

Part 5 · Multi-scale necks

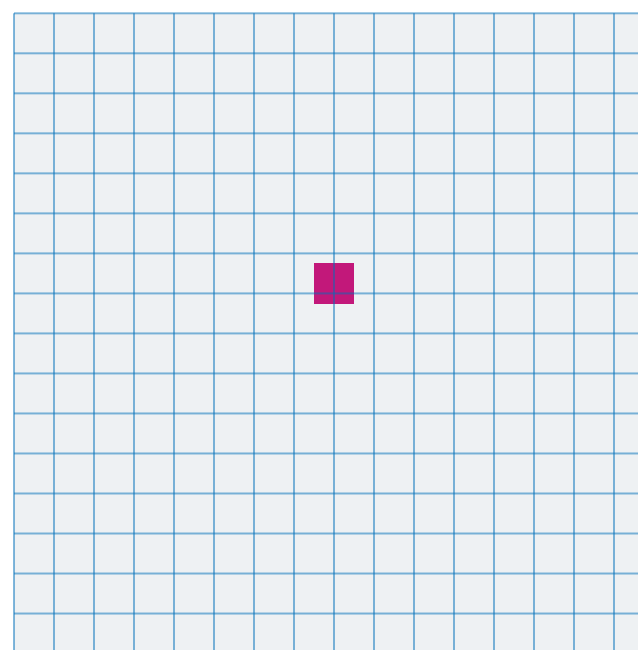
- On a stride-32 map, one cell summarizes  $32 \times 32$  input pixels.
- An object smaller than a cell is mixed with its background.
- Small objects need fine maps; large objects need wide context.

**stride 8**



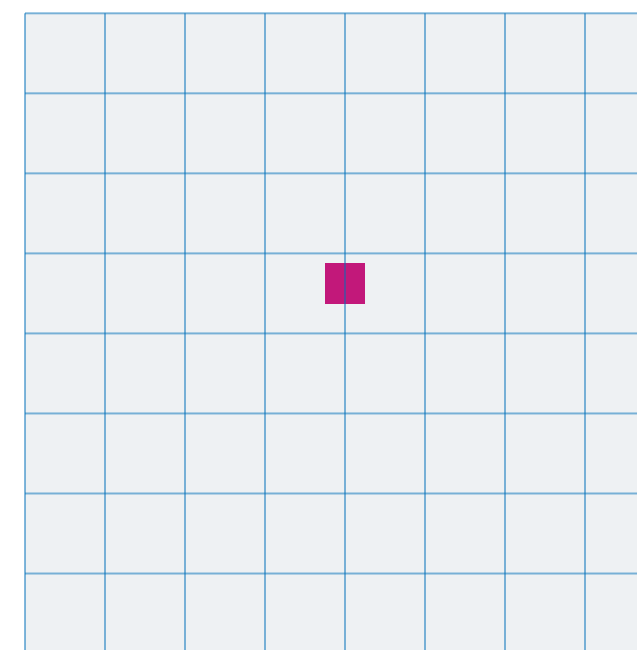
the 16-px defect fills  $2 \times 2$  cells

**stride 16**



fills exactly one cell

**stride 32**

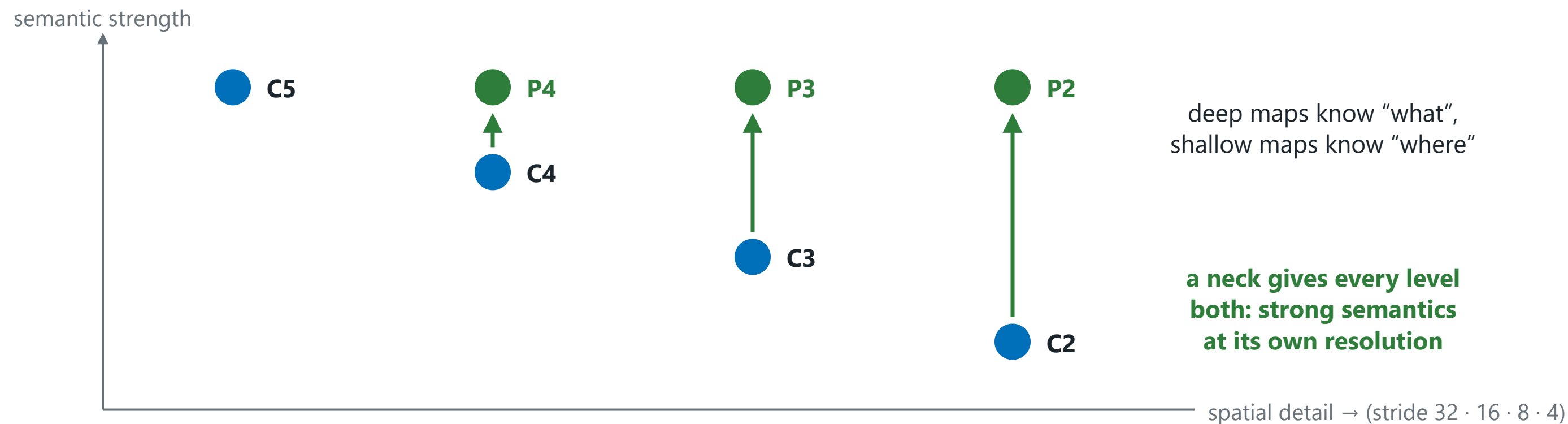


covers  $\frac{1}{4}$  of a cell: diluted by background

# Detail versus Meaning

Part 5 · Multi-scale necks

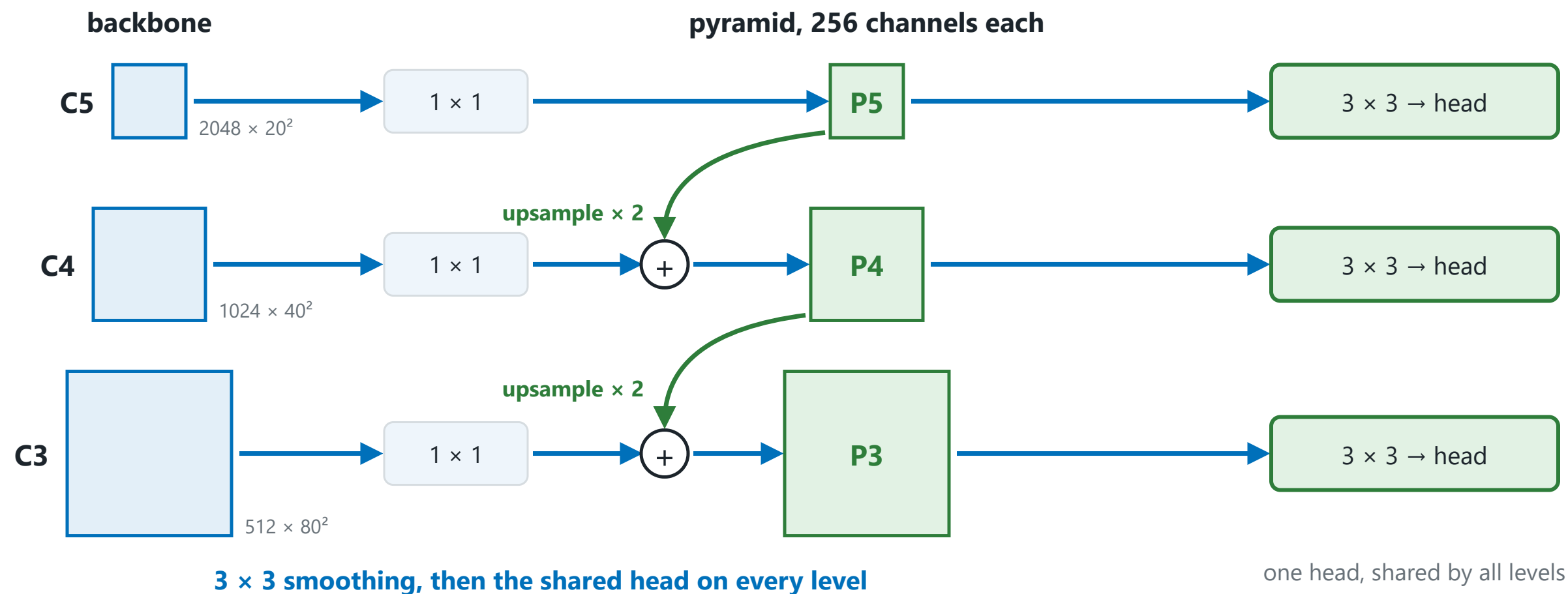
- Shallow maps are sharp but semantically weak.
- Deep maps are semantically strong but coarse.
- A neck brings the meaning of deep maps down to the fine levels.



# Feature Pyramid Network, Step by Step

Part 5 · Multi-scale necks

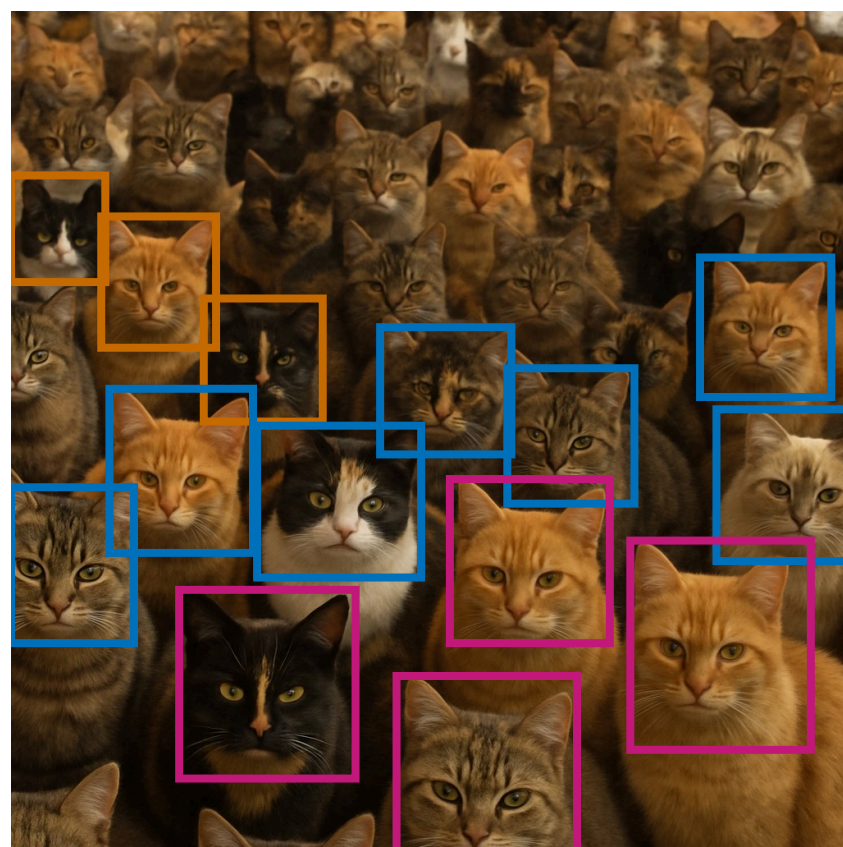
- Start from the deepest map; upsample it and add the next finer backbone map.
- $1 \times 1$  lateral convolutions make the channel counts equal (so they can be added).
- The result: a pyramid where every level is both fine and semantic.



# Which Level Detects Which Object

Part 5 · Multi-scale necks

- Small objects are assigned to P3, large ones to P5.
- Each level then only handles a limited range of sizes.
- The head is shared: it learns one rule applied at every scale.



 **P3 (stride 8): small objects**

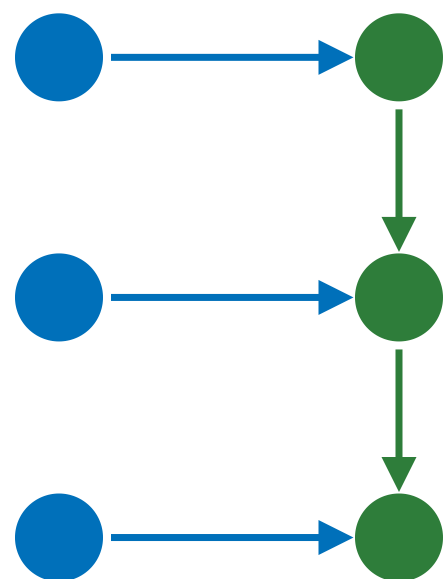
 **P4 (stride 16): medium objects**

 **P5 (stride 32): large objects**

each object is predicted on the level whose cells match its size:  
 $\text{level} = 4 + \log_2(\sqrt{w \cdot h} / 224)$  (FPN paper, rounded and clipped)

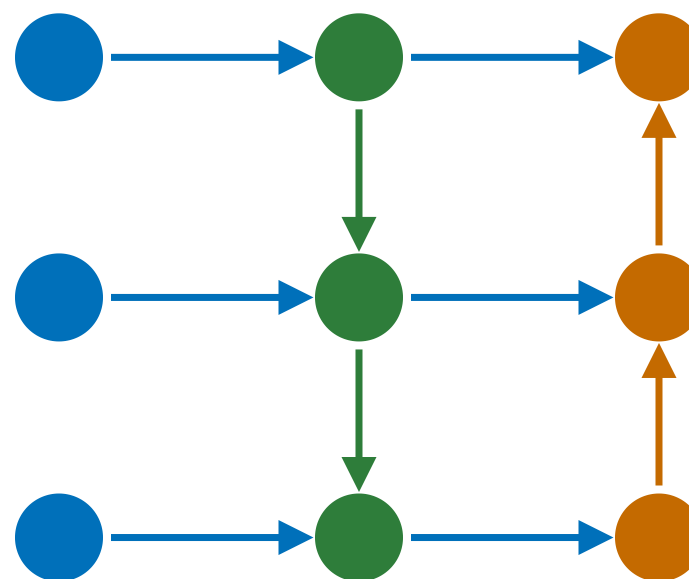
- FPN sends information only from deep to fine levels.
- PANet adds a bottom-up path so fine details reach the deep levels too.
- BiFPN repeats weighted two-way fusion; more fusion is not free — measure it.

**FPN**



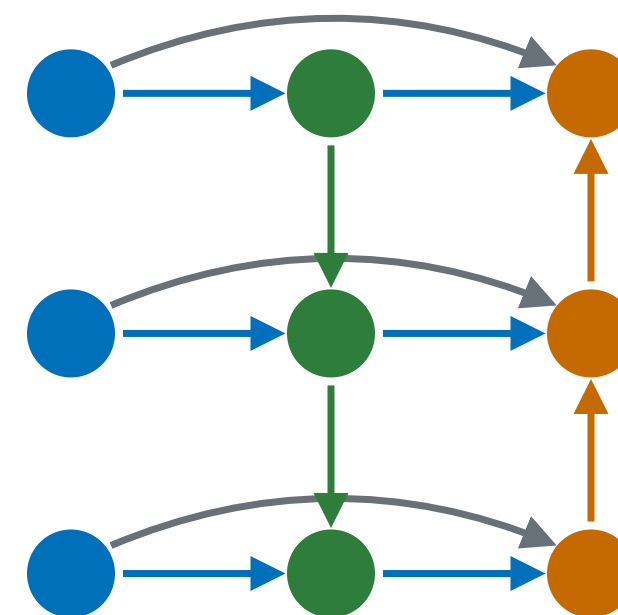
top-down only

**PANet**



+ a bottom-up path

**BiFPN**



weighted, repeated  $\times N$

$$w_1 P + w_2 Q$$

# Check question

Choose a level

**All objects in your data are between 8 and 20 pixels.  
What would you change in a standard FPN detector?**

Add or emphasize a finer level (P2, stride 4), drop P5–P7, and check anchor sizes; also consider a higher input resolution.



# Kinds of Segmentation

## Part 6 · Segmentation

- Semantic: every pixel gets a class; objects of one class merge.
- Instance: every object gets its own mask.
- Panoptic: both at once — countable objects plus background regions.

input



semantic



one label per pixel:  
all apples are "apple"

instance



each apple is a separate  
object with its own mask

- Thin cracks on steel: tiny, elongated, extremely imbalanced.
- Apples among leaves: occlusion, varying size, similar colours.
- The output must be as sharp as the annotation.

**apples in a tree: input, annotation, network output**



**input**



**target**



**output**

# The Segmentation Output

## Part 6 · Segmentation

- The network outputs  $C$  logit maps, one per class.
- Softmax across channels at each pixel gives class probabilities.
- Argmax per pixel gives the label map.

**logits: background**

|     |      |      |      |      |      |
|-----|------|------|------|------|------|
| 1.5 | 1.6  | 2.2  | 1.7  | 0    | -0.4 |
| 2.1 | -0.9 | -0.4 | -0.5 | 0.1  | 0.2  |
| 1.5 | -1   | -0.6 | -0.3 | -0.6 | -0.4 |
| 1.9 | -0.3 | -1   | -0.2 | -0.7 | 1.6  |
| 2.2 | 1.8  | -0.4 | -0.3 | 1.6  | 1.9  |
| 1.4 | 1.4  | 1.5  | 2.1  | 1.4  | 2.1  |

**logits: apple**

|      |      |      |      |      |      |
|------|------|------|------|------|------|
| -0.8 | -0.3 | -0.6 | 0    | -0.4 | -0.1 |
| -0.8 | 3.4  | 2.9  | 2.7  | -0.5 | -1   |
| -1   | 3.5  | 2.8  | 2.8  | 2.9  | -0.9 |
| -0.2 | 3.6  | 3.5  | 2.8  | 2.7  | -1   |
| -0.5 | -0.1 | 3.2  | 2.7  | -0.3 | -0.1 |
| -0.9 | -0.3 | -0.8 | -0.1 | -0.8 | -0.3 |

**logits: leaf**

|      |      |      |      |      |      |
|------|------|------|------|------|------|
| 0.1  | -0.5 | -0.4 | -0.1 | 2.9  | 2.9  |
| -0.6 | -0.2 | -0.5 | -0.1 | 2.5  | 2.6  |
| -0.3 | 0.1  | -0.6 | -0.1 | 0    | 3.1  |
| -0.2 | -0.7 | 0.1  | -0.3 | -0.6 | -0.4 |
| -0.2 | -0.6 | -0.2 | -0.5 | -0.3 | 0    |
| -0.2 | -0.5 | -0.6 | 0.1  | 0    | -0.7 |

**argmax per pixel**

|            |            |            |            |            |            |
|------------|------------|------------|------------|------------|------------|
| background | background | background | background | apple      | apple      |
| background | apple      | apple      | apple      | apple      | apple      |
| background | apple      | apple      | apple      | apple      | apple      |
| background | apple      | apple      | apple      | apple      | background |
| background | background | apple      | apple      | background | background |
| background | background | background | background | background | background |

one pixel: logits (-0.6, 2.8, -0.6) → softmax → (0.03, 0.94, 0.03)

**output tensor  $C \times H \times W \rightarrow$  softmax over  $C$  at every pixel  $\rightarrow$  class map  $H \times W$**

# Why the Backbone Alone Is Not Enough

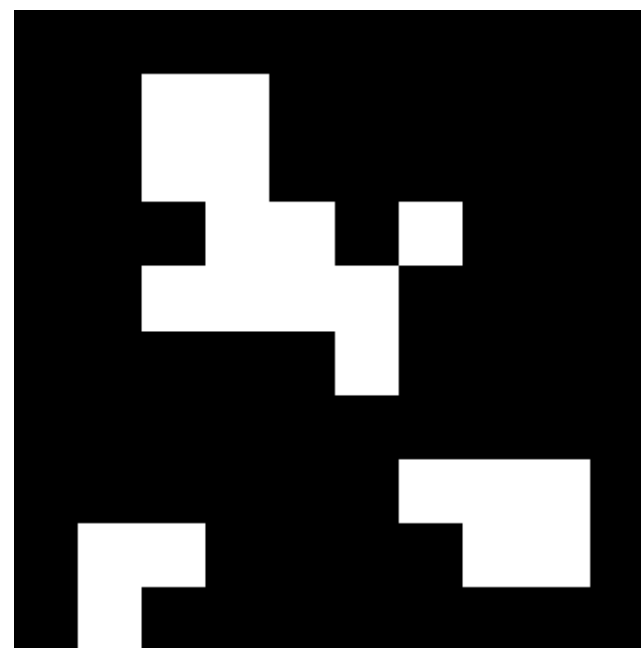
Part 6 · Segmentation

- The last backbone map has stride 32: a 331-pixel image becomes  $10 \times 10$  cells.
- Upsampled back, the mask is blocky and small objects vanish.
- We need to recover resolution — and the detail lost on the way down.

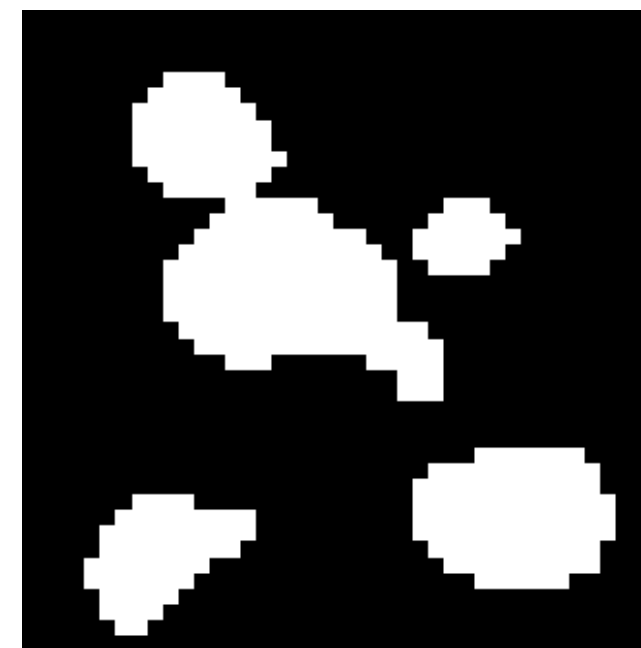
annotation



predicted at stride 32 ( $10 \times 10$ ), upsampled



predicted at stride 8 ( $41 \times 41$ ), upsampled



the backbone alone ends at stride 32: far too coarse for object boundaries

- Nearest neighbour and bilinear interpolation have no parameters.
- Transposed convolution learns the upsampling but can create checkerboard patterns.
- Bilinear resize followed by a convolution is a safe default.

**input 2 × 2**

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

**nearest neighbour**

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

blocky, no parameters

**bilinear**

|      |      |      |      |
|------|------|------|------|
| 1    | 1.5  | 2.5  | 3    |
| 1.25 | 1.75 | 2.75 | 3.25 |
| 1.75 | 2.25 | 3.25 | 3.75 |
| 2    | 2.5  | 3.5  | 4    |

smooth, no parameters

**transposed conv (3 × 3, stride 2)**

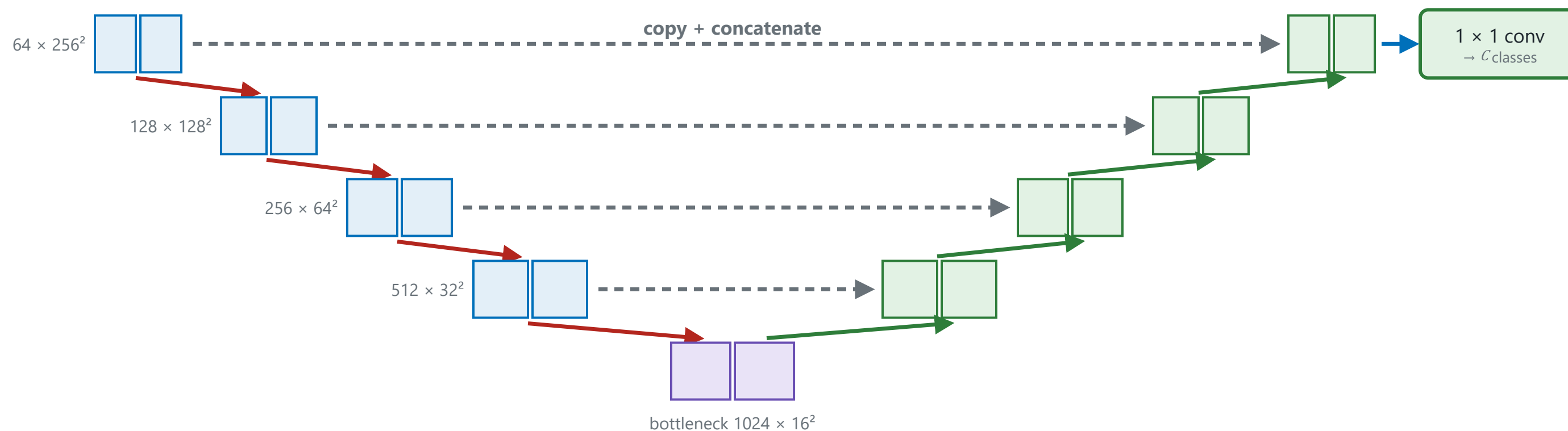
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 1 | 2 | 1 | 1 |
| 1 | 1 | 2 | 1 | 2 | 1 | 1 |
| 2 | 2 | 4 | 2 | 4 | 2 | 2 |
| 1 | 1 | 2 | 1 | 2 | 1 | 1 |
| 2 | 2 | 4 | 2 | 4 | 2 | 2 |
| 1 | 1 | 2 | 1 | 2 | 1 | 1 |
| 1 | 1 | 2 | 1 | 2 | 1 | 1 |

learned, but kernel overlaps differ  
(1, 2, 4 contributions) → checkerboard  
artifacts; resize + conv avoids them

# Encoder–Decoder with Skip Connections

Part 6 · Segmentation

- The encoder is a backbone; the decoder upsamples step by step.
- Skip connections concatenate encoder maps of the same size into the decoder.
- This is U-Net (2015) — originally for medical images, now a general pattern.



1 × 1 conv gives C logits per pixel at input resolution

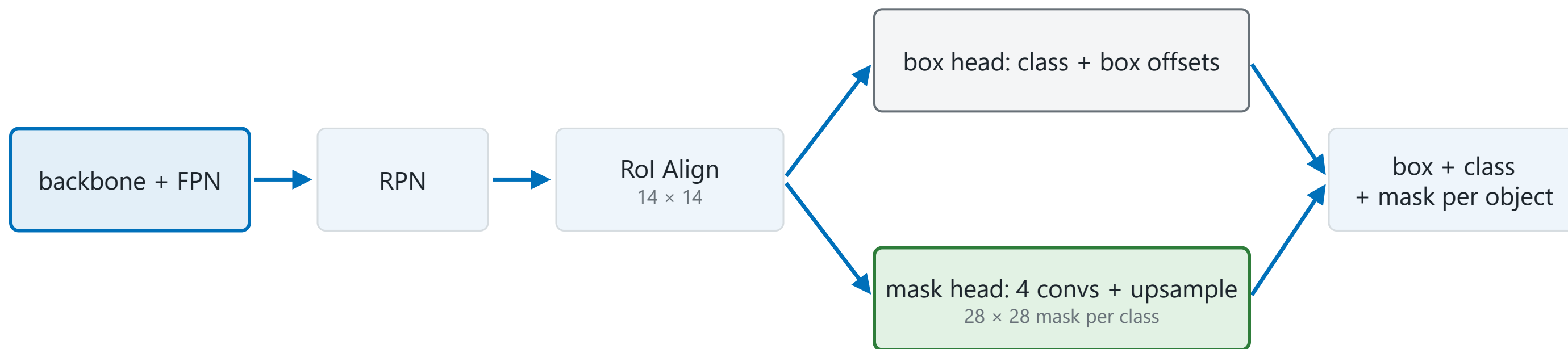
# U-Net Shapes

Part 6 · Segmentation

| level      | encoder output              | decoder input (after concat)            | decoder output                                                    |
|------------|-----------------------------|-----------------------------------------|-------------------------------------------------------------------|
| 1          | $64 \times 256 \times 256$  | $64 + 64 = 128 \times 256 \times 256$   | $64 \times 256 \times 256$                                        |
| 2          | $128 \times 128 \times 128$ | $128 + 128 = 256 \times 128 \times 128$ | $128 \times 128 \times 128$                                       |
| 3          | $256 \times 64 \times 64$   | $256 + 256 = 512 \times 64 \times 64$   | $256 \times 64 \times 64$                                         |
| 4          | $512 \times 32 \times 32$   | $512 + 512 = 1024 \times 32 \times 32$  | $512 \times 32 \times 32$                                         |
| bottleneck | $1024 \times 16 \times 16$  | –                                       | upsampled to $512 \times 32 \times 32$                            |
| output     |                             |                                         | $1 \times 1$ conv $\rightarrow \mathcal{C} \times 256 \times 256$ |

Read from the bottom up: each decoder level upsamples, concatenates the encoder map of the same size, then convolves. Padded convolutions keep the sizes aligned.

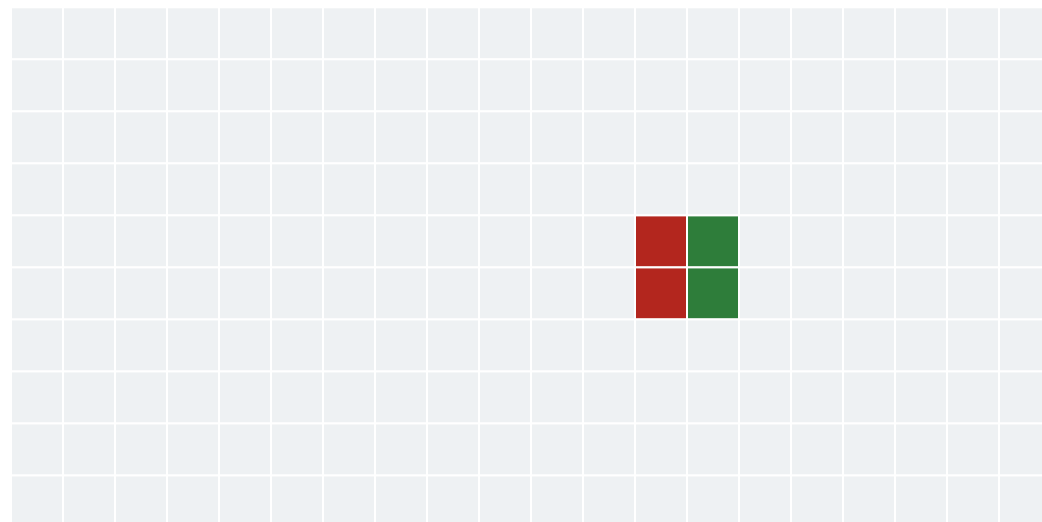
- Detect each object first, then segment inside its box.
- A small mask branch is added next to the box head of Faster R-CNN.
- RoI Align (Part 4) is what makes these masks accurate.



Mask R-CNN = Faster R-CNN + a small mask branch; the mask is pasted back into the box

- Pixel accuracy is misleading when the object is small (lecture 1: imbalance).
- IoU and Dice measure the overlap of the predicted and true regions.
- Report them per class; for thin structures also check boundaries.

**prediction C: misses half**



green = hit, red = missed, orange = false alarm

pixel accuracy **99 %**

IoU **0.50**

Dice **0.67**

**For the steel billet cracks: semantic or instance segmentation, which decoder, and which metric?**

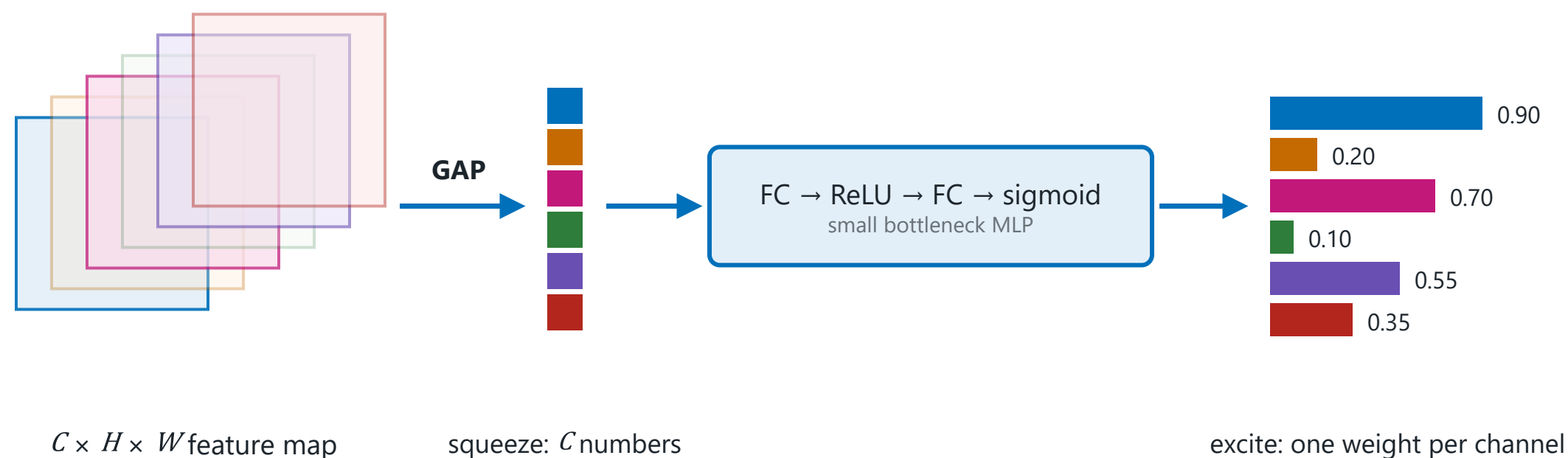
Semantic (one crack class), U-Net-style decoder with skips to keep 1-pixel detail, Dice or IoU of the crack class (maybe with a boundary tolerance) — never pixel accuracy.



# Channel Attention: Squeeze and Excitation

Part 7 · Attention and design

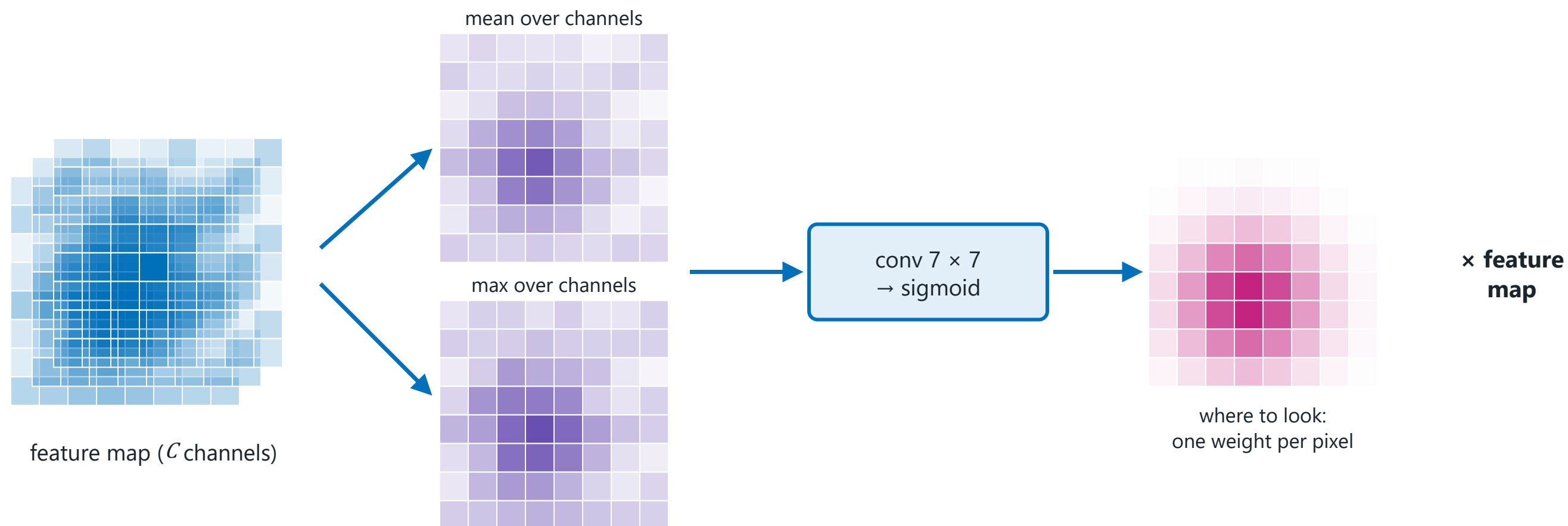
- Not every channel is useful for every image.
- SE summarizes each channel (GAP), computes a weight per channel, and rescales.
- Cheap to add to any block; typically +1 point on ImageNet.



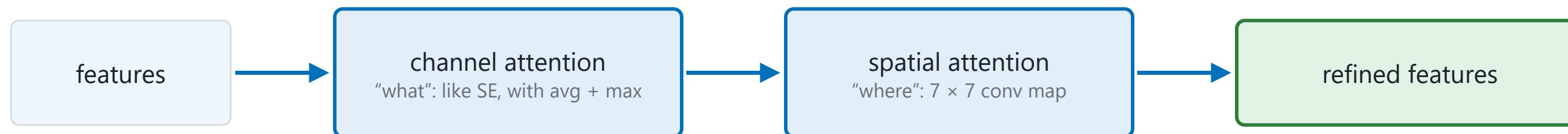
**scale: multiply each channel by its weight**

the network learns  
"which" features matter  
for this image

- The same idea for positions: which parts of the map matter?
- Pool over channels, convolve, and squash into a 0–1 weight per pixel.
- The feature map is multiplied by this mask.



- Channel attention first, spatial attention second.
- A drop-in module after convolutional blocks.
- Attention reweights features — it cannot recover detail that was never resolved.



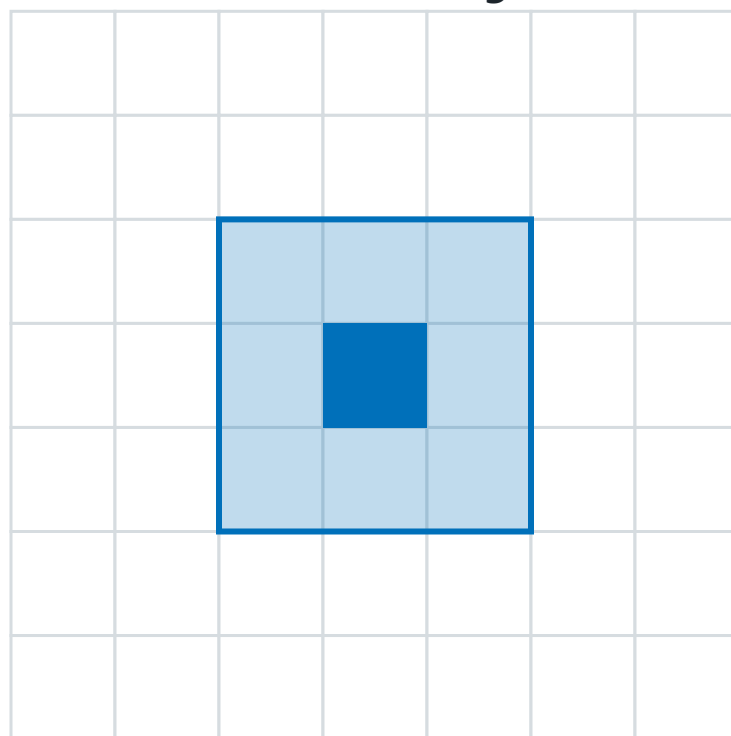
CBAM (2018): a few thousand extra parameters, inserted after any conv block

# Self-Attention versus Convolution

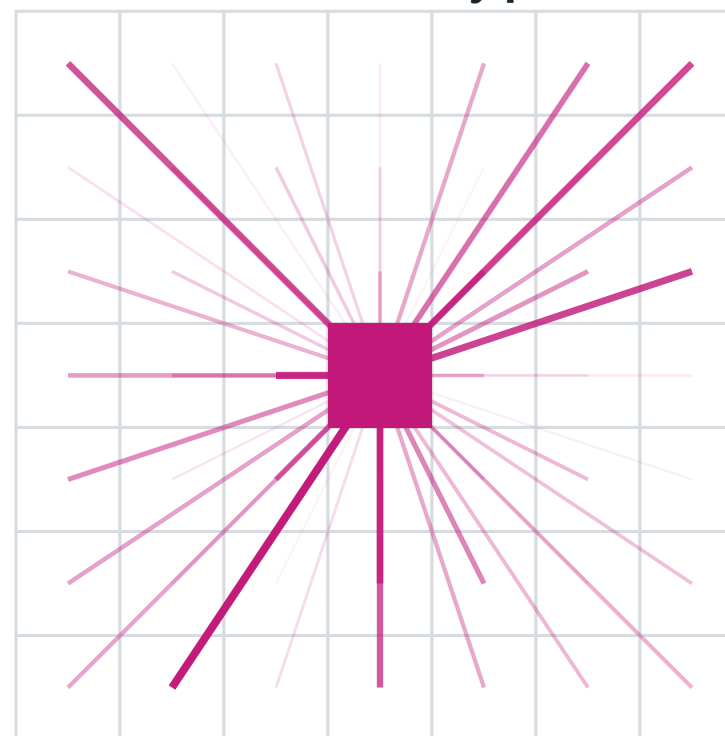
Part 7 · Attention and design

- A convolution only sees a fixed local neighbourhood.
- Self-attention lets every position look at every other, weighted by content.
- Global context at a quadratic cost; modern models mix both.

**convolution: a  $3 \times 3$  neighbourhood**



**self-attention: every position**



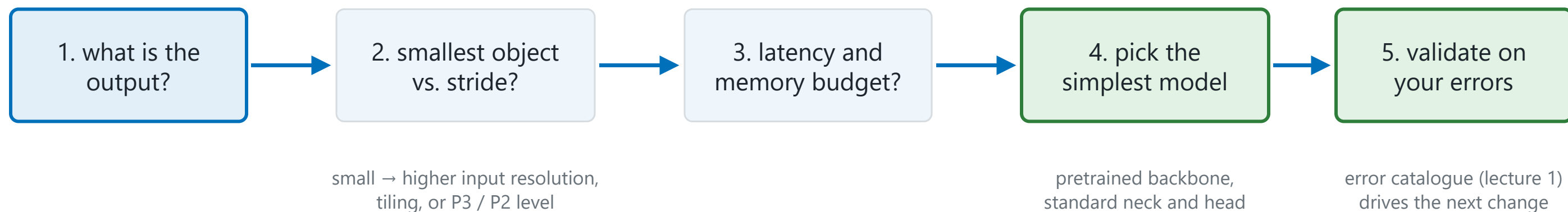
weights depend on  
the content of both  
positions

cost grows with  
 $(H \cdot W)^2$  — why  
transformers use patches

- Decide in this order; each answer narrows the next choice.
- Start from the simplest standard model that can produce the output.
- Change one block at a time, driven by the errors you observe.

label → classification head  
boxes → detector  
mask → U-Net / Mask R-CNN  
similarity → embedding

edge device → MobileNet,  
EfficientNet; server → ResNet,  
larger detectors



## Small dataset

- pretrained backbone, simple head
- freeze first, fine-tune later
- split by part / batch / day
- better labels beat a fancier neck

## Small objects

- higher input resolution or tiling
- use a fine level (P3 or P2)
- avoid early aggressive striding
- report recall by object size

## Edge deployment

- MobileNet / EfficientNet backbone
- static shapes, supported operators
- measure end-to-end latency on the device
- check quantized accuracy per class

# Ablations: Which Change Helped?

Part 7 · Attention and design

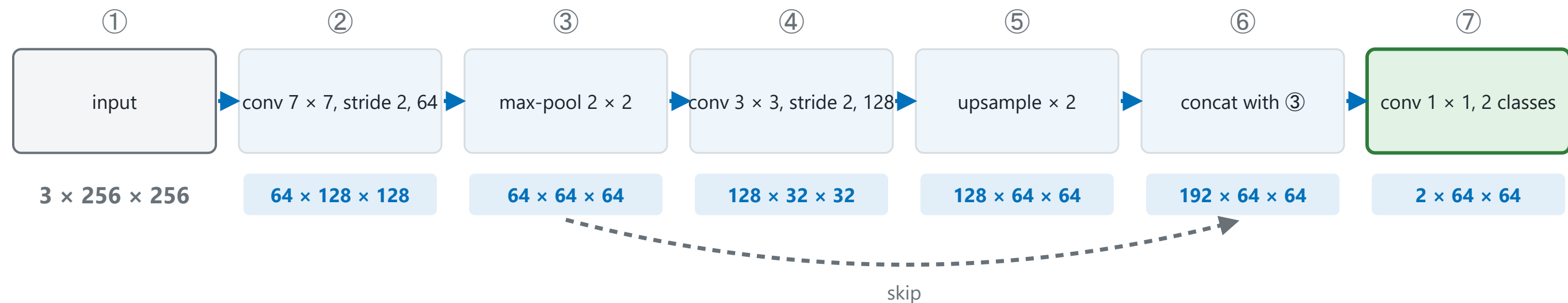
| variant                    | mAP (all) | recall, objects < 16 px | latency |
|----------------------------|-----------|-------------------------|---------|
| baseline: ResNet-50, P3–P5 | 0.61      | 0.42                    | 18 ms   |
| + input 640 → 960          | 0.64      | 0.58                    | 39 ms   |
| + P2 level instead         | 0.63      | 0.55                    | 24 ms   |
| + CBAM in the neck         | 0.62      | 0.44                    | 20 ms   |
| ResNet-101 backbone        | 0.62      | 0.43                    | 29 ms   |

Illustrative numbers. Change one thing per row, report quality *and* cost, and look at the metric that matches your problem — here, small-object recall.

# Reading a Network Diagram

Part 7 · Attention and design

- Annotate every node with  $C \times H \times W$  before trusting a diagram.
- For every merge, decide whether it adds or concatenates.
- Work along the diagram; press  $\rightarrow$  to check each shape.



for every node write  $C \times H \times W$ ; for every merge decide: add (shapes equal) or concat (channels sum)

**the output is 4× smaller than the input: is that acceptable for the smallest object?**

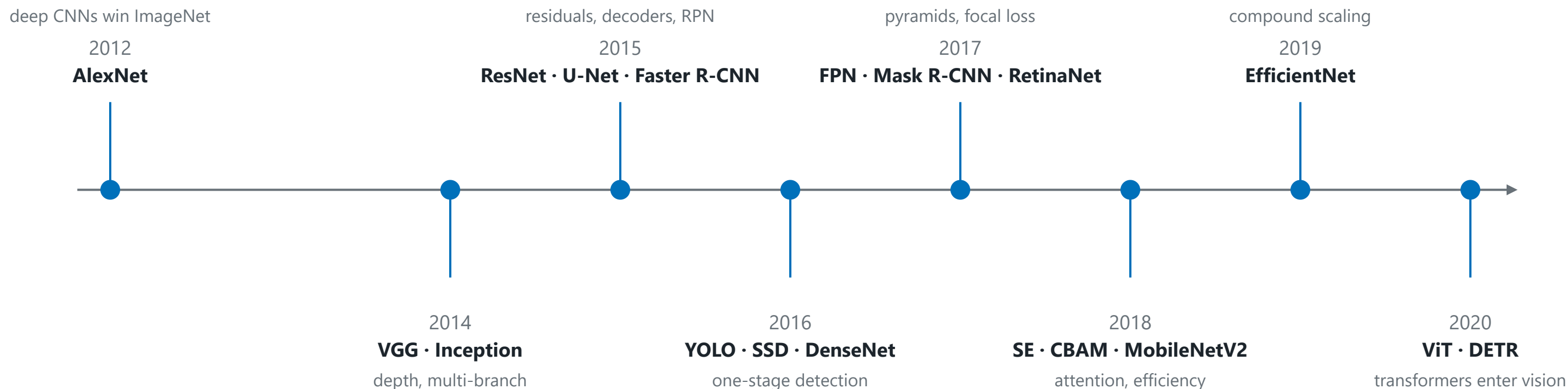
- Downsampling too early: the defect is gone before any layer can see it.
- A head that does not match the annotation (boxes for amorphous regions, masks for counts).
- Copying the biggest model from a paper instead of the simplest that fits.
- Benchmarking the network alone, ignoring preprocessing, NMS, and data transfer.
- Judging a change by one aggregate number instead of the errors that matter.

**A camera must find hairline cracks on a moving strip at 30 FPS. Sketch the backbone, neck, head, input size, and metric.**

Tiles at full resolution, efficient backbone, U-Net-style decoder or P2-level head, Dice / recall on cracks, latency measured end to end on the target hardware.



- Most building blocks used today appeared between 2012 and 2018.
- Later work refines efficiency, scaling, and training — and adds transformers.
- Knowing the blocks lets you read new architectures quickly.



- Backbone, neck, and head solve different parts of the problem.
- Spatial scale decides most localization problems.
- Choose by output, object size, budget, and observed errors — not by popularity.

## **BACKBONE**

depth via residuals,  
efficiency via separable convs

## **NECK**

multi-scale fusion:  
FPN, PANet, BiFPN

## **HEAD**

shapes the output:  
label, boxes, mask, embedding

## **CHOICE**

driven by output, scale,  
budget, and errors

- So far, every output lived in a single image plane.
- Next, the course turns to geometry: camera models, multi-view geometry, and conventional visual SLAM.
- Learned 3D reconstruction comes later and builds on both: CNN features and SLAM poses.

