Detection and segmentation

COCO

LECTURE 14 | GÉRON CH. 12

Applications of Machine Learning

BSc Mathematics of Artificial Intelligence · Third year

Where we are

Three lectures of vision, and every one of them answers the same question: *what is in this image?*

- Lecture 12 — a convolutional network, built and trained from scratch, gives one label per image
- Lecture 13 — a backbone pretrained on someone else's labels gives a much better one, cheaply: probe it frozen first, then let the last block move at a smaller rate, with batch-norm statistics left alone and augmentation applied to the training loader only

> **WHAT NEITHER CAN EXPRESS**
>
> **Where** the object is, and **how many** there are. A photograph with three cyclists in it has one label and three answers, and nothing in the architecture can say so.

Today the output stops being a label and becomes a set of boxes — which means the first thing we need is a new notion of what a *correct answer* is. That is the derivation.

Today

1. The task, the corpus, and what a detector returns
2. **The mathematics** — IoU, where it stops giving a gradient, and mAP as a mean of a mean
3. **The method** — running a detector, choosing a threshold, checking the number, non-maximum suppression
4. Tracking, and segmentation
5. **Design exercise** — a parking-analytics company's occupancy product from aerial imagery, in nine steps, with the pitfalls and how to address them

Open Lecture 14 in Colab

The stakeholder

> **RETAIL SHELF AUDIT**
>
> “We photograph shelves. We need to know *how many* of each product are on each shelf, and *where* each one is, so that a picking route can be planned. A list of the categories present is no use to us.”

- Two questions, not one: **how many** and **where**
- The second question is what changes the architecture

Naming and locating are different problems

	Classification	Detection
Question	what is in this picture	what is where, and how many
Output	one vector, length K	three arrays, length N
N is	✓ **fixed before you look**	✗ **decided by the image and by two cut-offs**
Wanted from pooling	✓ **invariance**	✗ **equivariance**

The last row is Lecture 12 with a consequence. A network that has learned to *not care* where the object is cannot tell you where it is.

What a detector returns, next to what a classifier returns

Classification

one image
3 × H × W
the input, either way

CNN backbone
Lectures 12–13
frozen, pretrained

Output: one vector of length K
“this image is a cat, 0.93”
Shape is fixed before you look at the image.
One label per image; nothing says where.

Detection

Region proposals
a few hundred candidate
boxes, scored and refined

Suppression
near-duplicates removed
by overlap — today

Output: three arrays, of a length nobody chose
boxes (N × 4) · labels (N) · scores (N)
N depends on the image, on a score cut-off, and on an overlap cut-off.
Two numbers you did not choose are already in your answer.

Two numbers you did not choose are already inside the answer on the right: a score cut-off and an overlap cut-off.

A box, precisely

Four numbers. Which four is not agreed on, and that is the first bug.

Convention	Meaning	Used by
`[x1, y1, x2, y2]`	two opposite corners	torchvision, and everything today
`[x, y, w, h]`	top-left corner, then size	the COCO annotation file
`[cx, cy, w, h]`	centre, then size	YOLO-family papers

FAILURE CONDITION — TWO BOX FORMATS IN ONE NOTEBOOK

COCO stores `[x, y, w, h]` and torchvision expects corners. Read `[100, 50, 20, 30]` as corners and every box wider than its own left edge gets $x_2 < x_1$ — almost all of them, with no error. Convert once, at the edge of the program, and assert $x_2 \geq x_1$ and $y_2 \geq y_1$; nothing else will catch it.

Our corpus is 128 images. Not COCO

128

images, out of **5,000** in `val2017`

- The 128 numerically lowest image ids — a rule, not a selection
- About 20 MB of JPEG, plus the official annotation file
- Full COCO with the training split is roughly 20 GB and is not downloaded anywhere in this course

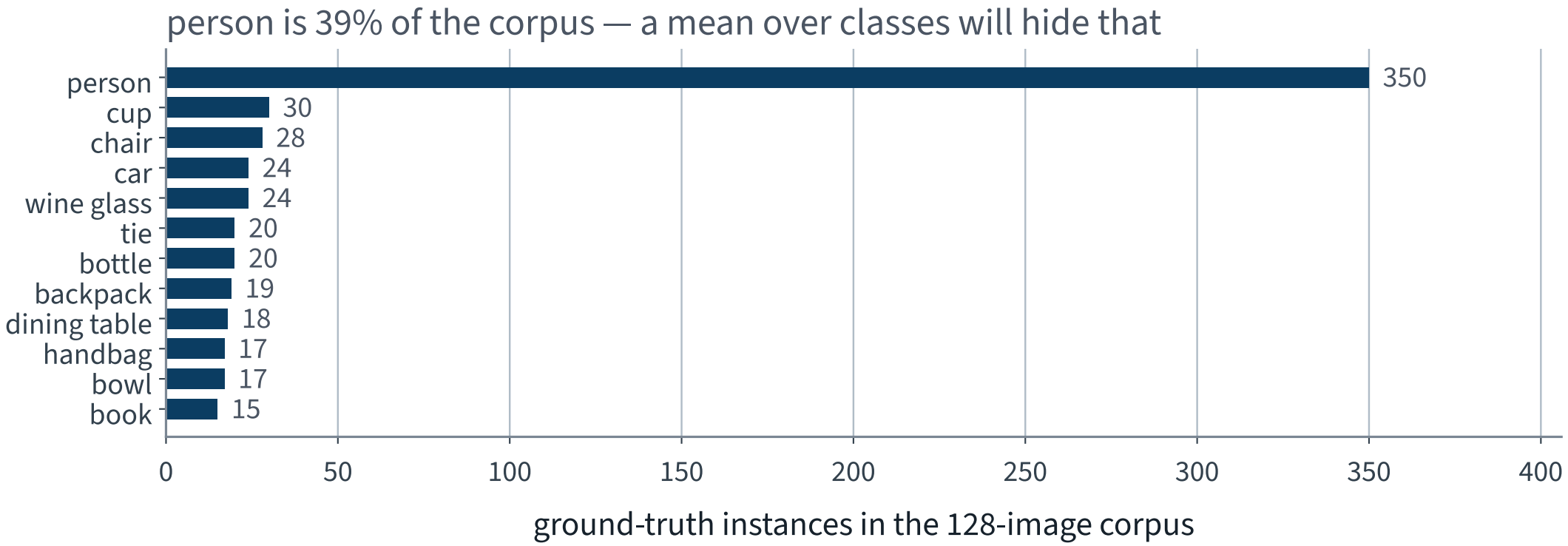
✗ Every number in this lecture is measured on 128 images. When we say a score, we say the sample size beside it.

The corpus, counted

Quantity	Value
Images	128
Annotated objects, crowd regions excluded	898
Crowd regions, dropped	8
Distinct categories appearing	73 of 80
Objects per image — mean	7.02
Objects per image — median	5
Objects per image — range	1 to 29

Mean 7.02 against median 5: the distribution has a long right tail, and one image alone carries 29 objects.

Twelve categories carry most of it



39.0% of every annotated object in this corpus is a person. Remember that when we start averaging over categories.

Intersection over union

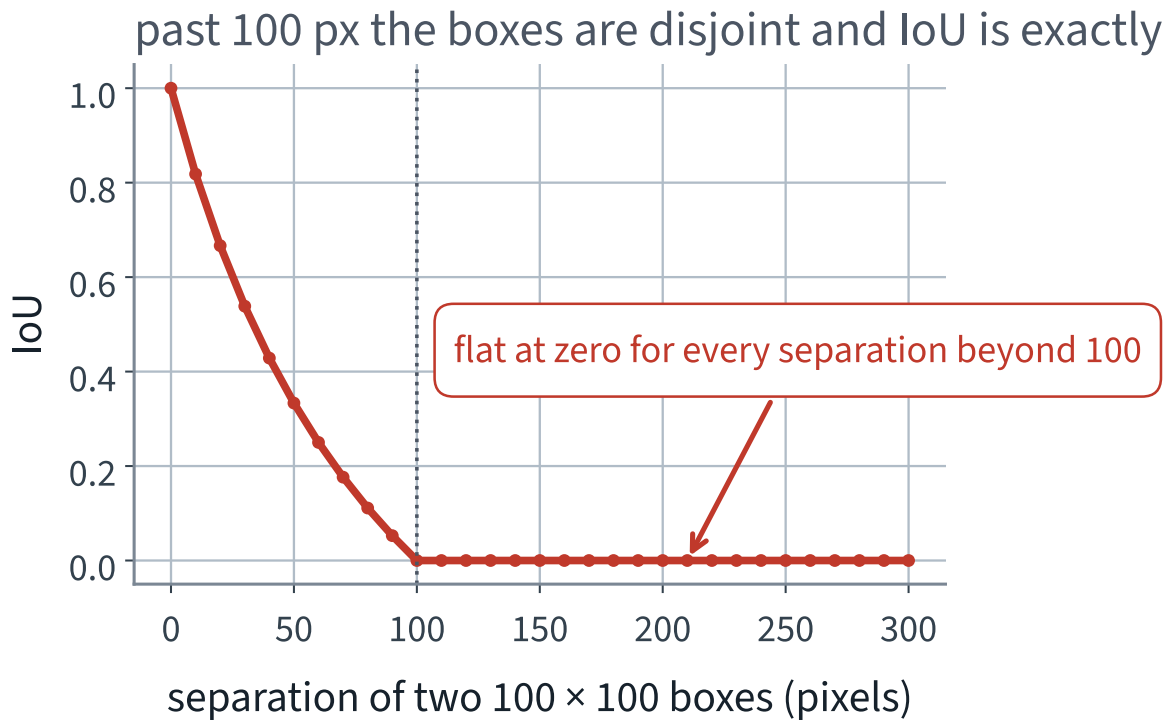
$$\mathrm{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

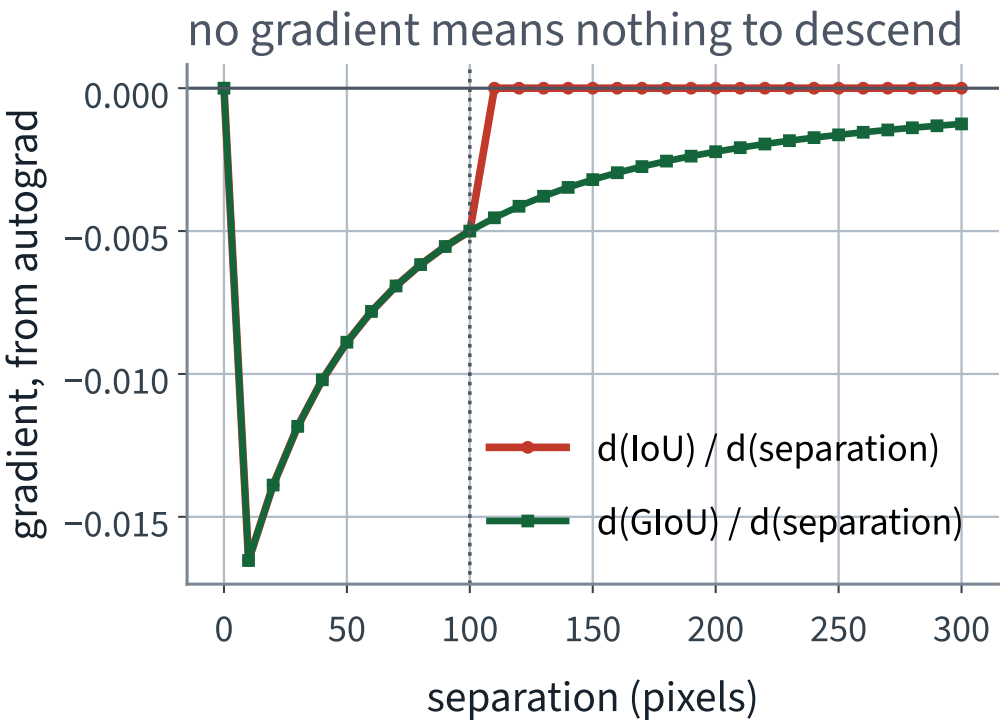
ALSO CALLED THE JACCARD INDEX

- The second form is the one you compute: the union is never built
- Subtracting the intersection is not an optimisation — add the two areas and you have counted the overlap twice
- Range $[0, 1]$, and it is exactly 1 only when $A = B$

Past 100 pixels there is nothing to descend

Use $1 - \mathrm{IoU}$ as a loss. Fix one 100×100 box, slide a second one right by d pixels, and ask autograd for the gradient.





Once the boxes are disjoint the intersection is clamped at zero, so IoU is *constant*: gradient exactly 0, not merely small.

The repair: put the empty space back in

Let C be the smallest axis-aligned box containing both A and B.

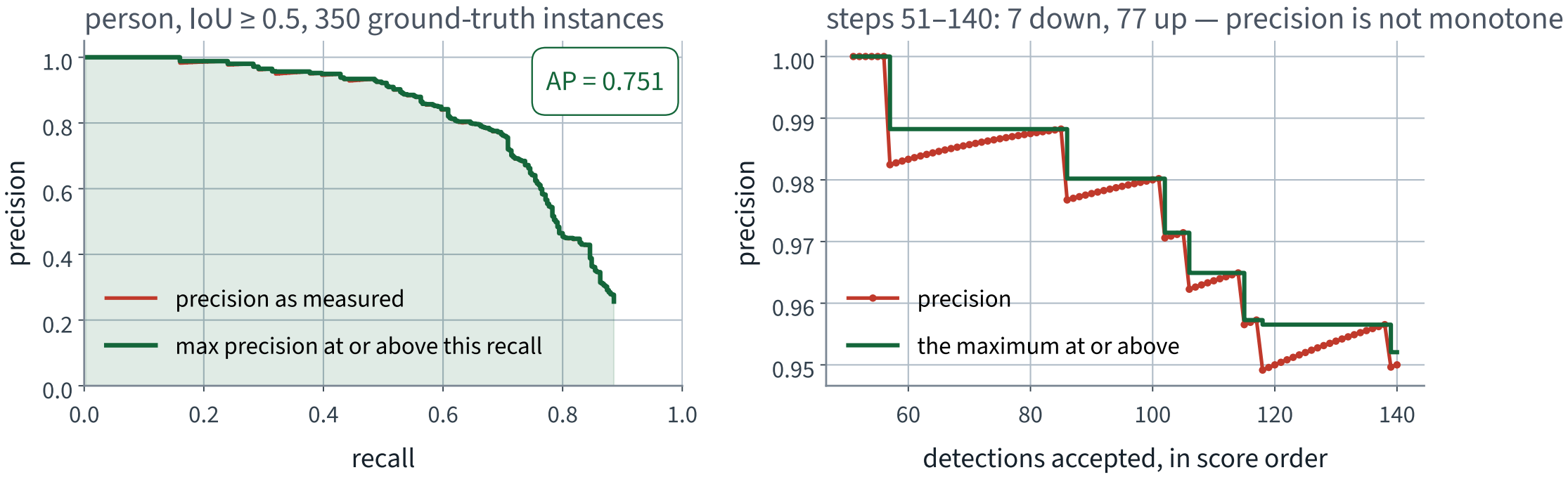
$$\mathrm{GIoU}(A, B) = \mathrm{IoU}(A, B) - \frac{|C| - |A \cup B|}{|C|}$$

GENERALISED IOU

- The subtracted term is the fraction of the enclosing box that is *neither* prediction nor truth
- When the boxes are disjoint and move apart, $|C|$ grows and $|A \cup B|$ does not, so the penalty grows
- Range $(-1, 1]$, and it equals IoU whenever $C = A \cup B$

Average precision, from a ranking

Rank one class's detections by score. Each is a true positive if its IoU with a still-unmatched annotation is at least t. AP is the area under the running maximum of precision, $p_{\text{env}}(r) = \max_{\tilde{r} \geq r} p(\tilde{r})$: detection's interpolated AP, which COCO's evaluator samples at 101 recall levels. Scikit-learn's `average_precision_score`, used in Lecture 3, takes no maximum: it averages the raw precision at each positive.



Precision is not monotone (right, ranks 51 to 140: 7 steps down and 77 up), as Lecture 3 proved; the maximum is what repairs it.

The first mean: over classes

$$\mathrm{mAP}(t) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \mathrm{AP}_c(t)$$

$\mathcal{C}$ = THE CLASSES PRESENT IN THE CORPUS

- Unweighted. A class with one annotation counts as much as `person` with 350
- We average over the 73 classes that appear in our 128 images. The other 7 of COCO's 80 have no annotations here and no AP to average

✗ That choice — which classes are in the mean — is a decision, it changes the number, and papers do not always say which one they made.

The second mean: over IoU thresholds

$$\mathrm{mAP}_{[.5:.95]} = \frac{1}{10} \sum_{t \in \{0.50,\, 0.55,\, \ldots,\, 0.95\}} \mathrm{mAP}(t)$$

- At $t = 0.50$ a box that half-covers the object counts as correct
- At $t = 0.95$ it must be very nearly exact
- Averaging the ten is COCO's headline number, and it is what "mAP" means unqualified in a modern paper

Step 1 · Specify, before any code is written

Input	128 JPEG files and one COCO annotation file
Output	per image: boxes in corner form, category ids, scores; plus one integer count
Constraint	nothing is trained; the pretrained weights are used as they ship
Check	every predicted box lies inside the image; scores are non-increasing; the count equals the number of boxes kept

The *Check* row is the one people skip, and it is the one that catches a corner-versus-size mix-up in one line.

Step 2 · The detector, off the shelf

```python
import torch
from torchvision.models.detection import (
    fasterrcnn_resnet50_fpn, FasterRCNN_ResNet50_FPN_Weights)

weights = FasterRCNN_ResNet50_FPN_Weights.COCO_V1
model = fasterrcnn_resnet50_fpn(weights=weights)
model.eval().to(DEVICE)                    # eval(), every time
preprocess = weights.transforms()

print(len(weights.meta["categories"]), "label slots")
print(weights.meta["_metrics"])
```

`model.eval()` is in the silent-failure catalogue from Lecture 8. Here it changes the batch-norm statistics inside a frozen ResNet, and nothing raises.

Non-examinable: the torchvision weights API.

Step 3 · Run it over the corpus

```
from PIL import Image

preds = {}
with torch.inference_mode():                # no graph, no gradients
    for im in images:
        img = Image.open(IMG_DIR / im["file_name"]).convert("RGB")
        out = model([preprocess(img).to(DEVICE)])[0]
        preds[im["id"]] = {k: v.cpu().numpy() for k, v in out.items()}

assert len(preds) == 128
```

One forward pass per image, with no gradients. The loop prints nothing while it runs; no output does not mean it has hung.

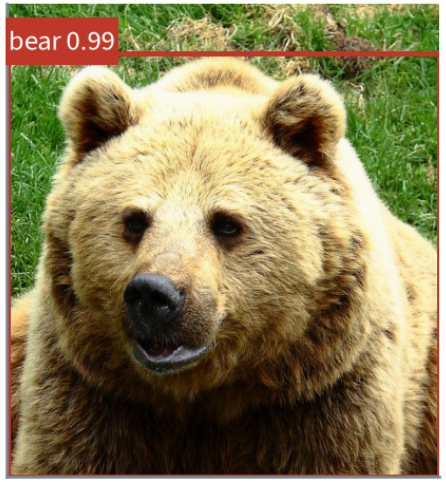
Step 4 · It works, and you can see it

Faster R-CNN, score ≥ 0.5, on three of our 128 images; labels shown for scores ≥ 0.90

25 boxes (truth: 20)



1 box (truth: 1)



31 boxes (truth: 17)

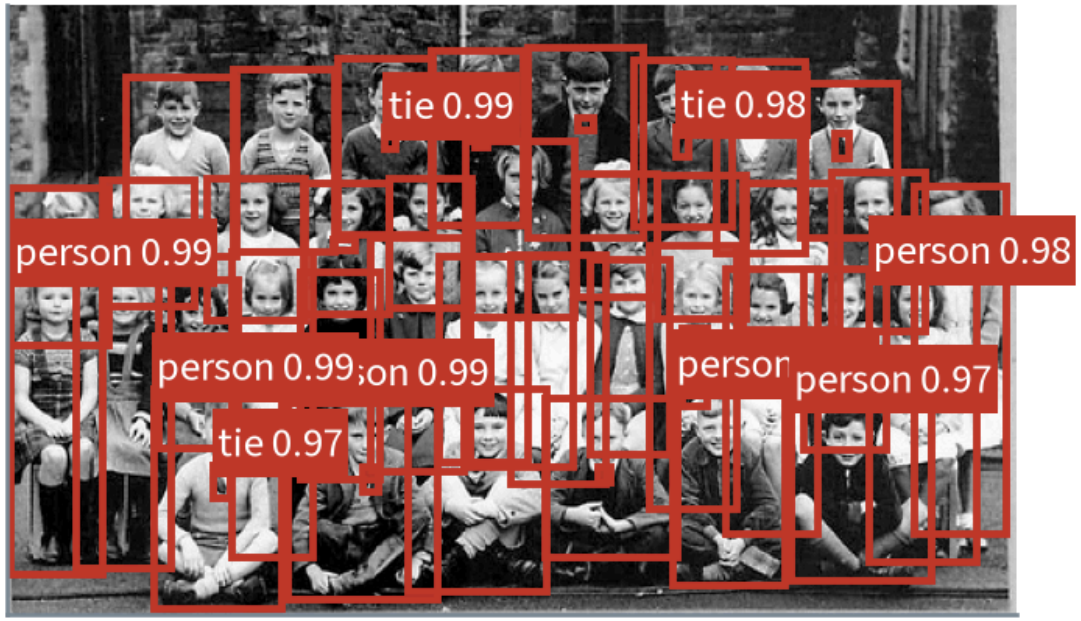


The bear is right. The two rooms have more boxes than there are objects, and you can see which ones are wrong without being able to say how wrong.

Where it visibly struggles

the two most crowded images in the first forty; labels shown for scores ≥ 0.97

50 boxes (truth: 22)



25 boxes (truth: 20)



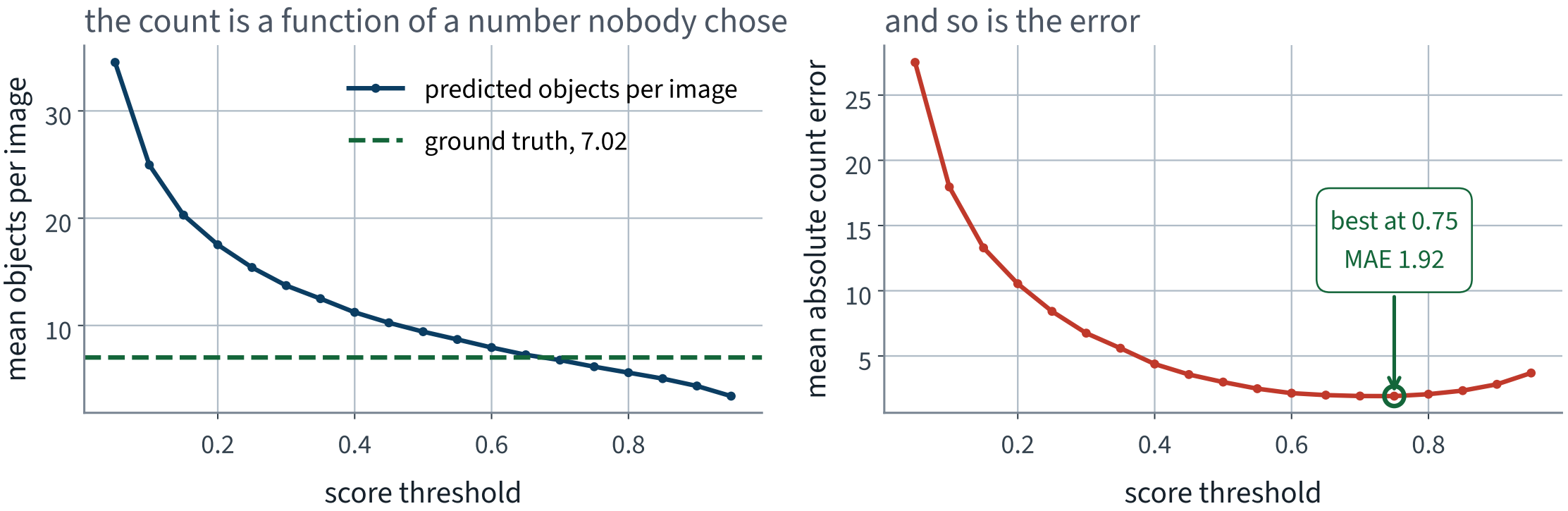
Fifty boxes for twenty-two objects. Several children carry two or three, and nothing says which is the good one.

Step 5 · There is already a cut-off, and it is not yours

- `box_score_thresh` defaults to **0.05**: anything below is never returned at all
- `box_nms_thresh` defaults to **0.5**: near duplicates are already removed before you see them
- `box_detections_per_img` defaults to **100**

> **READ THE DEFAULTS OF EVERY CALL**
>
> Three settings nobody chose here, and one is a hard cap you will hit: our busiest image returns exactly 100 boxes.

Step 6 · The count is a function of the threshold



Nothing in the model chose 0.5. Move the cut-off and the answer to the stakeholder's question changes by a factor of ten.

Pick one, and write down why

> **OUR OPERATING POINT**
>
> **0.5**, because it is the conventional reporting threshold and because it was chosen *before* looking at the error curve.

The minimum of the right-hand panel is at 0.75, with MAE 1.92. Adopting *that* would be choosing a hyperparameter on the same data you then report — Lecture 2's error, in a new costume. We will quote it as a diagnostic and not as a result.

Step 7 · Count the objects

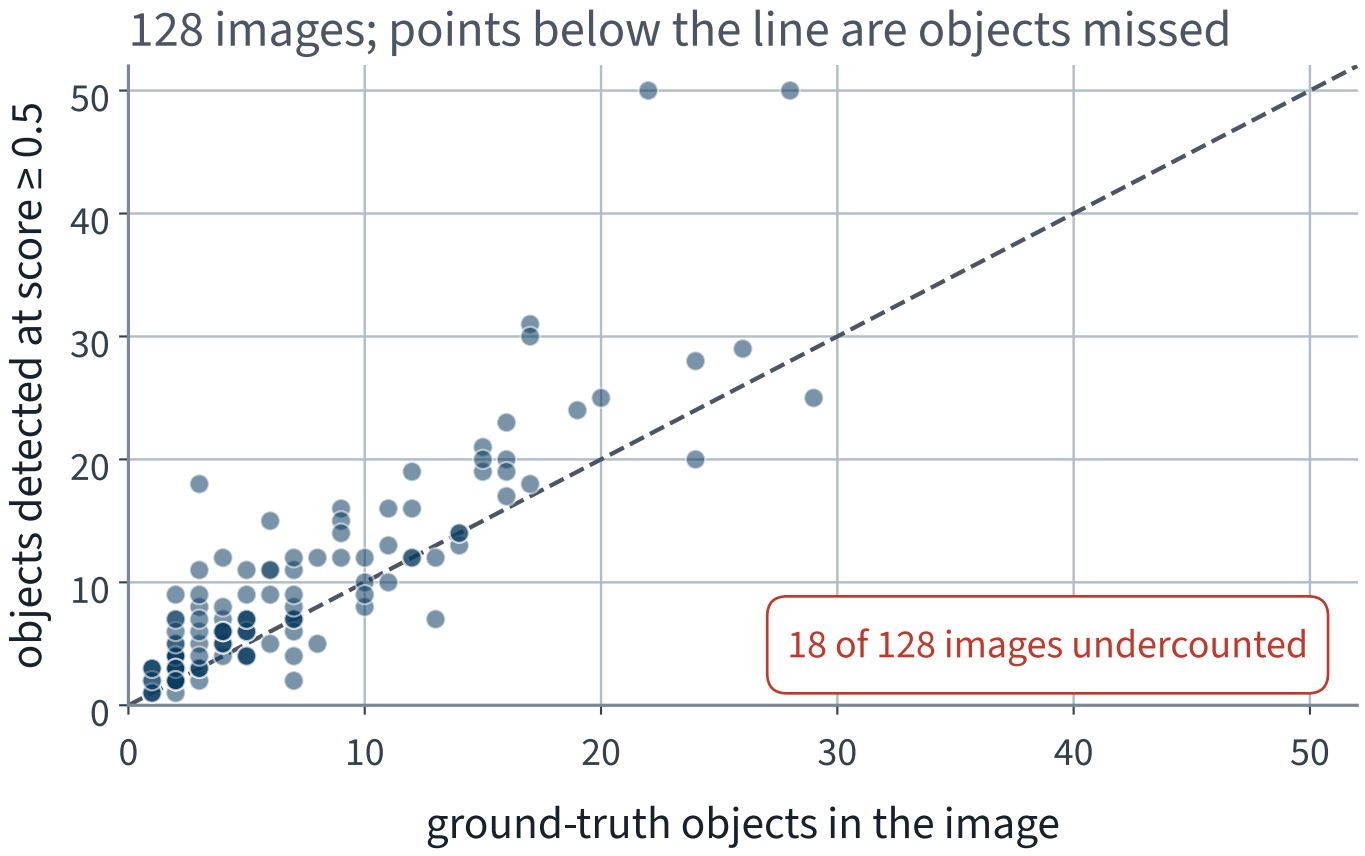
```
THRESH = 0.5

n_pred = np.array([int((preds[im["id"]]["scores"] >= THRESH).sum())
                   for im in images])
n_true = np.array([len(gt[im["id"]]["labels"]) for im in images])

assert n_pred.shape == n_true.shape == (128,)
mae  = np.abs(n_pred - n_true).mean()
bias = (n_pred - n_true).mean()
print(f"count MAE  {mae:.2f}")
print(f"signed     {bias:+.2f}   (positive = too many boxes)")
```

Two numbers, not one. The absolute error says how far off; the signed error says which way.

Step 8 · Every image, plotted



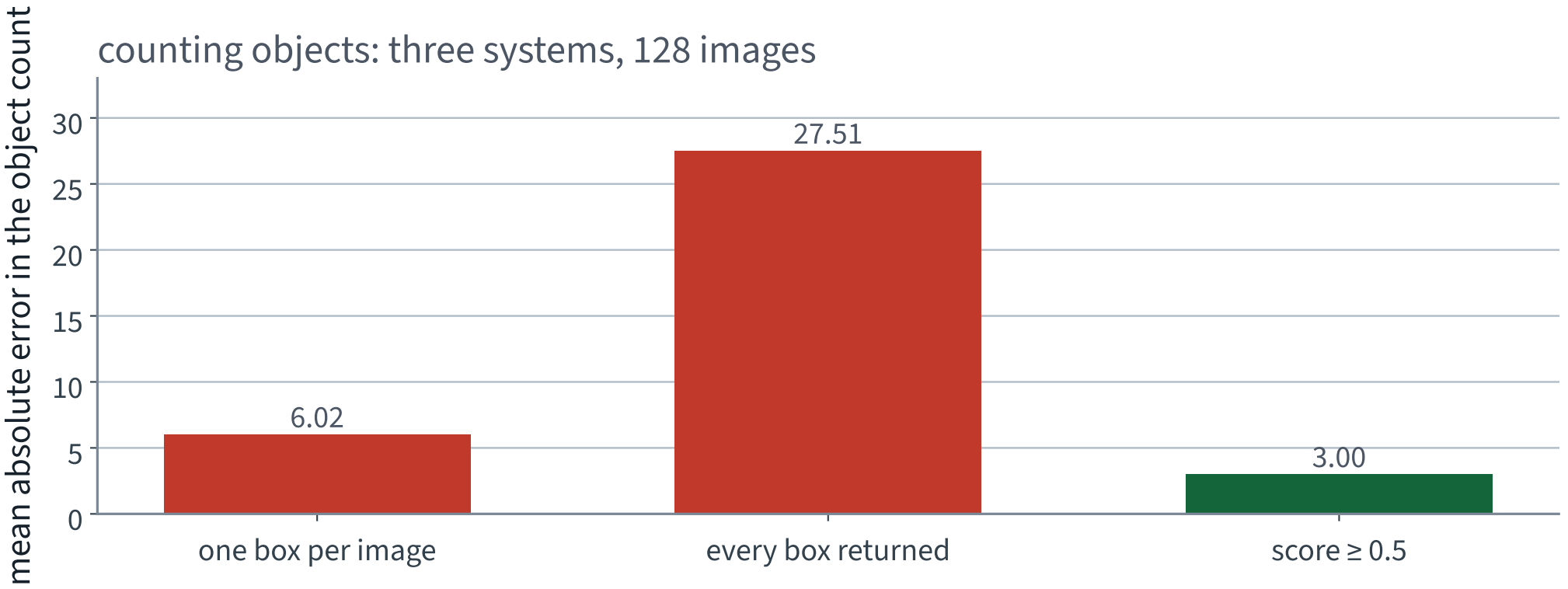
The cloud sits *above* the diagonal: the failure mode is too many boxes, not too few.

The result, on 128 images

At score ≥ 0.5	Value
Count MAE	**3.00 ✓**
Signed count error	**✗ +2.41**
Images counted exactly right	27 of 128
Images with too many boxes	**✗ 83**
Images with too few	18
Boxes kept in total	1,206

✓ 3.00 against a baseline of 6.02. The model halves the error of a system that never looks at the image.

Three systems, one metric



The middle bar is the same model with one line deleted, and it is four times worse than predicting nothing at all.

Before quoting our own number, check it against someone else's

We derived AP on these slides and implemented it ourselves. That is exactly the situation in which a result should not be believed.

mAP on our 128 images	Ours	Reference evaluator
at IoU 0.50	0.659	0.660
at IoU 0.75	0.475	0.476
averaged 0.50 to 0.95	0.439	0.440

✓ Largest disagreement: 0.0014. The reference is the official COCO evaluator, run on the same predictions and the same annotations, by a library we did not write.

Non-examinable: the reference library. Examinable: that you check an implementation against one.

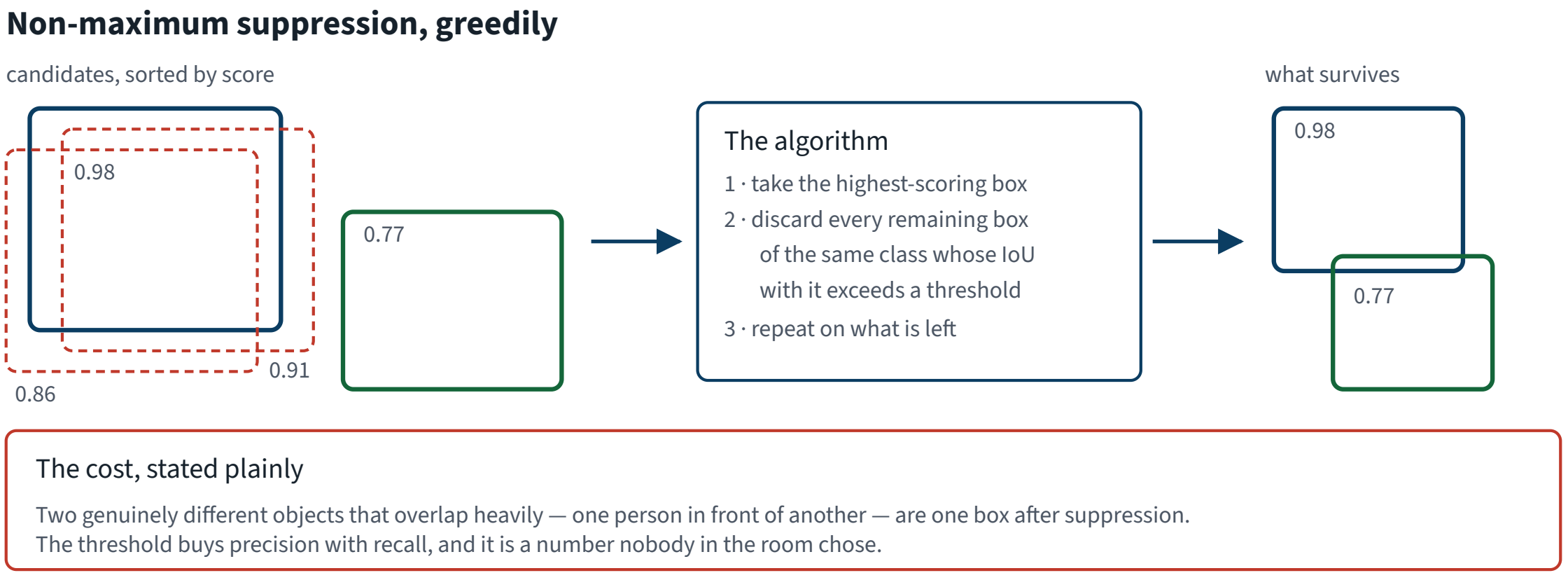
Now compare 128 images against 5,000

mAP, averaged 0.50 to 0.95	Value	Images
Ours	43.9	128
Reported by torchvision for these Faster R-CNN weights	37.0	5,000

✗ Our number is 6.9 points higher, and the model is identical.

Nothing about the detector changed. The difference is the sample: 128 images, of which many are easy, and 73 classes several of which have a single instance that happened to be found. This is what a small evaluation set does, and it does it in the flattering direction more often than not.

Non-maximum suppression: the rule



The `box_nms_thresh` of Step 5. Greedy, per class, and it deletes information that cannot be recovered afterwards.

Suppression, in code

```
from torchvision.ops import batched_nms

keep = batched_nms(torch.tensor(boxes),      # (N, 4), corners
                   torch.tensor(scores),     # (N,)
                   torch.tensor(labels),     # (N,) — per class
                   iou_threshold=0.5)

assert len(keep) <= len(boxes), "suppression cannot add boxes"
boxes, scores, labels = boxes[keep], scores[keep], labels[keep]
```

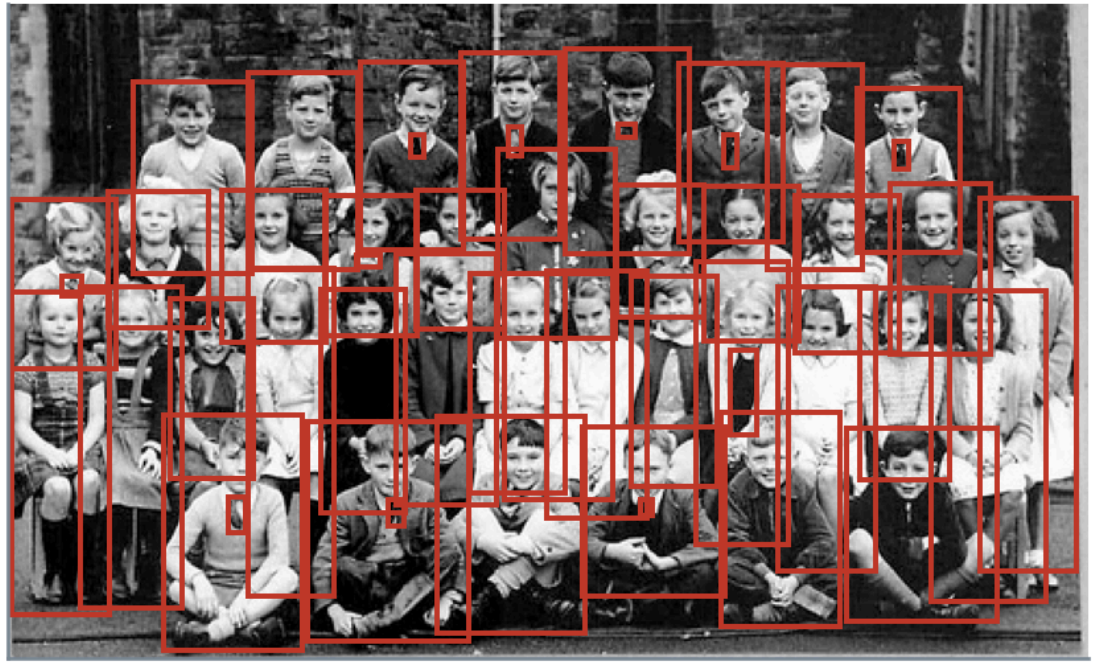
`batched_nms` rather than `nms`: a bottle overlapping a glass must not suppress it, because they are different classes and both are true.

Before and after, on one image

no suppression: 300 boxes (the 300-candidate cap)

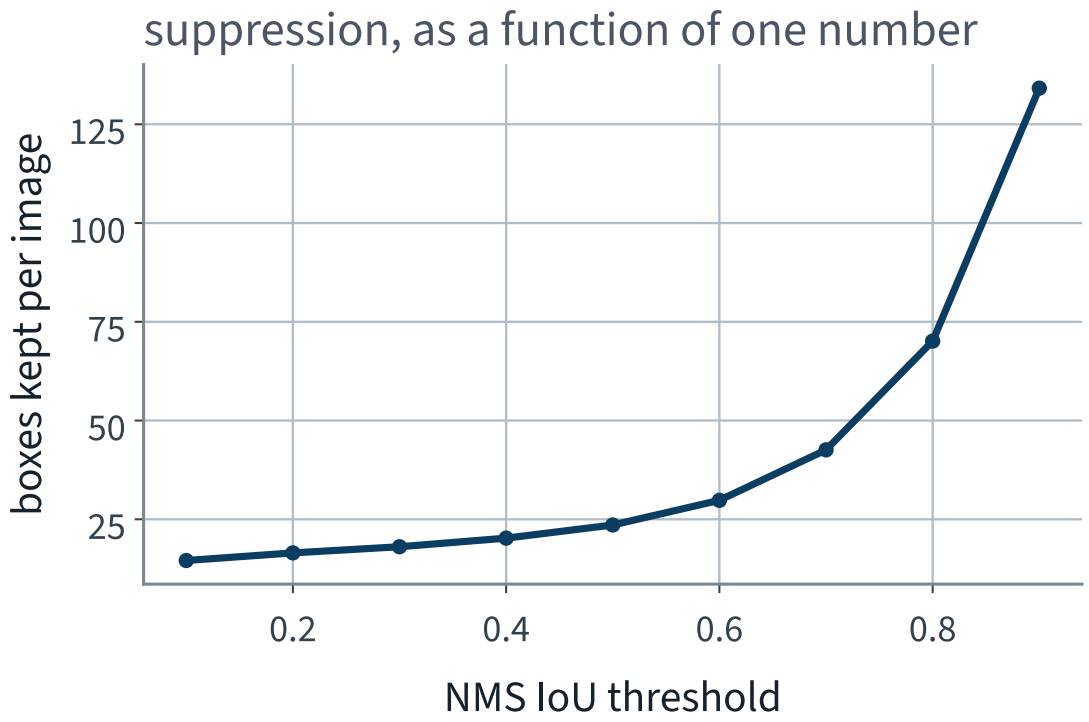


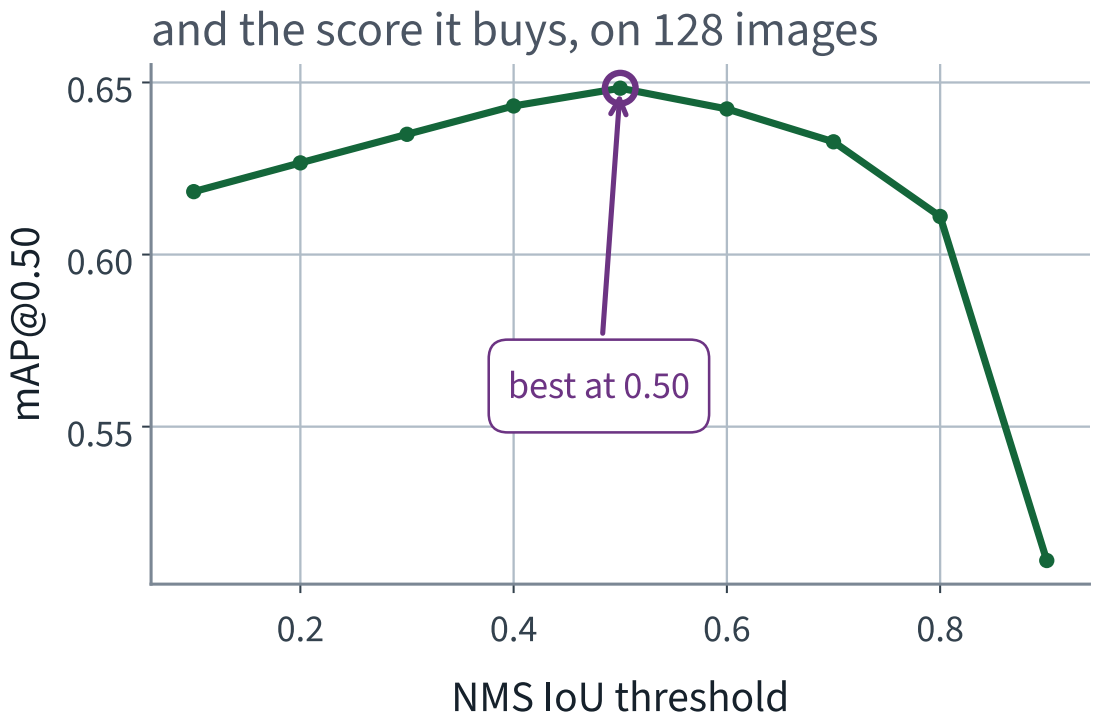
NMS at IoU 0.5: 50 boxes



Same model, same image, same score threshold. The only difference is one IoU comparison between pairs of boxes.

And it is another knob





The default of 0.5 happens to be near the best value on our corpus. That is a fact about this corpus, not a law.

Tracking: one identity across frames, in four lines

1. Predict where each existing track's box will be in this frame
2. Compute IoU between every prediction and every new detection
3. Solve the assignment: pair them up to maximise total IoU
4. Unmatched detections start new tracks; unmatched tracks age and die

WHERE IT FAILS, AND WHY IT IS IOU'S FAULT AGAIN

An object that is occluded for ten frames reappears somewhere its predicted box does not overlap. IoU is zero, the assignment finds nothing, and the object is issued a new identity. The same flatness, in a new place.

FAILURE CONDITION — IDENTITY CARRIED BY OVERLAP ALONE

A tracker that matches boxes by IoU loses an object the moment it is occluded for long enough to reappear somewhere else. It does not report a lost track; it reports a new one, and the count it hands the stakeholder is wrong in a way the video makes obvious and the number does not.

Semantic segmentation: a label on every pixel

A box was always an approximation: a bicycle's box is mostly not bicycle, and for a picking arm "somewhere in this rectangle" may not be good enough.

- Output: one class label per pixel. Shape $H \times W$, not $N \times 4$
- Two people standing together are one `person` region. You cannot count them
- The architectural problem is that pooling destroyed the resolution the answer needs — which is Lecture 12's invariance, arriving as a cost
- The repair is upsampling with skip connections: transposed convolutions that recover spatial detail from the earlier, higher-resolution layers

Instance segmentation

- Output: one binary mask per detected object, plus its class and score
- Mask R-CNN is Faster R-CNN with one extra head that predicts a small mask inside each proposed box
- Two lines of code away from what you already ran

```
from torchvision.models.detection import (
    maskrcnn_resnet50_fpn, MaskRCNN_ResNet50_FPN_Weights)

model = maskrcnn_resnet50_fpn(
    weights=MaskRCNN_ResNet50_FPN_Weights.COCO_V1).eval().to(DEVICE)
out = model([preprocess(img).to(DEVICE)])[0]
print(out["masks"].shape)      # (N, 1, H, W), soft masks in [0, 1]
```

Scoring a mask is the same argument again

- Mask AP replaces box IoU with **mask IoU**: intersecting pixel sets instead of rectangles
- Everything above — the ranking, the envelope, the area, the two means — is unchanged
- torchvision reports 34.6 mask mAP for its Mask R-CNN weights on the full 5,000, against 37.9 box mAP for the same model

A mask is harder than a box, which is what those two numbers say. We do not measure mask AP on our corpus; the pixel-level annotations are in the file and the code is an exercise.

The notebook for this lecture

Open Lecture 14 in Colab

- **Runs on free CPU.** 128 COCO images and a pretrained detector, inference only — nothing is trained here
- IoU implemented and checked against the cases the derivation names, including the disjoint one where the gradient vanishes
- AP computed from the sawtooth and from its running maximum, so the repair is something you compute rather than accept

> **WHAT TO CHANGE**
>
> Move one predicted box further and further from its match, and watch IoU sit at exactly zero the whole way. That flat zero is the vanishing gradient the derivation predicts.

Design exercise · in the manner of an ML system-design interview

The task: a parking-analytics company’s occupancy product

> **THE BRIEF**
>
> A parking-analytics company flies drones over car parks and sells operators and cities, per image, **how many** vehicles are parked and **where** each one is — a count and boxes — so an operator checks bays against bookings and a city plans. The business measures contracts kept, their accuracy clause, and the cost of a count proved wrong.

- **Given:** a few hundred drone images from a dozen sites, each vehicle boxed and classed — car, van, lorry, motorcycle — by a contractor; a pretrained detector
- **Scale:** a few hundred vehicles per full site; images several thousand pixels across; several flights a day per site
- **Consumers of the output:** the operators’ dashboard and booking check, the cities’ planners

✓ Nine steps follow, each with the pitfall that catches a first attempt and what to do about it. The model at the centre is today’s: an off-the-shelf detector, scored by IoU matching and average precision.

Step 1 — Frame it: boxes, a count, and what a correct answer is

- Output per image: boxes in corner form, a class and a score for each, and one integer count
- A predicted box is correct when its IoU with a still-unmatched annotation of the same class clears a threshold stated in advance; the count is a separate number with its own error, absolute and signed
- Fix the box convention once, at the edge: the tool exports a corner and a size, the detector returns two corners; convert on load and assert that the second corner lies below and to the right

> **PITFALL — THE COUNT ALONE AS THE METRIC**
>
> A detector that misses the cars under the trees and doubles the ones in the sun returns the right total: the count error is near zero, the boxes are wrong, and the operator’s bay check fails. Score the boxes by IoU matching, and the count beside them with its sign.

Step 2 — The annotations: what the contractor’s boxes are worth

- A box drawn on a drone image is a judgement: does it include the shadow, the open door, the roof rack; is a car half under a canopy annotated at all; is a trailer a vehicle
- Write the rule down, give the same images to two annotators, and measure their IoU on the same vehicles

PITFALL — A THRESHOLD THE ANNOTATORS CANNOT MEET

Two careful people boxing the same car from the air disagree by more than the strict thresholds allow, so at those thresholds a perfect detector is scored wrong by the annotation’s own noise, and the headline mean over thresholds is capped by the annotators, not by the model. Report the inter-annotator IoU beside the evaluation, take the loose threshold as the operating number, and send the disagreements back before anything is scored.

Step 3 — Split by site, and keep the empty tarmac in

- Hold whole sites out, as a new customer's is: tiles from the same flight share the vehicles, light and paint, so a split by tile scores memory
- Most of every image is background — tarmac, painted bays, drain covers, shadows — and the evaluation set must hold it in its true proportion; report the baselines of no boxes and one box per tile

PITFALL — THE BACKGROUND AS THE DOMINANT CLASS, DROPPED

A first attempt keeps the tiles with vehicles and discards the empty ones as "nothing to annotate". The false positives on empty bays — the markings and drain covers a detector likes — are never counted, and the dashboard overstates quiet sites. Keep the empty tiles, annotated as empty, and report the boxes returned on them.

Step 4 — Representation: the vehicle at the detector's input size

- The detector's transform resizes the image to its own working size: a drone frame several thousand pixels across is shrunk several-fold, and a car that was forty pixels long becomes a handful
- The backbone's pooling discarded resolution on purpose; a detector keeps what it can, but not an object a few pixels across

PITFALL — SMALL OBJECTS AT THAT RESOLUTION

Resize the whole frame and the vehicles at the far edge of an oblique shot vanish below what the feature maps can resolve. The AP is fine on the near, large vehicles and zero on the far, small ones, and the mean hides the second. Tile the frame at native resolution, or fly lower, and check it with an AP per object size.

Step 5 — The model: the detector as it ships, and its three defaults

- Load the pretrained detector in `eval()` mode; nothing is trained; run under inference mode
- Three settings nobody chose: a score cut-off, a suppression threshold, and a cap on detections per image. A full site holds more vehicles than the default cap, so the count saturates: raise it before measuring
- Pick the score cut-off before looking at the error curve, and say so in the contract; the minimum of that curve on your own images is a diagnostic, not a result

> **PITFALL — NON-MAXIMUM SUPPRESSION**
>
> Greedy, per class: a car parked bumper to bumper with the next, seen obliquely, overlaps it by more than the threshold and the second is deleted; the count comes out one short per row, and nothing raises. Suppress per class, raise the threshold, and check it on the packed rows — the sweep on your own corpus says what it does here, not in general.

Step 6 — Failure condition: where it fails, by size and by place

- The detector misses what is small, shaded or partly covered: the near half of an oblique frame is easy and the far half is not, and one AP over the whole site averages them
- Bucket the annotations by box area and compute AP for each bucket, as the reference evaluator does; and the count error by region of the frame

> **PITFALL — EVALUATION PER OBJECT SIZE, SKIPPED**
>
> The single AP is carried by the large near vehicles, the small-object AP is a fraction of it, and the customer’s complaint — the far rows are always under-counted — is invisible in the number reported. Report AP per size bucket and the signed count error per region, and fix the small ones at the input, not the threshold.

Step 7 — Evaluation: what the one number hides

- mAP is a mean over classes of a mean over thresholds: the motorcycle class, with a handful of boxes, counts as much as cars with thousands, and the strict thresholds are the ones the annotators cannot meet
- Check your implementation against the reference evaluator on the same predictions before quoting it
- A few hundred images flatter a detector, in the easy direction more often than not: say the sample size beside every number

> **PITFALL — A HEADLINE FROM YOUR OWN EVALUATOR ON A SMALL SET**
>
> A mAP computed by code you wrote, on a few hundred images, several points above the published figure for the same weights, goes in the contract as a gain; the gain is the sample, and possibly the evaluator. Match the reference to within rounding first, then state the sample size, the operating threshold and the per-class APs with each figure.

Step 8 — In production: tiles, seams, and the barrier count

- A site image is cut into overlapping tiles at native resolution, each run through the detector, the boxes mapped back to the frame
- The barrier's ticket count is a free ground truth for the total: a new detector runs in shadow beside the old on every flight, its signed error against the barrier read before it takes over, then watched for a step — new paint, snow, a canopy

PITFALL — TILED INFERENCE AND ITS SEAMS

A car cut by a tile edge is two half-cars, one in each tile, and both are detected — or neither, if each half is too small to clear the cut-off; the count doubles along every seam. Overlap the tiles by more than a vehicle's length, map the boxes to the frame, run suppression once more across it; then check the count against the barrier's.

Step 9 — Ship it: the pipeline, and the pitfalls in one table

Boxes to corners once, asserted → annotation rule, inter-annotator IoU → split by site, empty tiles kept → native-resolution tiles → detector as shipped, cap raised, cut-off fixed first → suppression per class → AP per size, signed count → evaluator checked against the reference → tiles merged, suppressed across seams → shadow run against the barrier.

Pitfall	Address it by
The count alone as the metric	IoU matching of the boxes; the count beside, with its sign
Thresholds the annotators cannot meet	inter-annotator IoU measured; operate at the loose threshold
Empty tiles dropped	empty tiles kept, their false positives counted
Small vehicles shrunk away	tiles at native resolution; AP per size
Suppression merging neighbours	per-class suppression, threshold checked on packed rows
One AP over all sizes	AP per size bucket, count error per region
A headline from a small set	the reference matched; sample size and threshold stated
Double-counting at tile seams	overlapping tiles, suppression across the frame

What we did

1. Established that a label is not an answer when the question is *where* and *how many*
2. Defined IoU, and saw it is exactly zero for every pair of disjoint boxes — so as a loss it gives no gradient however far apart they are; GIoU repairs that with a term that keeps moving
3. Built average precision on a ranking, with a maximum in its definition because precision is not monotone (Lecture 3)
4. Read mAP as a mean over classes of a mean over IoU thresholds, and saw how much each averaging hides
5. Ran a detector, chose a score cut-off before looking and defended it, checked our mAP against a reference evaluator, and saw 128 images flatter it
6. Applied non-maximum suppression, tracked by IoU, and looked at what a box cannot express: semantic against instance segmentation
7. Designed the occupancy product a parking-analytics company sells to operators and cities in nine steps — boxes scored by IoU beside a signed count, an annotation rule measured between annotators, empty tarmac kept in the evaluation, tiles at native resolution for the small vehicles, suppression checked on packed rows, AP per object size, the evaluator matched to the reference with the sample size stated, and tiles merged with suppression across the seams — each step named for the pitfall it avoids

NOT PRESENTED — FOR AFTERWARDS

Exercises Lecture 14

five, in the style of the written paper

Exercises — Lecture 14

1 Two boxes are disjoint and moved further apart. State what IoU and its derivative do, and why no learning rate repairs it. [5]

2 A detector counts objects with MAE 3.00 at the conventional score cut-off of 0.5. A sweep over the same 128 images finds MAE 1.92 at 0.75. State why 1.92 must not be reported as the result, and how it may be quoted. [5]

3 Average precision is defined with a maximum inside it. State what that maximum repairs, and which earlier lecture proved the problem. [4]

Solutions on Lecture 15’s deck.

Exercises — Lecture 14 (continued)

4 mAP is a mean over classes of a mean over IoU thresholds. Give one thing each averaging hides. [4]

5 Your mAP over IoU 0.50 to 0.95 on 128 COCO images is 43.9; torchvision reports 37.0 for the same weights on all 5,000. Name the check that rules out your own evaluator first, and then the cause of the gap. [4]

Solutions on Lecture 15’s deck.

THE FIVE SET AT THE END OF LECTURE 13

Solutions Lecture 13

not part of this lecture

Solutions — Lecture 13

1. State the order in which a pretrained network's head must be replaced and its body frozen, and what…

✓ Freeze first, then replace. In the other order the new head is frozen too.

- A newly constructed module has `requires_grad=True`, so the freezing loop would turn it off
- Nothing is left to train: counting the trainable parameters gives 0 instead of 52,326

2. Fine-tuning uses 10^{-4} on the pretrained block and 10^{-3} on the new head. Justify the difference in…

✓ The block already encodes something and the head starts from noise; one rate cannot suit both.

- A large step destroys what the block knows
- A small step leaves the random head untrained

Solutions — Lecture 13 (2)

3. Freezing a batch-norm layer's weights does not freeze the layer. State what still changes, and the call…

✓ The running statistics are buffers, updated in the forward pass; `.eval()` stops them.

- `requires_grad=False` governs gradients, not buffers
- A frozen feature extractor left in train mode still drifts

4. An augmented validation loader is used to select the epoch to keep. State the two things this costs, and the…

✓ The score becomes a random variable and the chosen epoch can change. Evaluate twice and require identical answers.

- Augmentation makes the metric depend on the evaluator's seed
- A deterministic function of fixed weights and fixed data does not wobble

Solutions — Lecture 13 (3)

5. A frozen-backbone probe is both more accurate and faster than training from scratch on 1,020 images. Explain…

✓ The backbone is a fitted function reused for free; the feature extraction pass must still be counted in the total.

- Only the head is fitted, so the variance problem moves to a dataset large enough to absorb it
- The probe is not free just because the head is

LECTURE 15 · GÉRON CH. 13

Time series

Weak stationarity and differencing, on a real daily series
Why copying last week is hard to beat, and why a random split is invalid on autocorrelated data
Backtesting on a rolling origin, and a linear model on eight weeks of lags

Run the Lecture 14 notebook first