African Institute for Mathematical Sciences — Cameroon
Structured Master's, Data Science Stream · Computer Vision

What, and Where?

Week 2 · Day 3: Object Detection, from R-CNN to YOLO

Yusuf Brima, Ph.D.

Introduction to Computer Vision · AIMS Cameroon, 2026–2027

Today

1. **The detection problem**: boxes, classes, and how to compare boxes
2. **From classification to detection**: localisation, sliding windows, proposals
3. **Two-stage detectors**: R-CNN, Fast R-CNN, Faster R-CNN
4. **One-stage detectors**: YOLO, SSD, RetinaNet; anchors and grids
5. **Transformers for detection**: DETR
6. **Measuring detectors honestly**: precision, recall, AP, mAP, on a real dataset

By the end of today you should be able to

- compute IoU and run non-maximum suppression by hand;
- explain how Faster R-CNN and YOLO produce boxes;
- explain anchors, and why DETR does not need them;
- compute precision, recall and average precision;
- choose a detector for a given speed and accuracy budget.

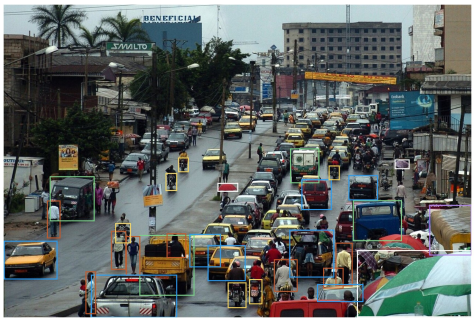
Where we are

So far this week

- CNNs turn an image into *one* label.
- Grad-CAM showed that a street photo contains evidence for several classes in different places.

Today

- Find **every** object, say **what** it is and **where** it is.
- This is what counts cars at a Douala junction, finds parasites in a smear, locates weeds for a sprayer.



Happiraphael, CC BY-SA 4.0

Part 1

The detection problem

What a detector outputs

One detection

$$(\underbrace{x_1, y_1, x_2, y_2}_{\text{box}}, \underbrace{c}_{\text{class}}, \underbrace{s}_{\text{confidence}})$$

A detector outputs a **variable-length list** of these, one per object.

- Boxes are axis-aligned rectangles: corners (x_1, y_1), (x_2, y_2), or centre (c_x, c_y) plus width and height.
- The number of objects is unknown in advance: 0 in an empty road, 30 at a busy crossroads.

Intuition

Classification answers a multiple-choice question. Detection fills in a form of unknown length: for each object, a class, a position, a size and a confidence.

Practitioner's Note

Check your coordinate convention: pixels or $[0, 1]$; corners or centre; (x, y) or (row, col). Mixing them is the most common detection bug.

Box formats, worked

One pedestrian, four notations

Corners: $(x_1, y_1, x_2, y_2) = (40, 60, 120, 220)$.

Centre + size: $c_x = 80$, $c_y = 140$, $w = 80$, $h = 160$.

COCO JSON: $[x, y, w, h] = [40, 60, 80, 160]$ (top-left corner + size).

YOLO text, on a 640×640 image (centre + size, divided by image size):

```
0 0.125 0.219 0.125 0.250
```

- Normalised coordinates survive resizing: the same numbers describe the box at any resolution.
- The leading `0` in the YOLO line is the class index (here, "person").

Practitioner's Note

After converting a dataset, *draw* a few boxes on their images. A box drawn in the wrong place is the fastest way to catch a format mistake.

The datasets that shaped detection

Dataset	Classes	Size	Notes
PASCAL VOC (2007–12)	20	about 11 500 images	the classic benchmark of the feature era
MS COCO (2014)	80	328 000 images, 2.5M objects	everyday scenes, many small objects; the standard today
Open Images (2018–)	600	1.9M images, 16M boxes	larger, noisier, more diverse
Penn-Fudan (2007)	1	170 images	small; we use it today to evaluate

Intuition

"COCO-pretrained" detectors know COCO's 80 classes: people, vehicles, animals, household objects. That defines what they can find out of the box, and what they will miss.

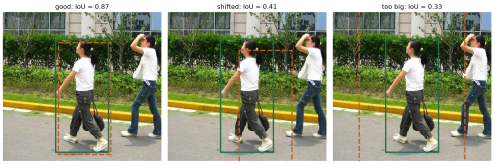
Comparing two boxes: intersection over union

IoU

$$\mathrm{IoU}(A, B) = \frac{\text{area}(A \cap B)}{\text{area}(A \cup B)}$$

By hand

Ground truth A: $(10, 10)$–$(50, 90)$, area $40 \times 80 = 3200$.
Prediction B: $(20, 20)$–$(60, 100)$, area 3200.
Overlap: $(20, 20)$–$(50, 90)$, $30 \times 70 = 2100$.
Union: $3200 + 3200 - 2100 = 4300$.
IoU $= 2100/4300 = 0.49$.

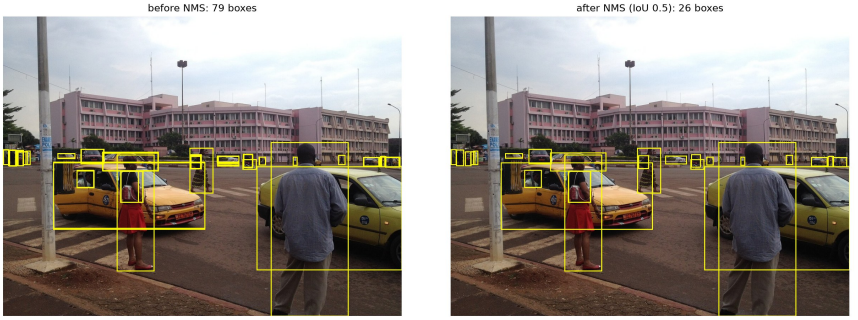


Real pedestrian from Penn-Fudan: green = ground truth, orange = prediction. IoU: 0.87 (good), 0.41 (shifted), 0.33 (too big).

Intuition

IoU is 1 for a perfect box and 0 for no overlap; it punishes both shifting and wrong size. A detection usually "counts" when IoU ≥ 0.5.

Too many boxes: non-maximum suppression



Bdx, CC BY-SA 4.0

Detectors fire many overlapping boxes on each object. NMS (Week 1, Day 5): sort by score; keep the best; delete boxes overlapping it with IoU > 0.5; repeat. Here: 79 boxes become 26.

Practitioner's Note

NMS can also delete *real* objects that overlap strongly, such as people in a dense crowd. Its IoU threshold is a trade-off you should tune for your scenes.

Non-maximum suppression, by hand

Four boxes, IoU threshold 0.5

Scores: $A = 0.95$, $B = 0.90$, $C = 0.80$, $D = 0.60$.
$\text{IoU}(A, B) = 0.70$, $\text{IoU}(A, C) = 0.10$, $\text{IoU}(B, C) = 0.05$, $\text{IoU}(C, D) = 0.60$, $\text{IoU}(A, D) = 0.0$.

1. Keep A (best). Remove B ($0.70 > 0.5$). C, D survive.
2. Next best: keep C. Remove D ($0.60 > 0.5$).

Result: $\{A, C\}$, two objects.

- Run separately for each class: a person box should not suppress an overlapping motorcycle box.
- Cost: sorting plus pairwise IoUs; fast in practice (`torchvision.ops.nms`).

Intuition

Greedy and local, like Canny's non-maximum suppression on edges (Week 1, Day 4): keep the peak, suppress its neighbours.

When NMS hurts: crowds

- Two passengers on one moto-taxi, or people in a queue, have heavily overlapping boxes. NMS may delete a real person.
- Raise the IoU threshold: fewer real objects lost, more duplicates kept. No single value is right for all scenes.
- **Soft-NMS** (2017): instead of deleting overlapping boxes, *lower* their scores in proportion to the overlap.

Soft-NMS, linear version

$$s_i \leftarrow s_i \left(1 - \mathrm{IoU}(M, b_i)\right) \quad \text{if IoU} > N_t$$

M: the box just kept. Heavy overlap means a big penalty; a strong second object survives with a reduced score.

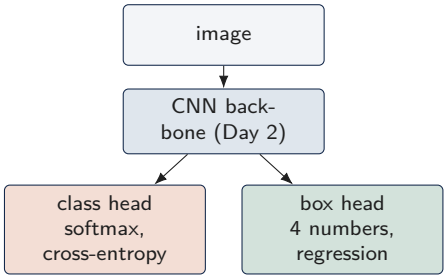
Practitioner's Note

For crowded scenes, also consider detectors trained on crowd data, and evaluate recall separately on crowded images.

Part 2

From classification to detection

Step 1: one object, so predict its box



- **Localisation**: if the image has exactly one main object, add a second head that outputs 4 numbers: the box.
- Train with a loss on both: cross-entropy for the class, a regression loss (e.g. L_1) for the box.

Multi-task loss

$$\mathcal{L} = \mathcal{L}_{\text{class}} + \lambda \, \mathcal{L}_{\text{box}}$$

Intuition

A box is just four more numbers to predict. The hard part of detection is not the box; it is the *unknown number* of objects.

Step 2: many objects. Sliding windows again?

- Week 1, Day 5: slide a window over every position and scale; classify each one.
- With HOG + SVM: about 170 000 windows per photo, each cheap.
- With a CNN classifier: each window costs a full forward pass. At 0.1 s per pass, that is **4.7 hours** per photo.

Intuition

We need to classify far fewer windows, or share the computation between them. The next ten years of detection research did both:

- propose a few likely regions (R-CNN);
- compute the CNN once and crop features (Fast R-CNN);
- let the network propose (Faster R-CNN);
- predict everything in one pass (YOLO).

Classical region proposals: selective search

1. Over-segment the image into small regions of similar colour (a graph-based method).
2. Repeatedly merge the two most similar neighbouring regions: similar colour, similar texture, small sizes first, regions that fit together well.
3. Every region formed along the way becomes a candidate box.

- About 2000 proposals per image, about 2 seconds on a CPU.
- Finds almost every object somewhere in its list (high recall), with many false candidates.

Intuition

Grouping by similarity and proximity: the Gestalt principles of Week 1, Day 1, turned into an algorithm. Objects are usually coherent regions, so merging coherent regions tends to produce object-shaped boxes.

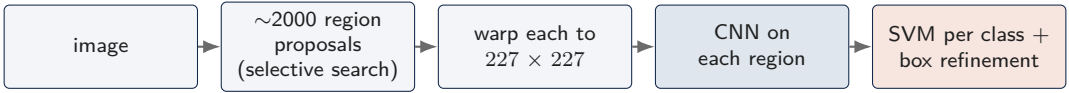
Practitioner's Note

Proposals do not need to be precise; they must not *miss* objects. The classifier and box refinement fix the rest.

Part 3

Two-stage detectors: the R-CNN family

R-CNN (2014): regions with CNN features



- **Selective search** (a classical algorithm) groups similar-coloured, similar-textured pixels into about 2000 candidate regions.
- A CNN, pre-trained on ImageNet and fine-tuned (Day 2's transfer learning), describes each region.
- PASCAL VOC accuracy jumped from 35% to 54% mAP.

Intuition

2000 well-chosen windows instead of 170 000 blind ones. But still 2000 CNN passes: about **47 seconds** per image on a 2014 GPU.

How R-CNN learns

1. **Pre-train** a CNN on ImageNet classification (transfer learning, Day 2).
2. **Fine-tune** it on warped proposals: a proposal with IoU ≥ 0.5 to a ground-truth box takes that box's class; others are "background".
3. **Train an SVM** per class on the CNN features.
4. **Train a box regressor** that nudges each proposal towards its ground-truth box.

Intuition

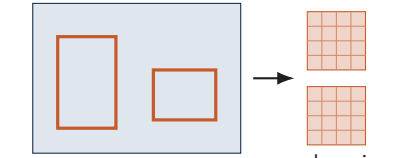
The first big success of transfer learning in vision: PASCAL VOC has only about 11 000 images, far too few to train AlexNet from scratch. ImageNet features made it possible.

Practitioner's Note

Four separately trained stages, and features for every proposal cached on disk (hundreds of gigabytes for VOC). Fast and Faster R-CNN merged them into one network trained end to end.

Fast R-CNN (2015): share the computation

- Run the CNN **once** on the whole image: one feature map.
- For each proposal, crop the corresponding part of the *feature map* and resize it to a fixed size (e.g. 7×7): **RoI pooling**.
- Small heads classify the region and refine its box.
- Trained end to end; about 0.3 s per image (plus proposals).



Intuition

Convolution is shift-equivariant (Week 1, Day 3): the features of a region inside the full feature map are almost the same as the features of the region cropped out first. So compute once, crop many times.

RoI pooling, worked

A 4×6 region to a 2×2 output

Feature values inside one proposal:

$$\begin{bmatrix} 1 & 3 & 2 & 0 & 5 & 1 \\ 4 & 2 & 1 & 6 & 2 & 0 \\ 0 & 1 & 8 & 2 & 1 & 3 \\ 2 & 5 & 1 & 0 & 4 & 2 \end{bmatrix}$$

Split into a 2×2 grid of 2×3 cells; take each cell's maximum:

$$\begin{bmatrix} 4 & 6 \\ 8 & 4 \end{bmatrix}$$

- Every proposal, whatever its size or shape, becomes a fixed-size grid, so the same small head can classify all of them.
- Real detectors use 7×7 cells per channel.
- Mask R-CNN (Day 4) replaces the max with bilinear sampling (RoIAlign) for pixel-exact masks.

Faster R-CNN (2015): the network proposes its own regions

- Selective search was now the bottleneck (2 s per image, on the CPU).
- The **region proposal network (RPN)**: a small conv network sliding over the feature map. At each position it scores a few **anchor** boxes of different sizes and shapes: "object or not?", and nudges them.
- The best few hundred proposals go to the Fast R-CNN head.

Box regression relative to an anchor

Anchor centre (a_x, a_y), size (a_w, a_h). The network predicts (t_x, t_y, t_w, t_h):

$$x = a_x + a_w t_x, \quad w = a_w e^{t_w}$$

(same for y, h). Small numbers mean "small nudge".

Practitioner's Note

With a feature pyramid (FPN), Faster R-CNN is still a strong, accurate baseline, available in one line in torchvision.

Training the region proposal network

- Every anchor gets a label from its overlap with the ground truth:
  - IoU > 0.7: **object**;
  - IoU < 0.3: **background**;
  - in between: ignored.
- Each image, sample 256 anchors, up to half of them objects.
- Loss: object/background cross-entropy + box regression (only for object anchors).

Decoding one anchor

Anchor: centre $(100, 100)$, size 64×128. Network output: $t = (0.1,\ -0.05,\ \ln 1.25,\ 0)$.

$x = 100 + 64 \cdot 0.1 = 106.4$
$y = 100 + 128 \cdot (-0.05) = 93.6$
$w = 64 \cdot e^{\ln 1.25} = 80$,
$h = 128 \cdot e^{0} = 128$: the anchor moved slightly and got wider.

Intuition

Predicting *offsets* from a template is far easier than predicting absolute coordinates from scratch: the numbers stay small and similar for objects anywhere in the image.

Faster R-CNN on a real street: the two stages



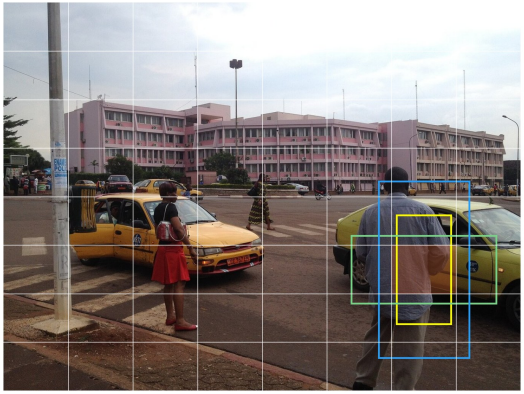
Happiraphael, CC BY-SA 4.0

Left: the region proposal network's first 300 of 1000 proposals: dense where objects are, sparse on the sky and walls. **Right**: the second stage keeps and refines 65 confident detections.

Intuition

Stage 1 asks "is there *something* here?", cheaply, everywhere. Stage 2 asks "*what* exactly, and where precisely?", carefully, for a few hundred candidates.

Anchors, intuitively



Bdx, CC BY-SA 4.0

- Divide the image into a grid (here 8×8; in practice the feature map, e.g. 50×75).
- Each cell owns a few **anchors**: template boxes of typical shapes: tall (people), wide (cars), large, small.
- For each anchor the network predicts: is there an object? which class? how to nudge the box?

How many anchors?

A 50×75 feature map with 9 anchors per cell: $50 \times 75 \times 9 = 33\,750$ candidate boxes, all scored in one pass.

The imbalance problem

Needles in a haystack

33 750 anchors per image (previous slide), about 10 objects.
At most a few dozen anchors overlap an object well; over 99.8% are background.

A detector that always says "background" is right 99.8% of the time, and useless.

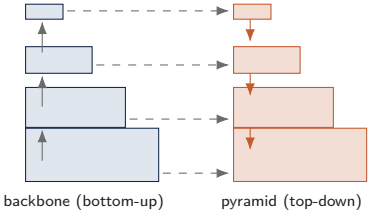
- **Two-stage** detectors handle it in stages: the RPN filters most background; the second stage samples a balanced batch.
- **One-stage** detectors face all anchors at once: the reason for focal loss (RetinaNet, later today).

Intuition

The pixel-accuracy trap of tomorrow's lecture, in a different form: when one class dominates, the loss and the metric must be designed so the rare class still counts.

Feature pyramids: objects of every size

- Deep feature maps are semantically strong but coarse (7×7 to 14×14): small objects vanish.
- Early maps are fine but semantically weak.
- **Feature Pyramid Network (FPN, 2017)**: pass deep information back down to the fine maps, then detect small objects on fine maps and large objects on coarse maps.



Intuition

The image pyramid of Week 1, Day 4, rebuilt inside the network, so that every level is both detailed *and* meaningful.

Part 4

One-stage detectors: YOLO and friends

YOLO (2016): you only look once

- Skip the proposal stage. One CNN pass predicts, for every grid cell, boxes, objectness and class probabilities directly.
- Original YOLO: 7×7 grid, 2 boxes per cell, 45 frames per second on a 2016 GPU: the first real-time deep detector.
- Detection becomes one big regression problem.

The size of YOLOv1's output

Each of 7×7 cells predicts 2 boxes $\times$ (4 coordinates $+$ 1 objectness) $+$ 20 class probabilities $= 30$ numbers.

Output tensor: $7 \times 7 \times 30 = 1470$ numbers, read off in one forward pass.

Intuition

Faster R-CNN: "first find candidates, then examine each". YOLO: "look once at the whole picture and write down everything".

What YOLO is trained to predict

- Each ground-truth object is assigned to the grid cell containing its centre (and, in later versions, to the best-matching anchor or nearby cells).
- That cell must predict: the box, high objectness, and the right class.
- All other cells must predict low objectness.

YOLO's loss, in words

box error (for cells with objects) + objectness error (all cells) + class error (cells with objects)

Intuition

Each cell is responsible for the objects whose centre falls inside it. Two small objects with centres in the same cell are the classic failure of early YOLO: flocks of birds, crowds far away. Finer grids and several scales fixed most of it.

The YOLO family and other one-stage detectors

- **SSD** (2016): predictions from several feature maps, i.e. several scales.
- **RetinaNet** (2017): *focal loss*: down-weights the thousands of easy background boxes so training focuses on hard examples. First one-stage detector to beat two-stage accuracy.
- **YOLOv3–v5, v8**: feature pyramids, better augmentation (mosaic), better training.
- **Anchor-free** designs (FCOS, YOLOX, YOLOv8+): predict distances from each point to the box sides; no hand-chosen anchor shapes.
- **YOLOv12** (2025): attention blocks inside a YOLO; the model we used on Week 1, Day 1.

Focal loss

$$\mathcal{L} = -(1 - p_t)^{\gamma} \log p_t$$

p_t: probability of the correct label; $\gamma \approx 2$.

An easy background box

$p_t = 0.99$: cross-entropy 0.010; focal ($\gamma = 2$): $0.0001 \times 0.010 = 10^{-6}$, ten thousand times smaller. Hard boxes keep almost their full loss.

Ten years of getting faster

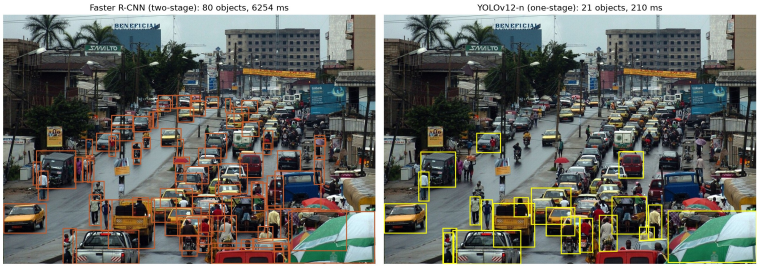
Detector	Year	Time per image	Main change
R-CNN	2014	about 47 s (GPU)	CNN on 2000 warped proposals
Fast R-CNN	2015	0.3 s + 2 s proposals	one CNN pass, RoI pooling
Faster R-CNN	2015	about 0.2 s (5 fps)	the network proposes regions
YOLO	2016	about 0.022 s (45 fps)	one pass, grid predictions

Times as reported in the original papers, on 2014–2016 GPUs.

Intuition

A 2000-fold speed-up in two years, mostly from removing repeated work: share computation across regions, then across stages. The same design question applies to any pipeline you build: what is being computed more than once?

Two-stage vs. one-stage on the same photograph



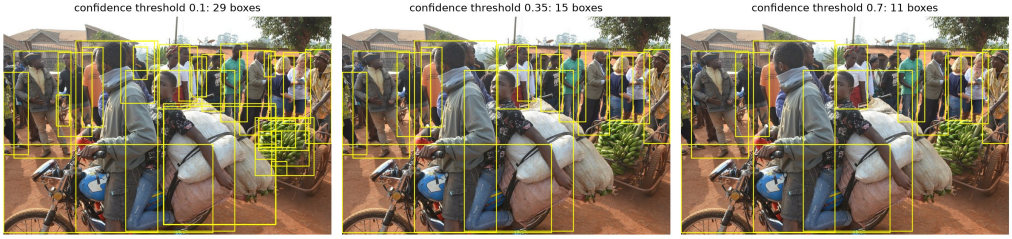
Happiraphael, CC BY-SA 4.0

On our laptop CPU: Faster R-CNN finds 80 objects in 6254 ms; YOLOv12-nano finds 21 in 210 ms. Both use COCO's 80 classes.

Intuition

Two-stage detectors spend more time per image. Early one-stage detectors traded accuracy for speed; modern ones (YOLOv8–v12) often match or beat Faster R-CNN, as our Penn-Fudan evaluation shows later today.

The confidence threshold is a decision, not a property of the model



Photokadaffi, CC BY-SA 4.0

YOLOv12-x on the Mbouda scene: at threshold 0.1, 29 boxes (more objects found, more false alarms); at 0.35, 15; at 0.7, 11 (only the surest).

Practitioner's Note

Choose the threshold for the *cost of each error*. Screening for malaria parasites: a missed parasite is worse than a false alarm, so use a low threshold. Counting traffic: balance both.

Real time, in numbers

The budget for live video

30 frames per second: $1000/30 \approx 33$ ms per frame for *everything*: reading the frame, resizing, the network, NMS, drawing, tracking.

On our laptop CPU, YOLOv12-n took about 210 ms on a large street photo; Faster R-CNN about 6254 ms.

- Smaller input images (e.g. 640 instead of 1280 pixels): about 4 times fewer pixels, much faster, but small objects vanish.
- Skip frames: detect every 5th frame, track in between.
- Use a GPU, or an accelerator on the device.

Practitioner's Note

State speed with its conditions: device, input size, batch size, and whether pre- and post-processing are included.

Augmenting detection data

- When you flip, crop or scale the image, the boxes must be transformed *the same way*.
- A crop can cut an object in half: keep it, drop it, or clip its box, depending on how much remains.
- **Mosaic** (YOLOv4+): stitch four training images into one; objects appear at new scales and in new contexts.

Horizontal flip

Image width $W = 640$. Box $(x_1, y_1, x_2, y_2) = (100, 50, 220, 300)$. After flipping: $(W - x_2,\ y_1,\ W - x_1,\ y_2) = (420, 50, 540, 300)$.

Practitioner's Note

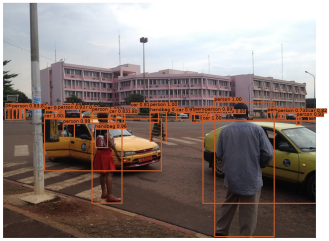
Libraries such as albumentations transform images and boxes (and masks) together. Always draw a few augmented samples with their boxes before training.

Part 5

Transformers for detection: DETR

DETR (2020): detection as set prediction

- A CNN backbone, then a transformer (Day 5).
- A fixed number of learned **object queries** (100): each query "asks" the image for one object and outputs a box and a class, or "no object".
- Training matches each ground-truth object to exactly one query (the Hungarian algorithm), so duplicates are penalised.
- **No anchors and no NMS.**



Bdx, CC BY-SA 4.0

Of DETR's 100 queries, 34 claim an object with confidence above 0.7; the rest say "nothing here".

Intuition

Anchors and NMS are hand-designed rules, the kind of engineering deep learning keeps replacing with learning. DETR learns not to produce duplicates.

Part 6

Measuring detectors honestly

Precision and recall

- A detection is a **true positive** (TP) if it matches an unmatched ground-truth box of the same class with IoU ≥ 0.5; otherwise a **false positive** (FP).
- Ground-truth objects nobody matched are **false negatives** (FN).

Precision and recall

$$\text{precision} = \frac{TP}{TP + FP}, \qquad \text{recall} = \frac{TP}{TP + FN}$$

One image

5 pedestrians. The detector outputs 6 boxes: 4 match pedestrians, 2 are on a lamp post and a bin.

TP $= 4$, FP $= 2$, FN $= 1$.
Precision $= 4/6 = 0.67$: two thirds of the boxes are right.
Recall $= 4/5 = 0.80$: four fifths of the people were found.

Intuition

Precision: "when it shouts, is it right?". Recall: "does it find everything?". Lowering the threshold raises recall and usually lowers precision.

The precision–recall curve and average precision

1. Sort all detections (over all images) by confidence.
2. Walk down the list; after each detection, record precision and recall so far.
3. Plot precision against recall.
4. **Average precision (AP)** = the area under this curve (COCO samples it at 101 recall levels).

Five detections, 3 objects

Sorted: TP, TP, FP, TP, FP.
After each: P $= 1, 1, 0.67, 0.75, 0.6$; R $= 0.33, 0.67, 0.67, 1, 1$.
Recall 1 reached with precision 0.75.

The numbers you will see in papers

AP50: AP with IoU threshold 0.5.
AP (COCO): AP averaged over IoU thresholds $0.50, 0.55, \ldots, 0.95$: rewards precise boxes.
mAP: the mean of AP over all classes.

Intuition

AP summarises the whole trade-off in one number, independent of any single threshold choice.

AP at different strictness levels and object sizes

- **AP50**: a box counts if IoU ≥ 0.5. Lenient about exact placement.
- **AP75**: IoU ≥ 0.75. Demands tight boxes.
- **AP** (COCO): the average over IoU from 0.50 to 0.95 in steps of 0.05.
- **AP_S, AP_M, AP_L**: objects smaller than 32^2 pixels, between 32^2 and 96^2, larger than 96^2.

Intuition

For counting cars, AP50 may be enough. For measuring a tumour or guiding a robot gripper, the tight-box AP75 matters. Choose the metric that matches the use.

Practitioner's Note

Small objects are where detectors are weakest: AP_S is often around half of AP_L. Parasites, distant pedestrians and seedlings in drone images are all "small".

From AP to mAP, worked

Three classes on a traffic dataset

AP(person) $= 0.82$, AP(car) $= 0.75$, AP(motorcycle) $= 0.55$.

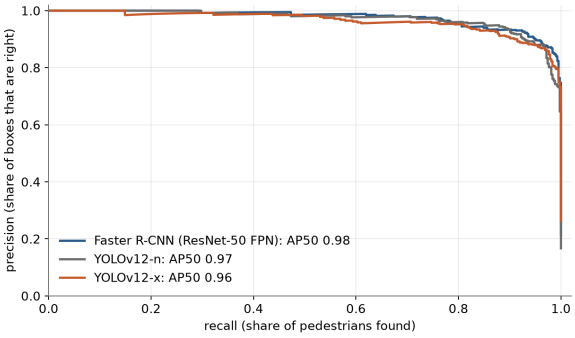
mAP $= (0.82 + 0.75 + 0.55)/3 = 0.707$.

- Every class counts equally, however rare.
- A high mAP can hide one class the system cannot do at all; always look at the per-class numbers.

Intuition

In Douala, motorcycles are everywhere; a traffic system that is weak on motorcycles is weak where it matters most. The average hides that.

A real evaluation: pedestrians in Penn-Fudan



170 street images with 423 annotated pedestrians, evaluated with our own code.

	AP50	AP	ms/img
Faster R-CNN	0.98	0.80	5747
YOLOv12-n	0.97	0.86	318
YOLOv12-x	0.96	0.90	4024

Times: median per image on a laptop CPU.

Practitioner's Note

Penn-Fudan does not label every person (distant or heavily hidden people are often unannotated), so some "false positives" are real people. Evaluation is only as good as the annotations: always look at the false positives yourself.

Reading the results

- All three find almost every pedestrian at IoU 0.5: AP50 is 0.97–0.98.
- The COCO-style AP, which demands tight boxes, separates them: Faster R-CNN (2015) 0.80, YOLOv12-n 0.86, YOLOv12-x 0.90. Precise localisation is the harder part, and nine years of progress shows there.
- The nano YOLO is both more accurate than Faster R-CNN here and about 18 times faster on CPU (318 vs. 5747 ms). The large YOLO buys a few more AP points for over ten times the nano's time.

Intuition

There is no single "best detector". There is a best detector *for a budget*: frames per second on your hardware, and which errors you can afford.

Practitioner's Note

Report AP, AP50, speed on the target device, and a few failure images. A table with only "mAP = 0.87" hides everything a user needs to know.

Counting with a detector

- Counting = run the detector, count the boxes above a threshold. Simple, and very useful: traffic, crowds, fruit, cells, livestock.
- Count error = (false positives) − (false negatives). The two can cancel and hide a poor detector.

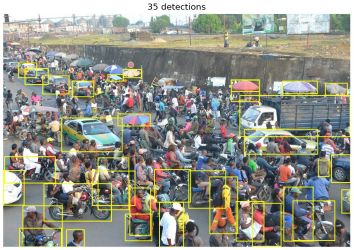
Week 1's Douala street

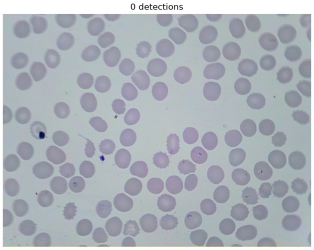
YOLOv12-x found 6 motorcycles and 8 people at threshold 0.35. In the full photo there are clearly more of both: partly hidden riders and distant people are missed. The count is a *lower bound*.

Practitioner's Note

To report counts, measure the counting error on a hand-counted sample of images from your setting. For dense crowds, density-map methods (predict "people per pixel" and integrate) often beat counting boxes.

Where pre-trained detectors fail





Photos: Minette Lontsie (CC BY-SA 4.0); OpenFlexure Project (CC BY-SA 4.0)

Left: a dense Douala crossroads: 35 detections, yet many partly hidden riders and distant people are missed. **Right**: a blood smear: 0 detections. COCO has no "parasite" or "red blood cell" class.

Intuition

A detector only knows the classes and the kinds of images it was trained on. For parasites, weeds, potholes or local vehicle types you must *fine-tune* on your own labelled boxes: Day 2's transfer learning, applied to detection.

Label formats you will meet

YOLO text: one `.txt` file per image, one line per object

```
# class  cx     cy     w      h    (all in
    [0,1])
0        0.125  0.219  0.125  0.250
3        0.610  0.540  0.220  0.180
```

Pascal VOC XML: one file per image, corners in pixels.

COCO JSON: one file for the whole dataset

```
{"images": [{"id": 7, "file_name": "a.jpg"
    ,
             "width": 640, "height":
    640}],
 "annotations": [{"image_id": 7, "
    category_id": 1,
             "bbox": [40, 60, 80, 160]}],
 "categories": [{"id": 1, "name": "person"
    }]}
```

Practitioner's Note

Labelling tools export all three. Choose the format your training code expects, and convert once with a tested script.

Training a detector on your own data

1. **Label boxes**: tools such as CVAT or Label Studio; a few hundred images per class is a start.
2. **Start from pre-trained weights** (COCO): the backbone and much of the head transfer.
3. **Replace the class layer** with your classes.
4. **Fine-tune** with augmentation (flips, scale, colour, mosaic).
5. **Evaluate** AP on images from new locations and days.

Ultralytics, in practice

```
model = YOLO("yolo12n.pt")
model.train(data="parasites.yaml",
epochs=50, imgsz=640)
```

A free Colab GPU trains a small YOLO on a few hundred images in well under an hour.

Intuition

Labelling boxes is slow and expensive. Draw them consistently: always the full object, including occluded parts, or never. Inconsistent boxes cap your AP.

From detection to tracking

- In video, we want *identities*: car #7 in frame 1 is car #7 in frame 2.
- **Tracking by detection**: detect in every frame, then link boxes across frames.
- **SORT** (2016): predict each track's next position with a simple motion model (Kalman filter), then match predictions to new detections by IoU.
- Later trackers (DeepSORT, ByteTrack) add appearance features and recover low-confidence boxes.

Intuition

Tracking turns a detector into a counter of *events*: vehicles that crossed a line, animals that entered a camera trap's view, cells that divided.

Practitioner's Note

Ultralytics: `model.track(source="video.mp4")`. Tracks break at occlusions: count how often identities switch, not just how many boxes are found.

Running detectors on small devices

- **Export** to a portable format: ONNX, TensorFlow Lite, CoreML.
- **Quantise**: store weights as 8-bit integers instead of 32-bit floats: about 4 times smaller and often 2–4 times faster, usually with a small accuracy loss.
- **Choose input size** for the smallest object you must see (Week 1, Day 2).

Model size

3 million parameters $\times$ 4 bytes $=$ 12 MB as float32; about 3 MB as int8.

Intuition

Offline, on-device detection matters where connectivity is poor or expensive: a phone in a field, a camera trap in a forest, a microscope in a rural clinic.

Practitioner's Note

Re-measure accuracy *after* quantisation on your own test set, on the target device. Small objects suffer first.

Detection and responsibility

- Person detection and tracking can count passengers at a bus station, or follow individuals without consent. Same technology, different use.
- Images of people, number plates and homes are personal data; many countries regulate how they may be collected and stored.
- Detectors can be less accurate for groups under-represented in their training data.

Questions to ask before deploying

- Who is in the images, and did they agree?
- What is stored, for how long, and who can see it?
- What happens to a person when the detector is wrong?
- Is accuracy similar across the groups affected?

Intuition

Week 3 returns to ethics and fairness in depth. Start asking these questions now, while you design your capstone.

Which detector, when?

Situation	Start with	Why
Live video, phone, drone	small YOLO (n or s)	real-time on modest hardware
Accuracy first, offline analysis	large YOLO, RT-DETR, or Faster R-CNN + FPN	precise boxes, small objects
Crowded scenes	detector + Soft-NMS or crowd-trained model	overlapping objects survive
New classes, few labels	fine-tuned small YOLO; open-vocabulary detector to pre-label (Day 5)	label efficiently, deploy fast
Need exact outlines	instance segmentation (Day 4)	boxes are not enough

Practitioner's Note

Whatever you choose, run the Penn-Fudan-style evaluation from today on *your* test images: AP, speed on your device, and a page of failure examples.

Detection in the world

Domain	Objects	What makes it hard
Health	parasites, cells, nodules, polyps	tiny objects, rare positives, expert labels
Agriculture	fruit, weeds, pests, lesions	occlusion by leaves, lighting, similar species
Transport	vehicles, people, plates, potholes	crowds, night, motion blur, real-time
Conservation	animals in camera traps or drone images	camouflage, night infrared, tiny animals from the air
Industry	missing or defective components	rare defects, need near-zero misses

Intuition

Every row is a capstone project waiting to happen, and in every row the hard part is the data: what to label, how, and how to test that the detector works somewhere new.

Part 7

Wrapping up

Key takeaways

1. Detection outputs a variable-length list of (box, class, score). Compare boxes with **IoU**; remove duplicates with **NMS**.
2. **Two-stage**: propose regions, then classify and refine (R-CNN $\rightarrow$ Fast $\rightarrow$ Faster R-CNN).
3. **One-stage**: predict everything from a grid in one pass (YOLO, SSD, RetinaNet).
4. **Anchors** and **feature pyramids** handle shapes and sizes; **DETR** learns to avoid duplicates without them.
5. Evaluate with **precision, recall, AP**; report speed and failures too.
6. Pre-trained detectors know only their classes; fine-tune on your own boxes.

Before the next lecture

Practise (tutorial with the tutors)

- Implement IoU, NMS and AP yourself; check them against `torchvision.ops`.
- Run Faster R-CNN and YOLO on your own photos; compare speed and errors.
- Fine-tune a small YOLO on a tiny labelled dataset on Colab.

Read

- Ren et al. (2015), *Faster R-CNN*
- Redmon et al. (2016), *You Only Look Once*
- Zhang et al., *Dive into Deep Learning*, §14.3–14.8

Next lecture: segmentation

- From boxes to pixels
- Fully convolutional networks and upsampling
- U-Net for medical images
- Mask R-CNN: instance segmentation
- Segment Anything: one model, any object

Credits

Penn-Fudan Pedestrian Database: L. Wang, J. Shi, G. Song, I. Shen (University of Pennsylvania). Faster R-CNN weights: torchvision (COCO). YOLOv12: Ultralytics / Tian et al. (2025). DETR: Facebook AI, via Hugging Face. All evaluations were run for this lecture with `Week2/tools/make_week2_figures.py`.

Photographs from Wikimedia Commons:
`douala_deido`: Happiraphael, CC BY-SA 4.0
`yaounde_taxis`: Bdx, CC BY-SA 4.0
`moto_mbouda`: Photokadaffi, CC BY-SA 4.0
`douala_ndokoti`: Minette Lontsie, CC BY-SA 4.0
`malaria_smear`: OpenFlexure Project, CC BY-SA 4.0