Detection & Segmentation

Lecture 9 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Learning outcomes

By the end of this lecture you will be able to:

1. Extend a classifier to **localization + detection** with multiple heads.
2. Compute **IoU** and explain **NMS**.
3. Contrast **R-CNN family vs YOLO vs DETR**.
4. Pick **anchor boxes** and explain delta parameterization.
5. Use a **U-Net** for per-pixel segmentation; choose appropriate loss.
6. Prompt **SAM** for zero-shot segmentation.

Recap · where we are

Module 4 so far:

- **L7** · CNN mechanics, receptive field, classic architectures
- **L8** · Modern CNNs (ResNet bottleneck, MobileNet, EfficientNet) + transfer learning

All of that was **classification** — one label per image.

> REFERENCE
>
> Today is different: UDL does NOT cover detection / segmentation. Read Bishop Ch 10 or CS231n OD notes alongside these slides.

Today's jump: **one label per pixel, per object, per region.**

Four questions

1. How do we go from classification to **bounding boxes**?
2. How do we compare boxes — what is **IoU**? How do we deduplicate predictions — **NMS**?
3. How does **YOLO** do all of it in one forward pass?
4. How do we predict a **mask per pixel** — **U-Net** and beyond?

PART 1

Classification → Localization → Detection

Three increasing levels of spatial specificity

The spectrum of "what's in this image"

TASK	OUTPUT	EXAMPLE
Classification	1 label	"cat"
Classification + localization	1 label + 1 bbox	"cat @ (60, 80, 200, 180)"
Object detection	many labels + many bboxes	"cat @ … dog @ … car @ …"
Semantic segmentation	label per pixel	every pixel → class (no instance)
Instance segmentation	label + instance per pixel	"cat 1 vs cat 2 as separate masks"

Today we cover the last four.

Training for two goals at once

INTUITION

Analogy · grading a two-section exam. Section 1 = multiple-choice (classification). Section 2 = a drawing (localization). Final score = MCQ + drawing.

If the drawing is more important: $\text{score} = \text{MCQ} + 2.0 \cdot \text{drawing}$. Our loss does the same — sum two losses, weight one with λ.

Classification + localization · the multi-task loss

Keep the CNN. Add **two heads**:

```python
class ClsLocHead(nn.Module):
    def __init__(self, feat_dim, n_classes):
        super().__init__()
        self.cls = nn.Linear(feat_dim, n_classes)   # class scores
        self.box = nn.Linear(feat_dim, 4)           # (x, y, w, h)
```

For one image:

- True class y, model predicts class logits.
- True box $\mathbf{b} = [x, y, w, h]$, model predicts $\hat{\mathbf{b}}$.

$$\mathcal{L} = \underbrace{\mathcal{L}_{\text{class}}(\text{logits}, y)}_{\text{cross-entropy}} + \lambda \cdot \underbrace{\mathcal{L}_{\text{box}}(\hat{\mathbf{b}}, \mathbf{b})}_{\text{Smooth L1 / MSE}}$$

λ is the **balancing knob**: $\lambda > 1$ → box matters more; $\lambda < 1$ → class matters more. Typical $\mathcal{L}_{\text{box}}$ is **Smooth L1** (L2 near 0, L1 far → robust).

Worked numeric · multi-task loss

One image of a cat. True class index 1. True box $[100, 120, 50, 80]$.

Model predicts:

- Class logits $[0.1, 2.5]$ → confidently "cat" → say $\mathcal{L}_{\text{class}} = 0.08$.
- Box $\hat{\mathbf{b}} = [102, 119, 55, 81]$.

Box loss (MSE).

$$\mathcal{L}_{\text{box}} = (102-100)^2 + (119-120)^2 + (55-50)^2 + (81-80)^2 = 4 + 1 + 25 + 1 = 31$$

Total ($\lambda = 0.1$).

$$\mathcal{L} = 0.08 + 0.1 \cdot 31 = 0.08 + 3.1 = \mathbf{3.18}$$

This `3.18` is what backprop sees → updates *both* the class head and the box head simultaneously.

PART 2

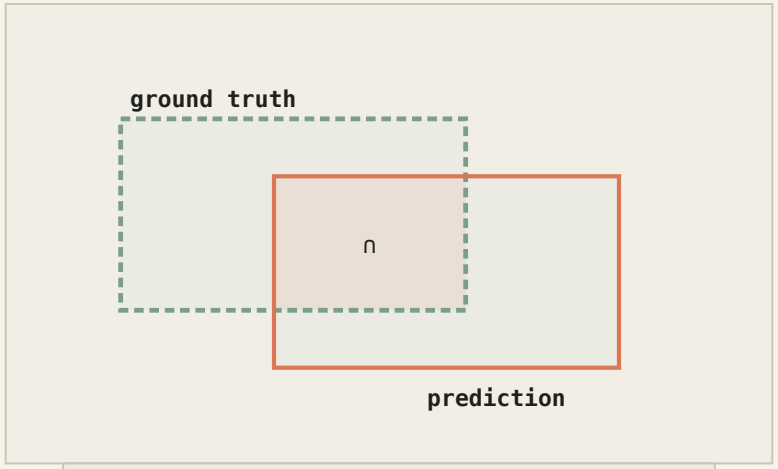
IoU and NMS

Two primitives every detector needs

IoU · the metric · NMS · the cleanup

IoU and NMS — the two building blocks every detector needs

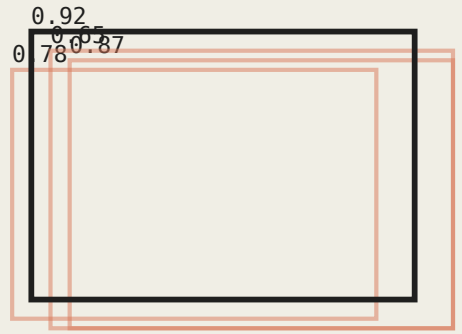
INTERSECTION OVER UNION




```
IoU = Area(∩) / Area(∪)
= 7000 / 25400 ≈ 0.28
```

typical threshold: 0.5 for detection

NON-MAX SUPPRESSION



keep: highest score (0.92)

```
① sort by score (desc)
② keep top, discard any overlap (IoU > 0.5)
③ repeat on remaining
```

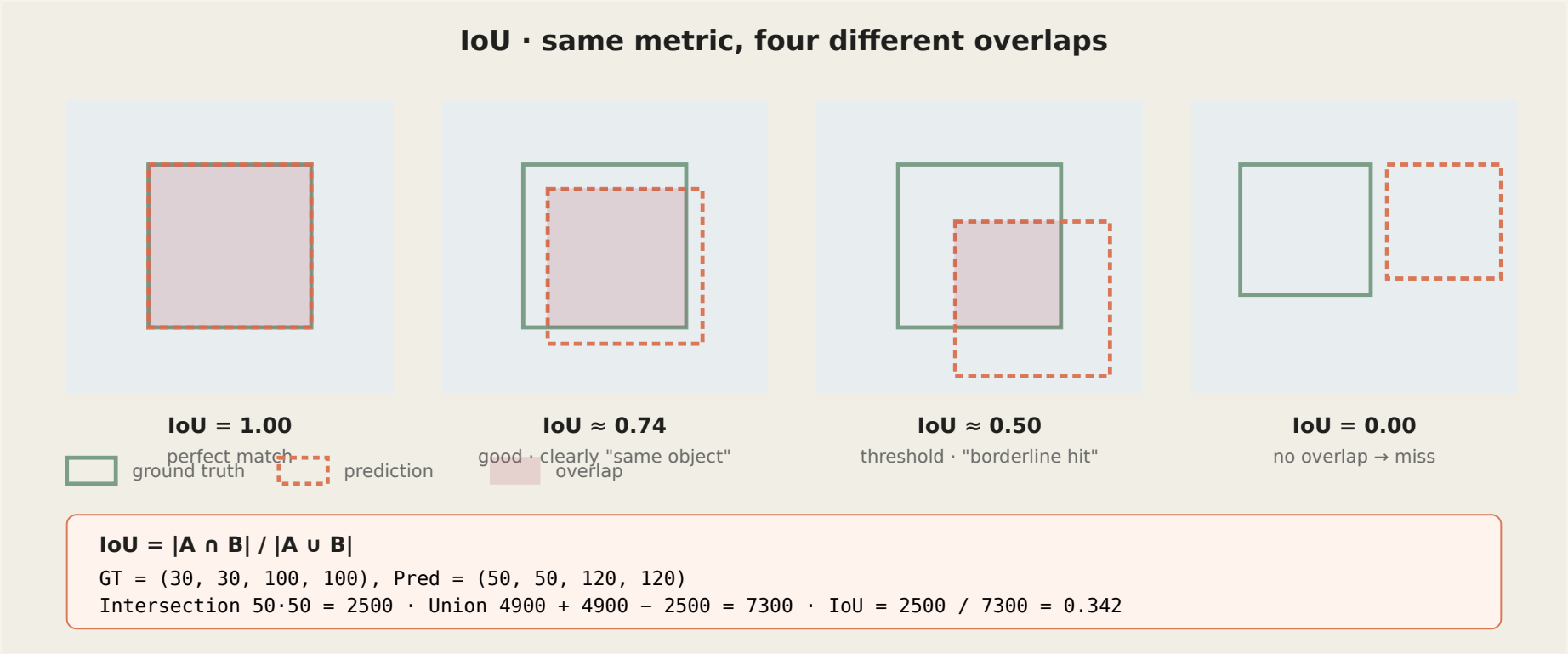
IoU measures bbox quality — NMS collapses redundant predictions on the same object. Together they turn hundreds of raw boxes into one clean detection per object.

IN PRACTICE

▶

Interactive: draw boxes on a canvas, see IoU + NMS live — object-detection.

IoU · visual examples



IoU = |A ∩ B| / |A ∪ B|

```
GT = (30, 30, 100, 100), Pred = (50, 50, 120, 120)
Intersection 50·50 = 2500 · Union 4900 + 4900 − 2500 = 7300 · IoU = 2500 / 7300 = 0.342
```

IoU · a quick worked example

Predicted box · $(x_1, y_1, x_2, y_2) = (50, 50, 150, 150)$ · area = 100 × 100 = 10,000.
Ground-truth box · $(100, 100, 200, 200)$ · area = 10,000.

Intersection · $(100, 100, 150, 150)$ · area = 50 × 50 = 2,500.
Union · 10,000 + 10,000 − 2,500 = 17,500.

$$\text{IoU} = 2{,}500/17{,}500 \approx 0.14$$

INTUITION

An IoU below 0.5 is almost always considered a miss. Detectors report mAP at multiple IoU thresholds because a box that's 90% right (IoU 0.9) is much more useful than one that's 60% right (IoU 0.6) — the metric penalizes imprecision.

NMS · the "shouting answers" analogy

KEY IDEA

After predictions you get a **pile of overlapping boxes** on the same object · like several people shouting the same answer.

NMS · keep the most confident shout · silence the others.

Concretely · sort by confidence, keep the best, discard any later box that overlaps it (IoU above threshold). Repeat until none left. Standard since R-CNN; survives in YOLO and most detectors.

NMS · step-by-step example

5 predicted boxes for one car. IoU threshold 0.5.

BOX	SCORE
A	0.95
B	0.90
C	0.80

Step 1 · pick A (highest score). Add to `keep`.
Compare IoU(A, others): IoU(A,B)=0.8 → drop B; IoU(A,C)=0.2 → keep; IoU(A,D)=0.7 → drop D; IoU(A,E)=0.1 → keep.
Pool now `[C, E]`.

Step 2 · pick C. Add to `keep`.
IoU(C, E) = 0.15 → keep E.
Pool now `[E]`.

Step 3 · pick E. Add to `keep`. Pool empty. Stop.

Final · **`keep = [A, C, E]`**. 5 boxes → 3 final detections.

NMS · pseudocode

```python
def nms(boxes, scores, iou_threshold=0.5):
    idx = scores.argsort(descending=True)
    keep = []
    while len(idx):
        i = idx[0]
        keep.append(i)
        # drop any later box that overlaps >threshold with this one
        idx = [j for j in idx[1:] if iou(boxes[i], boxes[j]) < iou_threshold]
    return keep
```

Greedy — highest-confidence box wins in its neighborhood. Every detector uses some form of NMS (Faster R-CNN, YOLO) or its learned replacement (DETR's Hungarian matching).

How do we grade a detector?

INTUITION

Analogy · search engine for "cat". You return 10 results.

- **Precision** = fraction that *actually* are cats. (8/10 → 80%.) *Of the answers you gave, how many were right?*
- **Recall** = fraction of all cat photos on the internet you found. (8 of 100 → 8%.) *Of all right answers, how many did you find?*

Trade-off · 100% recall = return everything; 100% precision = return only one sure answer. **mAP** summarizes the trade-off curve.

Computing AP · a worked example

3 ground-truth cats. 5 predictions, sorted by confidence. IoU > 0.5 → True Positive.

RANK	CONF	TP/FP	TP CUM	FP CUM	RECALL (TP/3)	PRECISION (TP/(TP+FP))
1	0.98	TP	1	0	0.33	1.00
2	0.95	TP	2	0	0.67	1.00
3	0.88	FP	2	1	0.67	0.67
4	0.75	TP	3	1	1.00	0.75
5	0.60	FP	3	2	1.00	0.60

AP = area under the precision–recall curve built from these points.
mAP = average AP over all classes.

mAP@0.5 = AP at IoU threshold 0.5. **mAP@[0.5:0.95]** = average over 10 IoU thresholds (COCO standard).

PART 3

One-stage vs two-stage detectors

R-CNN family · YOLO · DETR

R-CNN family · two-stage (brief)

R-CNN (2014) → Fast R-CNN (2015) → Faster R-CNN (2015)

Two-stage idea:

1. **Propose** candidate regions (where might objects be?).
2. **Classify** each region (what's in it?).

Evolution:

	HOW REGIONS PROPOSED	INFERENCE TIME
R-CNN	selective search (outside CNN)	~50 s
Fast R-CNN	selective search + shared backbone	~2 s
Faster R-CNN	**Region Proposal Network (RPN)** inside CNN	~0.1 s

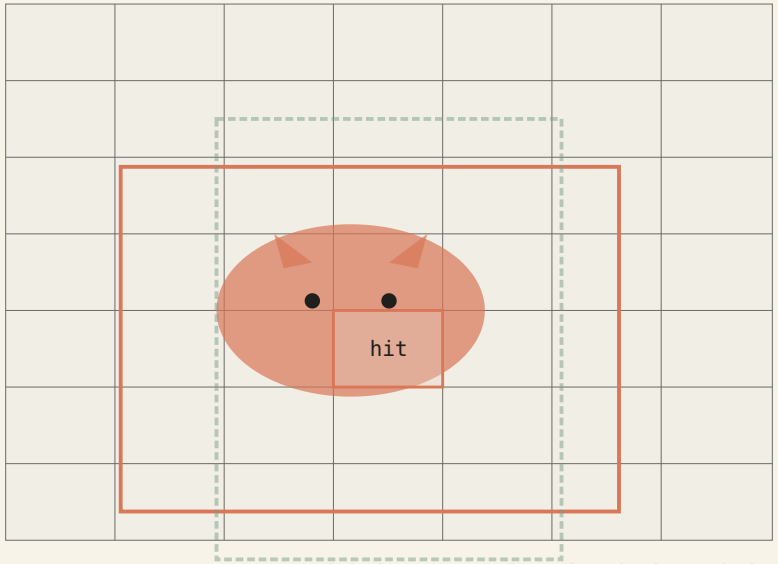
INTUITION

Two-stage detectors still win on accuracy for small objects. They are slower and more complex — often replaced by YOLO in production.

YOLO · you only look once

YOLO — one forward pass predicts every box on an S×S grid

S × S GRID · S = 7



WHAT EACH CELL OUTPUTS

For each of B anchors:

(x, y) — offset within cell

(w, h) — box width / height

confidence — is there any object?

Plus for the whole cell:

p(cat) · p(dog) · p(car) · ...
= K class probabilities

Total output shape

S × S × (B · 5 + K)

e.g. 7 × 7 × (2 · 5 + 20) = 7 × 7 × 30

…predicted in ONE forward pass

YOLO · "you only look once" — reshape detection into a single regression problem on a grid. Real-time inference, all modern versions (v3–v11).

YOLO · the grid-of-responsibilities idea

Divide the image into a 7×7 grid. Each cell fills out a **form** with three questions:

1. **Is there an object centred here?** (yes/no + confidence) → **objectness loss** ($\mathcal{L}_{\text{obj}}$)
2. **If yes, where exactly?** (4 numbers x, y, w, h) → **box loss** ($\mathcal{L}_{\text{box}}$)
3. **If yes, what class?** (cat / dog / car…) → **class loss** ($\mathcal{L}_{\text{class}}$)

YOLO's total loss = sum of all three across every cell:

$$\mathcal{L} = \lambda_{\text{coord}} \sum \mathcal{L}_{\text{box}} + \sum \mathcal{L}_{\text{obj}} + \sum \mathcal{L}_{\text{class}}$$

- **Box** active **only** if a cell contains an object. MSE on $(x, y, \sqrt{w}, \sqrt{h})$ (sqrt so small boxes matter).
- **Objectness** active **always** — BCE pushes toward 1 if object, 0 if not.
- **Class** active **only** if cell contains an object — cross-entropy.

Worked numeric · YOLO loss for one cell

Cell holds a **dog** (class index 2).

Ground truth. Box $\mathbf{b} = [0.5, 0.5, 0.2, 0.3]$. Objectness 1. Class one-hot $[0, 0, 1, 0, \ldots]$.

Prediction. $\hat{\mathbf{b}} = [0.6, 0.4, 0.25, 0.31]$. Objectness 0.85. Probs $[0.1, 0.2, 0.6, 0.1, \ldots]$.

Components.

- $\mathcal{L}_{\text{box}} = (0.6 - 0.5)^2 + (0.4 - 0.5)^2 + (0.25 - 0.2)^2 + (0.31 - 0.3)^2 = 0.01 + 0.01 + 0.0025 + 0.0001 \approx 0.02$
- $\mathcal{L}_{\text{obj}} = -\log(0.85) \approx 0.16$
- $\mathcal{L}_{\text{class}} = -\log(0.6) \approx 0.51$

Total per cell (no weighting): $0.023 + 0.16 + 0.51 \approx \mathbf{0.69}$. Sum across all cells = full image loss.

Why predict deltas · relative directions

INTUITION

Analogy · giving directions.

- *Absolute*: "Go to GPS coordinates (40.7128, -74.0060)." Hard. Different from every starting point.
- *Relative (deltas)*: "50 m forward, 10 m right." Easy. Works from any starting corner.

Anchor boxes are starting corners. Conv layers are translation-equivariant — the **same filter** runs at every spatial location, so it should predict the **same correction style** everywhere, not different absolute coordinates.

Decoding the box prediction · with example

Anchor (x_a, y_a, w_a, h_a) is known per cell. Network predicts deltas (t_x, t_y, t_w, t_h).

Centre — small shift, scaled by anchor size:
$b_x = x_a + t_x \cdot w_a, \quad b_y = y_a + t_y \cdot h_a$

Size — log-space correction; exp keeps width/height positive:
$b_w = w_a \cdot \exp(t_w), \quad b_h = h_a \cdot \exp(t_h)$
(If $t_w = 0$, $\exp(0) = 1$ → width unchanged.)

Worked numeric. Anchor $(120, 240, 80, 100)$, deltas $(0.1, -0.2, 0.3, -0.1)$.

- $b_x = 120 + 0.1 \cdot 80 = 128$
- $b_y = 240 - 0.2 \cdot 100 = 220$
- $b_w = 80 \cdot e^{0.3} \approx 80 \cdot 1.35 = 108$
- $b_h = 100 \cdot e^{-0.1} \approx 100 \cdot 0.90 = 90$

Final box · $(128, 220, 108, 90)$. Network only had to learn small corrections, anchor did the heavy lifting.

Speed vs accuracy · a detector comparison

DETECTOR	MAP (COCO)	FPS (V100)	NOTES
Faster R-CNN	42	~5	accuracy king, slow
YOLOv8-m	50	~250	production default
DETR	44	~30	elegant, data-hungry
RT-DETR	53	~100	real-time Transformer-based

INTUITION

Choose by constraint · real-time camera feed → YOLO. Labeled-data poor → DETR with good augmentations. Highest accuracy → large backbone + Faster R-CNN variant. There is no universally-best detector.

The YOLO lineage

VERSION	YEAR	KEY CONTRIBUTION
YOLOv1	2015	grid formulation, one shot
YOLOv3	2018	multi-scale predictions, anchor clustering
YOLOv5	2020	mosaic augmentation, practical toolkit
YOLOv8	2023	anchor-free, efficient
YOLOv11	**2024**	current production default

IN PRACTICE

For any real-time detection task in 2026, start with `ultralytics` YOLOv11. `pip install ultralytics` → model downloads + runs in 10 lines.

DETR · ditch the post-processing

YOLO and Faster R-CNN are conceptually two-step: predict **densely** (thousands of boxes), then **clean up** with NMS.

DETR's question · can we just directly predict the final, clean set?

It outputs a **fixed-size set** of 100 predictions. No grid, no anchors, no NMS.

Challenge · the model outputs 100 boxes; the ground truth might have only 3. Prediction order is arbitrary — pred #47 might match ground truth #1.

> INTUITION
>
> **Solution · Hungarian matching.** Imagine 3 tasks (the GT objects) and 100 workers (predictions). Assign one worker per task to minimize total cost. The Hungarian algorithm finds this optimal one-to-one matching. Once matched, compute loss on the matched pairs; the other 97 are matched to "no object."

DETR cleans up detection conceptually but is slower and data-hungry. YOLO still wins speed; DETR wins elegance.

PART 4

Semantic segmentation · U-Net

Pixel-level classification

From detection to segmentation

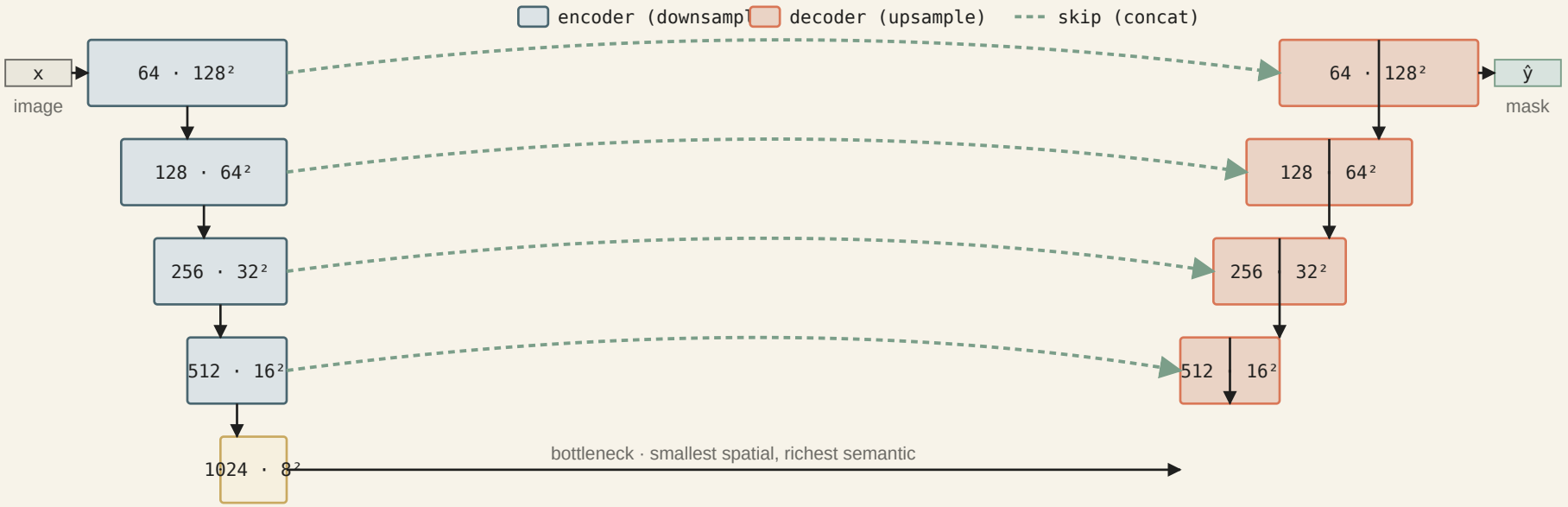
Detection gives boxes around objects. Segmentation gives a **label per pixel**.

Key architectural change: we need to go *back up* in spatial resolution — the feature map shrinks through convs/pooling, but the output must match the input size.

Solution: encoder-decoder with upsampling.

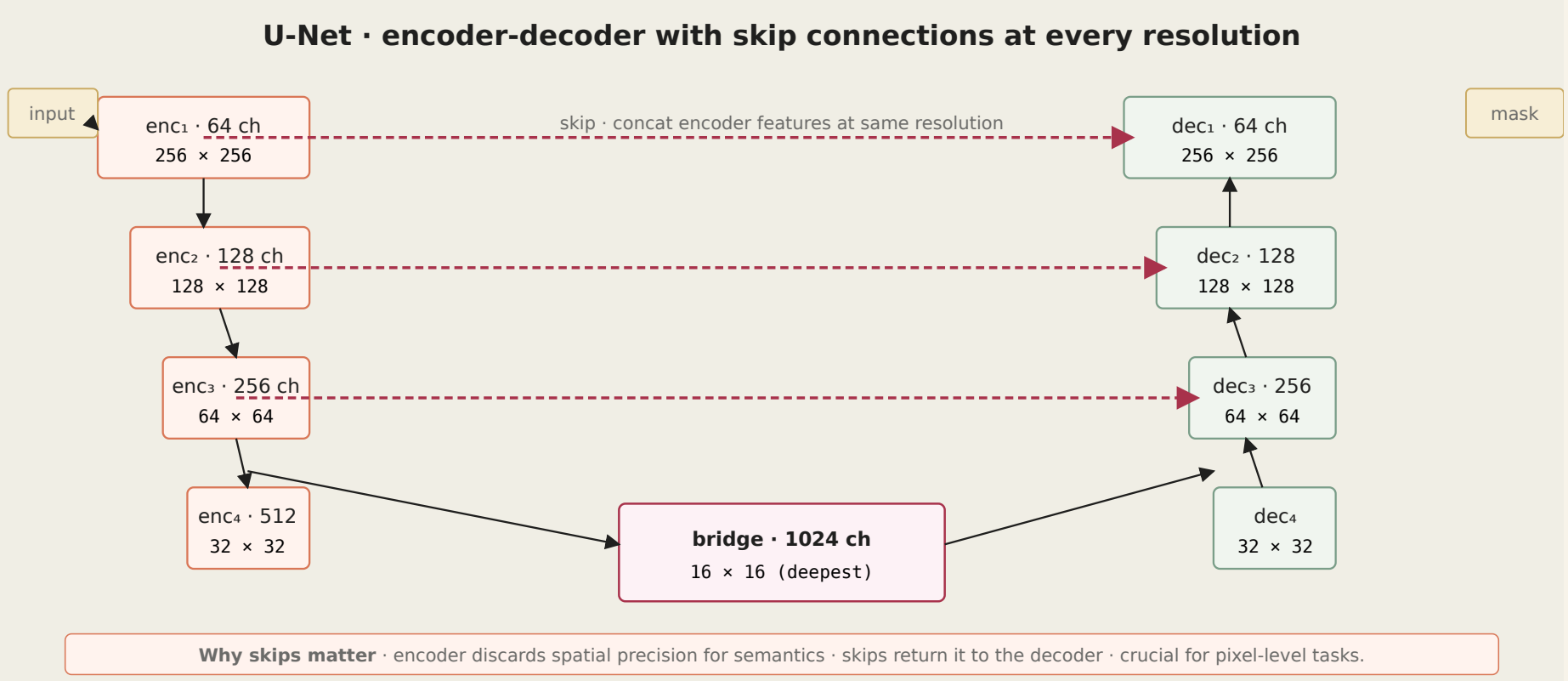
U-Net architecture

U-Net — encoder-decoder with skip connections · the standard for segmentation



*Ronneberger 2015 · skip connections **bring back fine spatial detail** that downsampling lost. Still the default for medical, satellite, and diffusion U-Nets (L21–22).*

U-Net · with channels and sizes



IN PRACTICE

▶

Interactive: click an image region, see segmentation fill in — image-segmentation.

U-Net skips · the cheat sheet analogy

KEY IDEA

Think of the encoder as creating a **rich but blurry summary** of the image · "I see a kidney here, a vessel there" · but exact edges have been smudged by pooling.

The skip connections give the decoder a **cheat sheet** from the matching-resolution encoder layer · the original crisp pixel detail.

Result · the decoder uses high-level summary for *what* and the cheat sheet for *exactly where the boundary is*. Net effect · sharp accurate masks.

Why skip connections are essential

The encoder compresses spatial info into richer features. But **spatial precision is lost** — a 16×16 feature map can't localize edges accurately.

Skip connections let the decoder *concatenate* encoder features at matching resolution:

```
def forward(self, x):
    e1 = self.enc1(x)         # 128×128
    e2 = self.enc2(pool(e1))  #  64×64
    e3 = self.enc3(pool(e2))  #  32×32
    b  = self.bridge(pool(e3))

    d3 = self.dec3(cat([up(b),  e3]))    # ← skip from e3
    d2 = self.dec2(cat([up(d3), e2]))    # ← skip from e2
    d1 = self.dec1(cat([up(d2), e1]))    # ← skip from e1
    return self.final(d1)
```

Every modern segmentation net (DeepLab, SegFormer) uses this pattern.

Segmentation loss · IoU to Dice

For boxes we used **IoU**. For masks we can do the same:

$$\text{IoU} = \frac{|P \cap T|}{|P \cup T|}$$

Good metric — but its gradient is "sharp," tricky for SGD. The **Dice coefficient** is the smoother cousin:

$$\text{Dice} = \frac{2\,|P \cap T|}{|P| + |T|}$$

Optimizers minimize, so:

$$\mathcal{L}_{\text{Dice}} = 1 - \text{Dice}$$

Perfect prediction → Dice 1 → loss 0. No overlap → loss 1.

Why it handles imbalance. It only counts pixels in P or T — the vast background of true negatives doesn't dominate (unlike plain CE).

Worked numeric · Dice loss on a 2×2 image

Task · segment the top-left pixel.

Ground truth $T = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, prediction (probabilities) $P = \begin{pmatrix} 0.9 & 0.2 \\ 0.1 & 0.3 \end{pmatrix}$.

1. Intersection $|P \cap T| = \sum P \odot T = 0.9 \cdot 1 + 0.2 \cdot 0 + 0.1 \cdot 0 + 0.3 \cdot 0 = \mathbf{0.9}$
2. $|P| = 0.9 + 0.2 + 0.1 + 0.3 = 1.5$
3. $|T| = 1.0$
4. $\text{Dice} = (2 \cdot 0.9)/(1.5 + 1.0) = 1.8/2.5 = \mathbf{0.72}$
5. $\mathcal{L}_{\text{Dice}} = 1 - 0.72 = \mathbf{0.28}$

The model backprops this 0.28.

Other segmentation losses

- **Pixel-wise cross-entropy** · default. Per-pixel softmax over C classes.
- **Focal loss** · down-weights easy pixels → model focuses on boundaries.
- **Boundary loss** · explicit pixel accuracy near edges.

WATCH OUT

Class imbalance is the #1 issue. A medical image with 99% background pixels optimizes to "predict background always" under plain CE. Reach for Dice (or weighted CE) first.

Why U-Net caught on in medical imaging

Ronneberger 2015 targeted electron microscopy cell segmentation. Two reasons it spread:

1. **Small data tolerance** · medical datasets are in the hundreds. U-Net's strong spatial prior (encoder-decoder + skip) makes this workable.
2. **Permissive license + simple architecture** · anyone could port it to any framework in a day.

   By 2020, U-Net was the default segmentation network not just in medicine but in satellite imagery, materials science, audio-spectrogram analysis, and later **diffusion models** (L21 / L22) — where the same encoder-decoder-with-skips handles noise-to-image mapping.

Instance segmentation · Mask R-CNN (brief)

Built on Faster R-CNN. Adds a third head:

- **Class** head (from Faster R-CNN)
- **Bbox** head (from Faster R-CNN)
- **Mask** head — a small FCN producing a pixel mask per region

> REFERENCE
>
> He et al. 2017 · Mask R-CNN — cleanly combines detection and segmentation. Standard baseline for instance tasks.

PART 5

2026 frontier · SAM

Zero-shot segmentation by prompting

Segment Anything · SAM (Meta, 2023)

A **foundation model for segmentation**:

- Trained on 1.1 billion masks across 11M images.
- Takes a prompt — a point, a box, or text — and returns a mask.
- **Zero-shot** on unseen categories and domains.

KEY IDEA

SAM changed segmentation the way CLIP changed classification — you don't need to train for your specific dataset; you just prompt a pretrained model.

IN PRACTICE

In 2026: for most segmentation tasks, start with SAM-2 and fine-tune only if the domain is truly specialized (medical, satellite).

The fixed-dictionary problem

INTUITION

Traditional detectors are like a translator with a fixed dictionary. Train on "cat / dog / car" → it can only ever detect those 3 things. Ask for "bicycle" → *"sorry, not in my dictionary."*

The open-vocabulary goal · build a **universal translator** that understands concepts, not fixed labels.

The open-vocabulary shift · how it works

Embedding · a vector that represents data. Image embedding = vector representing an image; text embedding = vector representing text.

CLIP · OpenAI 2021. Trained on millions of (image, caption) pairs. Learns a **shared embedding space**: the vector for a dog photo lands close to the vector for "a photo of a dog"; both far from "a photo of a cat."

How open-vocab detectors use this:

1. User provides a text prompt — e.g. "a red bicycle".
2. Model computes the **text embedding** via CLIP's text encoder.
3. Model processes the image with a CNN → spatial features.
4. **Search** the image for regions whose embeddings are close to the text embedding.
5. Match → draw a box.
   Examples · OWLv2, GroundingDINO, SAM-with-text. The 2024–2026 frontier is **fully prompt-driven** vision.

Summary · Lecture 9 — summary

- **Detection** = classification + localization at multiple objects.
- **IoU** quantifies bbox quality; **NMS** deduplicates; **mAP** is the headline metric.
- **R-CNN family** — propose regions then classify (two-stage, accurate, slow).
- **YOLO** — one forward pass on a grid (one-stage, fast, production standard).
- **DETR** — Transformer-based set prediction; no NMS.
- **U-Net** — encoder-decoder with skip connections · canonical for segmentation.
- **SAM** — zero-shot segmentation via prompting (2023+).

Read before Lecture 10

Bishop Ch 12 · sequences and recurrence.

Next lecture

RNNs, LSTMs, GRUs — why MLPs fail on sequences, BPTT, gating mechanisms.

NOTEBOOK

Notebook 9 · `09-yolo-unet.ipynb` — run pretrained YOLOv11 on sample images; train a small U-Net on a toy segmentation task; measure IoU per class.