

Object Detection and Segmentation

Donald Shenaj

donald.shenaj@di.unipi.it

Oct 28, 2026

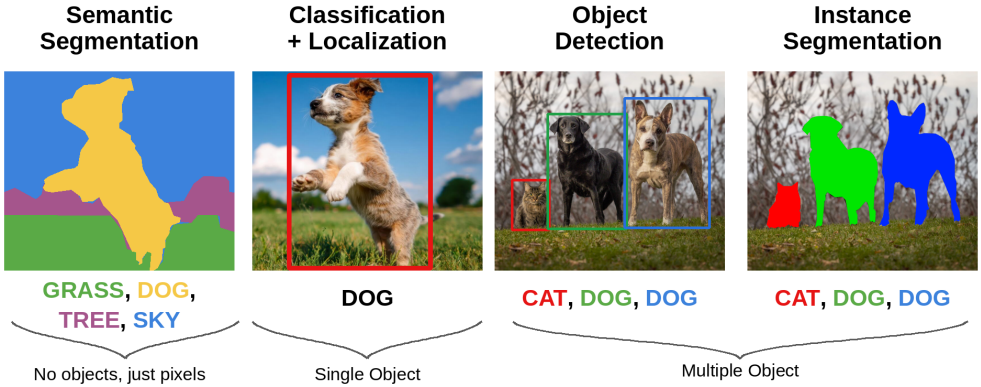
Lecture Outline

1. Object Detection
2. Advanced Architectures for Semantic Segmentation
3. Instance and Panoptic Segmentation

Recap

- Last lecture: networks that predict **spatially structured** outputs: one label per pixel (FCN, U-Net), with a **fixed-size** output grid matching the input resolution; also recall the semantic/instance/panoptic task *definitions* and the **mIoU** metric
- **Today's question**: classification gives one label and segmentation gives one label per pixel. How does a network predict a *separate* box and class for each object when the **number of objects isn't known in advance**?
- Object detection → one-stage vs. two-stage → NMS → DETR → evaluation; Advanced segmentation: (DeepLab, ViT-based decoders) → **instance** segmentation (reusing the detection tools) → **panoptic** segmentation unifies both

Classification, Detection, and Segmentation



- Same image, three different output shapes: one label, a set of boxes, or a per-pixel map

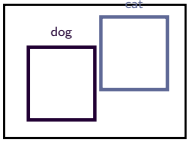
Classification, Detection, and Segmentation

Input: image $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$, classes $c \in \{1, \dots, C\}$, network f_θ

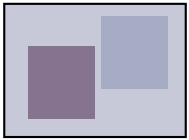
Task	**Output** $f_\theta(\mathbf{x})$	**Size**
Semantic seg.	$\hat{\mathbf{y}}_{h,w} \in [0,1]^C$ for every pixel (h, w)	$H \times W \times C$ fixed
Classif. + loc.	$(\hat{\mathbf{y}}, \hat{\mathbf{b}})$, $\hat{\mathbf{b}} = (x, y, w, h)$	$C + 4$ fixed
Detection	$\{(\hat{\mathbf{y}}_i, \hat{\mathbf{b}}_i)\}_{i=1}^{\hat{N}}$	$\hat{N}$ unknown set
Instance seg.	$\{(\hat{\mathbf{y}}_i, \hat{\mathbf{b}}_i, \hat{\mathbf{M}}_i)\}_{i=1}^{\hat{N}}$, $\hat{\mathbf{M}}_i \in [0,1]^{H \times W}$	$\hat{N}$ unknown set

- Fixed-size output $\Rightarrow$ each output unit has an obvious target $\Rightarrow$ plain CE / regression
- Set output $\Rightarrow$ *how many* boxes, and *which* prediction is matched to *which* object?

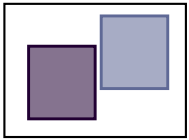
What Supervision Does Each Task Require?



Detection
box + class label per object



Semantic seg.
one class *per pixel*, no object identity



Instance seg.
one mask *per object*, each with a class

Panoptic segmentation combines both: per-pixel class for *stuff*, per-instance mask for *things*

$$\mathcal{D}_{\text{det}} = \{(\mathbf{x}_n, \{(\mathbf{b}_i, c_i)\}_{i=1}^{N_n})\}_{n=1}^{M} \qquad \mathcal{D}_{\text{sem}} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{M}$$

$$\mathcal{D}_{\text{inst}} = \{(\mathbf{x}_n, \{(\mathbf{b}_i, c_i, \mathbf{M}_i)\}_{i=1}^{N_n})\}_{n=1}^{M}$$

Different prediction tasks require different annotation structures

Major Datasets for Detection and Segmentation





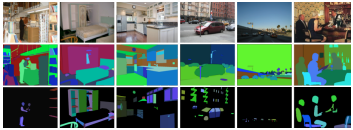
PASCAL VOC: 20 classes; boxes for every image, masks for a labeled subset



MS COCO: 80 "thing" classes (+53 "stuff"); boxes, instance & panoptic masks



Cityscapes: 19 evaluation classes; dense urban semantic + instance labels



ADE20K: 150 benchmark classes; dense semantic (+ instance) masks

[*Everingham et al., 2010: PASCAL VOC; Lin et al., 2014: COCO; Cordts et al., 2016: Cityscapes; Zhou et al., 2017: ADE20K*]

Object Detection

From region proposals to direct and set-based prediction

Object Detection: Problem Definition

- Predict a **set** of bounding boxes, each with a class label and confidence score
- A detector's output is a **variable-length set**: the network must decide *how many* objects there are and where they are
- Two broad families:
  - **Two-stage**: propose candidate regions, then classify/refine
  - **One-stage**: predict boxes directly, densely, in one pass

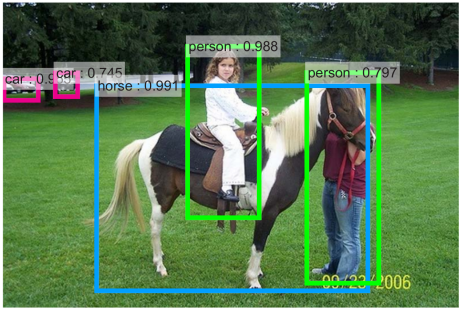


Figure: [*Ren et al., 2015: Faster R-CNN, qualitative result*]

Object Detection: One-Stage vs. Two-Stage

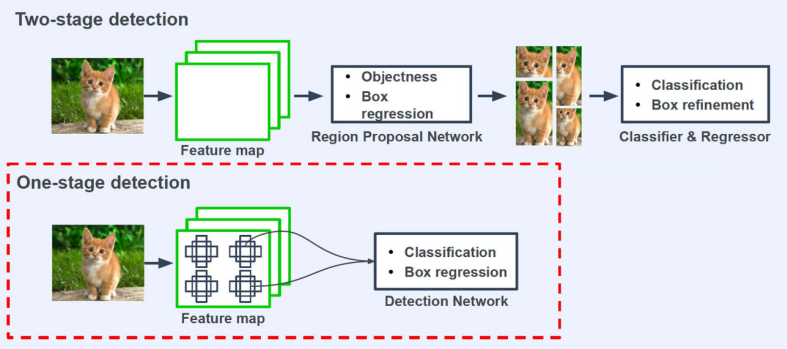


Figure: Adapted from [*Raza et al., 2025*]

- Both families solve the same set-prediction problem, they differ in whether candidate regions are proposed explicitly before classification
- **Supervision**: both need the same ground truth, a set of (box, class) pairs per image

Sliding Window Detection

- Early detectors used a **sliding window** approach: scan the image with a fixed-size window, classify each window as object vs. background
- To detect objects of different sizes, repeat at multiple scales (image pyramid)
- $\rightarrow$ too many windows $\sim 10^5$–10^6 per image, almost all background

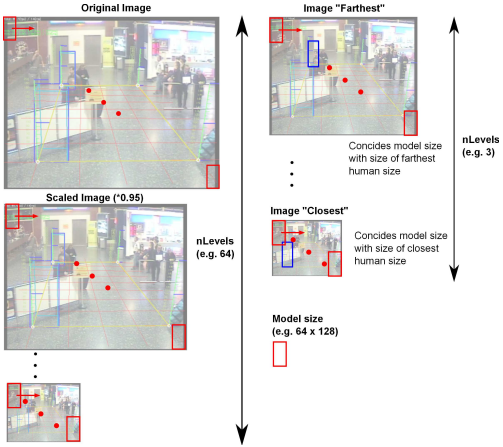


Figure: [*Nieto et al., 2014*]

Object Detection Before Deep Learning: Viola-Jones

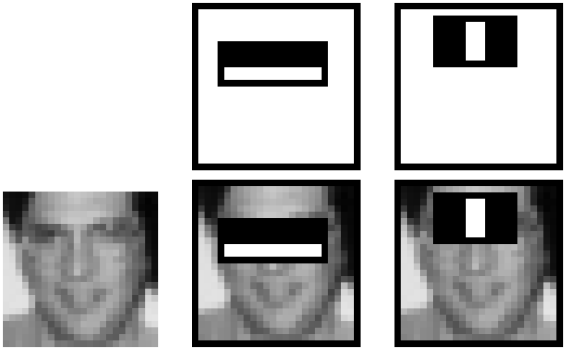
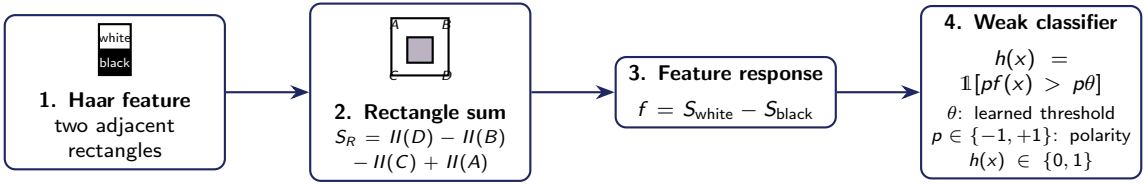


Figure: [*Viola & Jones, 2001, Fig. 3*]

- Simple **Haar-like** features: rectangles comparing weighted sums of pixel intensities between adjacent regions (e.g. dark eyes vs. bright cheeks), computed via the integral image (next slide)
- **AdaBoost** selects a small set of **weak classifiers** (one Haar feature + threshold each) from a huge fixed pool, and learns a combination weight for each selected one
- **Supervision**: supervised training on labeled face / non-face patches; no bounding-box regression
- Classifier stages are chained into a **cascade**: early stages are cheap and reject most background windows immediately, so only survivors reach later, stricter, costlier stages

Viola-Jones: A Haar Feature



- Integral image is the cumulative sum of *raw pixel intensities*, computed once per image: $II(x, y) = \sum_{x' \leq x,\, y' \leq y} I(x', y')$
- S_R is a plain sum of pixel intensities in rectangle R, computed from the integral image
- AdaBoost combines hundreds of such weak classifiers into one strong classifier (appendix). [*Viola & Jones, 2001*]

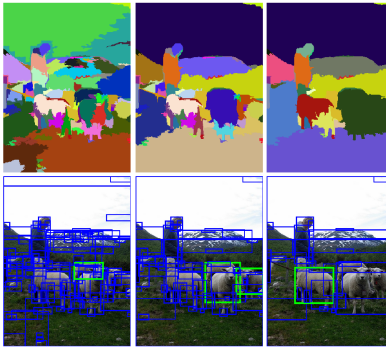
Two-Stage Detectors

- **Two-stage**: propose a sparse set of candidate regions, *then* classify and refine each one
- Each addresses key limitations of its predecessor

	Proposals	Training steps	Fixes
R-CNN	Selective Search	3, separate	/
Fast R-CNN	Selective Search	1, joint	no repeated CNN pass, no separate SVMs
Faster R-CNN	**RPN** (learned)	4 (orig.) / 1 (modern)	no external, fixed proposals

Region Proposals: Selective Search

- **1. Small, coherent regions**: oversegment the image (superpixels)
- **2. Merge by similarity**: repeatedly merge the most similar *adjacent* pair of regions, ranked by a similarity score $s(r_i, r_j)$
- **3. Extract boxes**: take the bounding box of every region in the resulting merge hierarchy, $\sim$2000 proposals per image
- Unsupervised and bottom-up: a fixed algorithm, not learned from data; Faster R-CNN later replaces it with a *learned* RPN



[*Uijlings et al., 2013, Fig. 2a*]

Definition (Region proposal). a candidate rectangular region likely to contain an object, scored and refined later instead of scanning every possible window.

Selective Search



Color

cats differ in color, same furry texture



Texture

chameleon camouflaged in color, differs in texture



Size

no single scale captures bowl, salad, *and* table



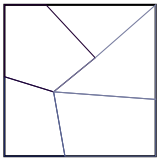
Fill

wheels enclosed by car despite differing in both

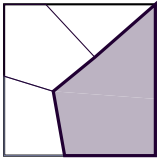
$$s(r_i, r_j) = a_1 s_{\text{color}} + a_2 s_{\text{texture}} + a_3 s_{\text{size}} + a_4 s_{\text{fill}}, \quad s_\bullet \in [0, 1]$$

- **Color/texture**: each region holds an L1-normalized histogram (c_i^k: fraction of region r_i's pixels in color bin k; t_i^k: same for a texture bin, e.g. Gaussian-derivative responses), compared by histogram intersection: $s_{\text{color}}(r_i, r_j) = \sum_k \min(c_i^k, c_j^k)$ (same form for texture, with t_i^k, t_j^k)
- **Size**: $s_{\text{size}}(r_i, r_j) = 1 - \frac{\text{size}(r_i)+\text{size}(r_j)}{\text{size(im)}}$, favors merging small regions first
- **Fill**: $s_{\text{fill}}(r_i, r_j) = 1 - \frac{\text{size}(BB_{ij})-\text{size}(r_i)-\text{size}(r_j)}{\text{size(im)}}$, BB_{ij}: tight box around $r_i \cup r_j$; rewards compact unions

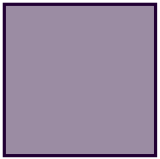
Selective Search: The Merge Algorithm



1. Oversegment: superpixels from a fixed, unsupervised algorithm



2. Merge: the most similar *adjacent* pair (shaded), by color/texture/size/fill

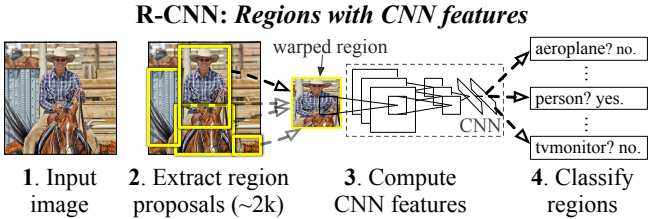


3. Repeat until one region remains: every intermediate region $\rightarrow$ one proposal

- At each step, merge the single highest-scoring *adjacent* pair (similarity $s(\cdot, \cdot)$), recompute its similarities, repeat
- Each merged region at each step becomes a proposal: the hierarchy yields $\sim$2000 boxes spanning all scales, from single superpixels up to the whole image

Full pseudocode: appendix. [*Uijlings et al., 2013, Algorithm 1*]

R-CNN: Regions with CNN Features



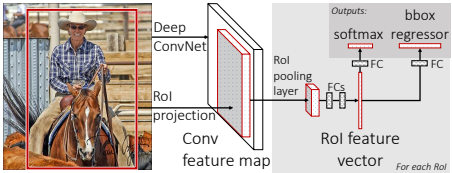
[*Girshick et al., 2014: R-CNN*]

- **Fixes**: replaces hand-crafted features (HOG, Haar) with features *learned* by a CNN
- **Test time**: one linear **SVM** per class c on the frozen CNN feature $\phi(r) \in \mathbb{R}^{4096}$ of warped region r, score $s_c(r) = w_c^\top \phi(r)$; ~2000 independent forward passes per image, no shared computation

Training, 3 separate steps, on the same warped region crops:

1. Fine-tune the CNN as a $(C+1)$-way **softmax classifier** (object classes + background, labels from IoU with ground truth); this softmax head is then **discarded**
2. Freeze the CNN; train one linear **SVM** per class on its features (stricter negative threshold)
3. Freeze the CNN; train a separate **box regressor**

Fast R-CNN: Sharing Computation via RoI Pooling



[*Girshick, 2015: Fast R-CNN*]

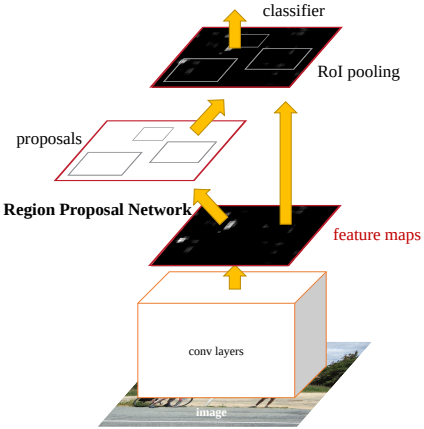
- **Key idea:** compute the CNN feature map **once per image**, rather than once per region proposal.
- **RoI Pooling:** project each proposal onto the shared feature map and extract a fixed-size feature representation (e.g. 7×7).
- **Joint training:** a softmax classifier and a bounding-box regressor are trained together with the CNN using a multi-task loss:

$$\mathcal{L} = \mathcal{L}_{\text{cls}}(p, u) + \lambda \, \mathbb{1}[u \geq 1] \, \mathcal{L}_{\text{loc}}(t^u, v)$$

Localization loss applies only to foreground RoIs ($u \geq 1$); background has label $u = 0$.

Faster R-CNN: Learned Proposals via RPN

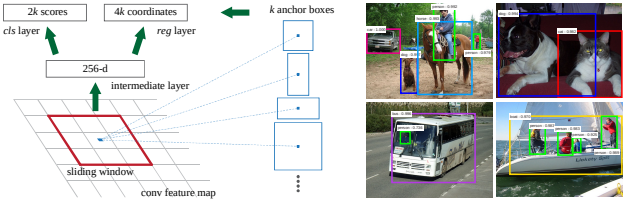
- Fast R-CNN still requires slow, externally-computed region proposals
- **Key idea**: a small **Region Proposal Network (RPN)** slides over the *same* shared feature map, predicting objectness + box offsets relative to **anchor boxes**
- **Optimization**: the original paper alternates training in **4 steps** (train RPN, train Fast R-CNN on its proposals, re-tune RPN with shared layers frozen, re-tune Fast R-CNN), modern implementations typically train the whole pipeline jointly, end-to-end, in one step



[*Ren et al., 2015: Faster R-CNN*]

Anchors

Definition (Anchor). a fixed reference box at a feature-map location, at a predefined scale and aspect ratio; the network regresses an offset from this box to the nearest object instead of predicting absolute coordinates.



[*Ren et al., 2015: RPN & anchors*]

- The anchor scales and aspect ratios are predefined, the network learns per-anchor offsets and objectness scores
- **Why/when effective**: a small offset from a known reference is easier to regress than absolute coordinates, but only if the anchor set's scales/aspect ratios cover the dataset's objects

From Offsets to Boxes: Encode vs. Decode

- **Encode**: given anchor (x_a, y_a, w_a, h_a) and matched ground truth (x, y, w, h), both in center-form:

$$t_x = \frac{x - x_a}{w_a}, \quad t_y = \frac{y - y_a}{h_a}, \quad t_w = \log \frac{w}{w_a}, \quad t_h = \log \frac{h}{h_a}$$

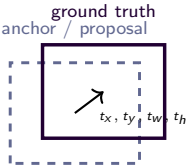
- **Why** log **for** w, h: a size ratio w/w_a is always positive and multiplicative; log turns it into an unbounded, roughly symmetric target around 0, like t_x, t_y, so a linear regressor can predict it directly. Decoding with $e^{(\cdot)}$ then guarantees $\hat{w}, \hat{h} > 0$ automatically, and equal steps in t_w mean equal *relative* size changes at any object scale
- **Decode** (inference): the network predicts $(\hat{t}_x, \hat{t}_y, \hat{t}_w, \hat{t}_h)$ per anchor; invert the same formulas to recover an absolute box:

$$\hat{x} = \hat{t}_x w_a + x_a, \quad \hat{y} = \hat{t}_y h_a + y_a, \quad \hat{w} = e^{\hat{t}_w} w_a, \quad \hat{h} = e^{\hat{t}_h} h_a$$

- **Center-to-corner**, for the NMS/IoU step:

$$x_1 = \hat{x} - \tfrac{\hat{w}}{2}, \quad y_1 = \hat{y} - \tfrac{\hat{h}}{2}, \quad x_2 = \hat{x} + \tfrac{\hat{w}}{2}, \quad y_2 = \hat{y} + \tfrac{\hat{h}}{2}$$

Box Regression & Multi-Task Losses



	cls	loc
positive (fg)	✓	✓
background (bg)	✓	✗

Background boxes still train the classifier ("not an object"), but **never** contribute to the localization loss

- Both losses follow the same recipe: a **classification** term (every box), plus a **localization** term that only counts for **positive** (foreground) boxes
- **Fast R-CNN** (per RoI; p: predicted class distribution, u: true class with u=0 background, t^u: predicted offset for class u, v: target offset):

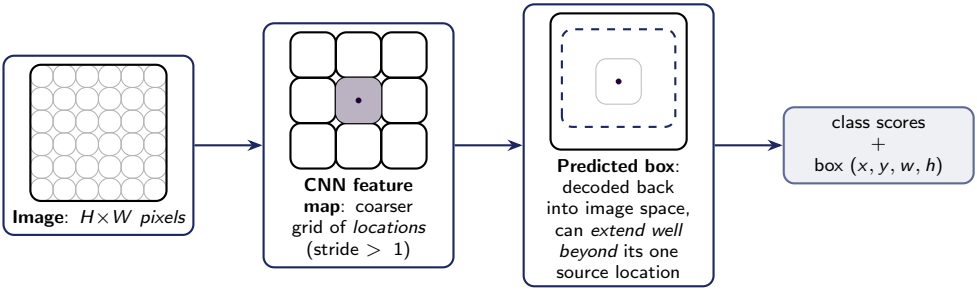
$$\mathcal{L}(p, u, t^u, v) = \underbrace{-\log p_u}_{\text{classification}} + \underbrace{\lambda\,[u \geq 1] \sum_i \mathrm{sm}_{L_1}(t_i^u - v_i)}_{\text{localization, } \mathbf{0} \text{ if } u=0}$$

- **Faster R-CNN RPN** (per anchor i; predicted objectness $p_i \in [0,1]$, ground-truth label $p_i^* \in \{0,1\}$; $t_i, t_i^* \in \mathbb{R}^4$: predicted/target offset):

$$\mathcal{L} = \tfrac{1}{N_{\text{cls}}} \textstyle\sum_i \underbrace{\mathcal{L}_{\text{cls}}(p_i, p_i^*)}_{\text{classification}} + \lambda \tfrac{1}{N_{\text{reg}}} \textstyle\sum_i \underbrace{p_i^*\, \mathrm{sm}_{L_1}(t_i - t_i^*)}_{\text{localization, } \mathbf{0} \text{ if } p_i^*=0}$$

$\mathrm{sm}_{L_1}(x) = 0.5x^2$ if $|x| < 1$, else $|x| - 0.5$: smooth near zero, linear further away, robust to outlier offsets.

From Feature Maps to Detection Predictions



Term	Meaning
Pixel	one element of the input image
Feature-map location / cell	one spatial position in the CNN's (coarser) output grid
Anchor box	a *predefined* reference box at a location (anchor-based methods only)
Predicted bounding box	the final decoded (x, y, w, h); not confined to its source cell
Receptive field	the region of the *input image* that influences one location's value

YOLO: You Only Look Once

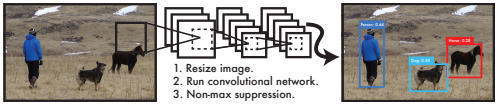


Figure: [*Redmon et al., 2016: YOLO*]

- Skip proposals: predict boxes directly in a *single* regression pass, still followed by NMS
- An $S \times S$ grid: each cell predicts B boxes + confidence, and one shared set of C class probabilities

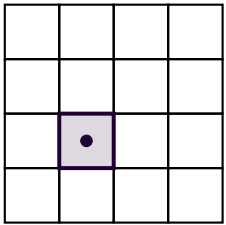
Definition (Grid cell). one of the $S \times S$ regions the image is divided into; a cell is **responsible** for an object if that object's center falls inside it (at most one object per cell). The predicted box itself is not confined to the cell: its (x, y) center is cell-relative, but its (w, h) can extend well beyond the cell's boundaries (previous slide).

Training loss, sum-squared error in 3 parts:

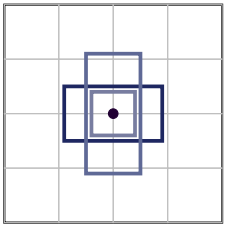
$$\mathcal{L} = \underbrace{\mathcal{L}_{\text{box}}}_{\text{resp. box only}} + \underbrace{\mathcal{L}_{\text{conf}}}_{\text{obj. vs. background}} + \underbrace{\mathcal{L}_{\text{cls}}}_{\text{per cell}}$$

- **Box**: $(x, y, \sqrt{w}, \sqrt{h})$ error, only for the one box responsible for an object
- **Confidence**: scored for every box; background boxes are down-weighted (far more of them than object boxes)
- **Class**: one probability vector per cell

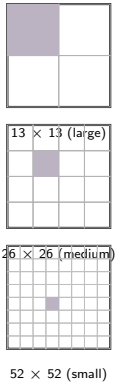
From YOLOv1 to Modern YOLO



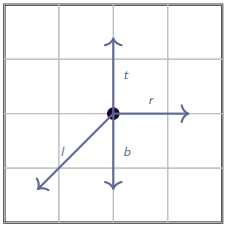
YOLOv1 (2016)
grid cell predicts boxes directly



YOLOv2 (2017)
anchor priors from k-means, centered per cell

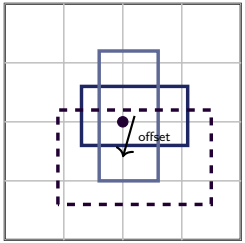


YOLOv3 (2018)
anchor-based: multiple boxes per location, at every location of all 3 scales



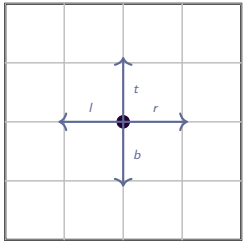
Ultralytics YOLOv8 (2023)
anchor-free, decoupled heads (not all variants identical)

Anchor-Based vs. Anchor-Free Detection



Anchor-based

- Predefined reference boxes (anchors) at every location
- Network predicts **offsets** relative to the nearest anchor
- Ground truth is assigned to the best-matching anchor



Anchor-free

- Predictions tied directly to feature-map locations, no predefined box shapes
- E.g. FCOS regresses (l, t, r, b): distances from the location to the box's 4 edges
- Ground truth is assigned directly to relevant locations (e.g. inside the box, near its center)

SSD: Multi-Scale Single-Shot Detection

- **Multiple** default boxes (anchors), at different aspect ratios, predicted at *every* location of *several* backbone resolutions at once, single pass; the multi-scale idea YOLOv3 later adopts
- Shallow, high-res. maps detect small objects; deep, low-res. maps detect large objects
- Unlike YOLOv1's *one* grid, SSD predicts from several differently-sized grids at once

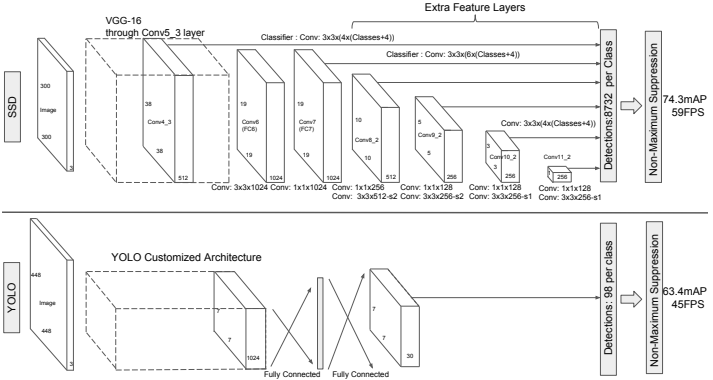


Figure: [*Liu et al., 2016: SSD vs. YOLO architecture*]

Default box scale at level k of m: $s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m-1}(k-1)$; size for aspect ratio a_r: $(w_k^a, h_k^a) = (s_k\sqrt{a_r},\ s_k/\sqrt{a_r})$

Predictions at multiple feature-map resolutions, each with its own receptive field and default boxes.

Fully Convolutional One-Stage Object Detection (FCOS)

- **Anchor-free**: every *positive* feature-map location (one that falls inside a ground-truth box) directly regresses (l, t, r, b), the distances to that object's 4 box edges; no predefined box shapes
- Multiple locations typically fall inside the same box, so *several* locations can all be associated with, and predict, the same object
- A **"center-ness"** branch down-weights boxes regressed from locations far from the object's center, suppressing low-quality predictions near box boundaries

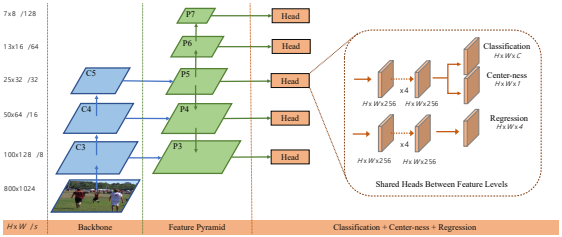
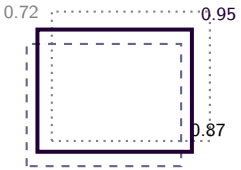


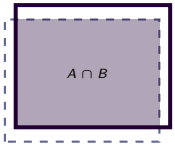
Figure: [*Tian et al., 2019: FCOS*]

Centerness target (distances l^*, t^*, r^*, b^* to the box edges): $\text{centerness}^* = \sqrt{\frac{\min(l^*, r^*)}{\max(l^*, r^*)} \cdot \frac{\min(t^*, b^*)}{\max(t^*, b^*)}}$

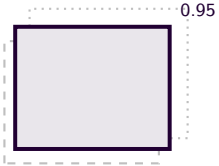
Non-Maximum Suppression



1. Sort and select: highest-confidence box (0.95) kept first



2. Compute IoU against each remaining box (shaded = intersection)



3. Suppress boxes with IoU $>$ threshold, **repeat** from step 1 with what remains

$$\text{IoU}(A, B) = \frac{\text{Area}(A \cap B)}{\text{Area}(A \cup B)} \approx \frac{3.0}{4.3} \approx 0.70$$

Threshold 0.50: $\text{IoU}(0.95, 0.87) \approx 0.70 > 0.50 \Rightarrow$ the 0.87 box is **suppressed** (and likewise the 0.72 box)

Greedy Non-Maximum Suppression

Input: boxes $\{(x_1^i, y_1^i, x_2^i, y_2^i)\}_{i=1}^n$, scores $\{s_i\}_{i=1}^n$, IoU threshold τ (run once per class)

$\mathcal{D} \leftarrow \emptyset$; $I \leftarrow$ indices sorted by s_i, descending
while $I \neq \emptyset$ **do**
  $m \leftarrow I[0]$ // highest remaining score
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{m\}$; remove m from I
  for $i \in I$ **do**
    $xx_1, yy_1 \leftarrow \max(x_1^m, x_1^i),\ \max(y_1^m, y_1^i)$ // intersection corners
    $xx_2, yy_2 \leftarrow \min(x_2^m, x_2^i),\ \min(y_2^m, y_2^i)$
    $\text{inter} \leftarrow \max(0, xx_2 - xx_1) \cdot \max(0, yy_2 - yy_1)$
    $\text{IoU} \leftarrow \text{inter} \,/\, (\text{area}_m + \text{area}_i - \text{inter})$
    if IoU $> \tau$ **then**
      remove i from I
    end if
  end for
end while
return $\mathcal{D}$

- Lower τ: more aggressive, risks suppressing nearby objects; higher τ: more duplicates survive. Naive cost $O(n^2)$; this is `torchvision.ops.nms`
- DETR avoids this step via one-to-one training, **YOLOv10** explores NMS-free inference via consistent dual assignments [*Wang et al., 2024: YOLOv10*], not shared by all YOLO variants

DETR: Detection as Set Prediction

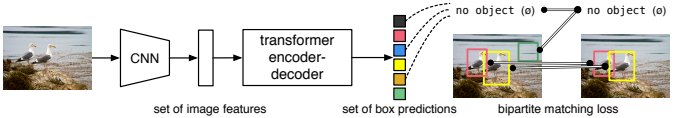


Figure: [*Carion et al., 2020: DETR, Fig. 1*]

1. A CNN backbone extracts a feature map, flattened into a sequence with fixed positional encodings
2. A **Transformer encoder** applies self-attention over this sequence (global context, unlike a CNN's local receptive field)
3. A **Transformer decoder** takes N fixed, **learned object queries** and cross-attends to the encoder output
4. A shared **FFN head** turns each decoder output embedding q_i into one prediction, $(\hat{p}_i(c), \hat{b}_i) = \mathrm{FFN}(q_i)$: a class distribution (incl. *"no object"*) and a box, for $i = 1, \ldots, N$

- **No anchors, no conventional NMS**: exactly N predictions per image, a direct *set* prediction matched **one-to-one** to ground truth (Hungarian algorithm) during training
- **DETR's backbone (step 1) is a CNN**, not a Vision Transformer; the Transformer only processes the resulting feature sequence. **YOLOS** [*Fang et al., 2021: YOLOS*] is a ViT-*native* detector: a *plain* ViT with DETR-style object queries

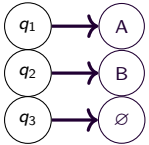
DETR: Why One-to-One Matching Removes NMS

- **Matching cost**: ground truth $y = (y_1, \ldots, y_N)$ is padded with $\varnothing$ up to N; Hungarian matching picks the assignment $\hat{\sigma}$ of queries to y minimizing:

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i=1}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$

$$\mathcal{L}_{\text{match}} = -\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)})$$

- **Training loss**: minimize the **same** cost at the optimal $\hat{\sigma}$ above (the Hungarian loss). The **classification** term runs over *all* N queries, including those matched to $\varnothing$ (trained to predict "no object"); the **box** term ($\mathcal{L}_{\text{box}} = \lambda_{L1}\|b - \hat{b}\|_1 + \lambda_{\text{iou}}\mathcal{L}_{\text{iou}}$) applies only to matched, foreground queries
- Two queries predicting the *same* object are penalized directly by this loss: duplicates are discouraged during **training**, not removed at **inference** like conventional NMS.



3 queries, 2 GT: $q_1 \to A$, $q_2 \to B$, $q_3 \to \varnothing$

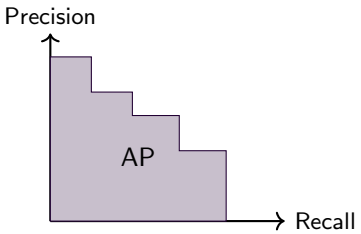
Three Ways to Organize Detection Predictions

Approach	**How predictions are organized**
Faster R-CNN	Learned region proposals, *then* classification and box refinement on each (two-stage)
YOLOv1	Dense predictions tied to *spatial grid cells*, each cell responsible for objects whose center falls inside it (one-stage, grid-based)
DETR	A fixed set of learned *object queries*, matched one-to-one to ground truth during training (one-stage, set-based)

Evaluating Detectors: Average Precision

1. For one class, rank *all* detections across the whole evaluation set by descending confidence
2. A detection is a **true positive (TP)** if its IoU with an unmatched ground-truth box of that class is $\geq$ threshold; otherwise it is a **false positive (FP)**
3. Walk down the ranked list: at each rank k, compute cumulative precision/recall over the first k detections
4. This traces one point per rank, the **precision–recall curve**; **AP** is the (interpolated) area under it

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}$$



AP = shaded area under the PR curve

From AP to mAP

- **mAP**: average AP over all classes
- **One-to-one matching**: each ground-truth box is matched by at most one prediction, the best-qualifying one; any extra detection of the same object is a **false positive**, not an extra true positive
- COCO convention, **mAP@[.5:.95]**: average over IoU thresholds 0.5 to 0.95 (step 0.05); stricter than mAP@0.5, and the standard reported number today
- A model can score high on mAP@0.5 but much lower on mAP@[.5:.95] if its boxes are only loosely aligned; always check both

Object Detection: Accuracy vs. Efficiency

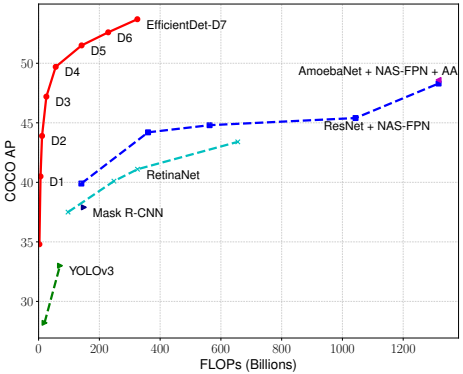


Figure: [*Tan et al., 2020: EfficientDet, Fig. 1*].

- Backbone and neck design matter as much as the detector family
- **FLOPs** $\neq$ **latency**: real speed also depends on resolution, memory access, and hardware

Advanced Semantic Segmentation

Multi-scale context and Transformer-based dense prediction

Why Dilated (Atrous) Convolution?

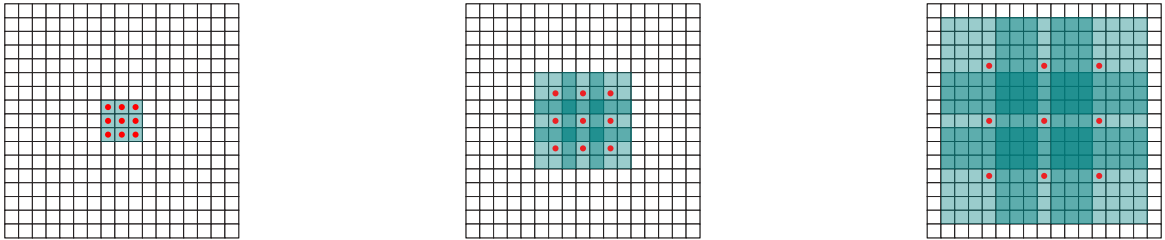


Figure: [*Yu & Koltun, 2016*]: dilation rate 1, 2, 4

$$\text{effective kernel size} = d(k-1)+1$$

- Builds on last lecture's FCN/U-Net: pooling grows the receptive field but shrinks resolution
- **Stack layers with increasing dilation** (e.g. $d = 1, 2, 4, \dots$): the receptive field grows exponentially with *depth*, while the kernel weight count per layer stays fixed at k^2
- An alternative to pooling for more context *without* losing spatial resolution

Atrous Spatial Pyramid Pooling (ASPP)

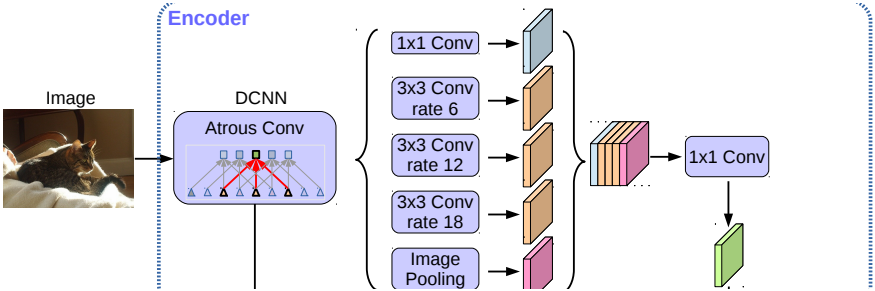


Figure: [*Chen et al., 2018: DeepLabv3+, Fig. 2 (encoder, cropped)*]

- Objects in the same image appear at different scales; one receptive field cannot match all of them
- The *same* backbone feature map passes through **5 parallel branches**: a 1×1 conv, three 3×3 atrous convs at increasing rates (6, 12, 18), and global image pooling; concatenated and fused by a final 1×1 conv
- Structurally the same idea as an **Inception module** (parallel branches at different receptive fields, concatenated); here the branches vary *dilation rate* instead of kernel size

DeepLabv3+: Full Pipeline

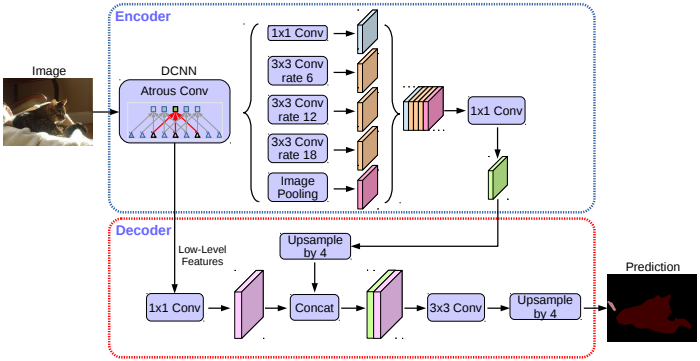


Figure: [*Chen et al., 2018: DeepLabv3+, Fig. 2*]

- **Encoder**: an atrous backbone feeds **ASPP** for multi-scale context without losing spatial resolution
- **Decoder**: the **skip connection** (highlighted) carries low-level backbone features via a 1×1 conv around ASPP, **concatenated** with $4\times$ upsampled ASPP output, refined by a 3×3 conv, upsampled $4\times$ again
- This U-Net-style skip distinguishes DeepLabv3+ from plain DeepLabv3, giving sharper object boundaries

Recall: Evaluating Segmentation – IoU



Ground Truth



Prediction



Intersection ($\hat{Y} \cap Y$)

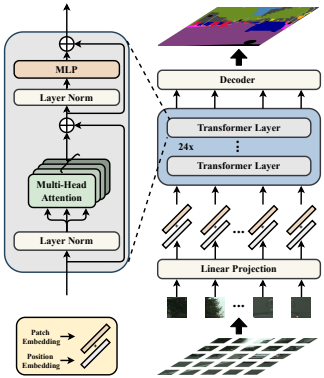


Union ($\hat{Y} \cup Y$)

- Legend (class: vegetation): **green** = agree (TP), **red** = prediction only (FP), **blue** = ground truth only (FN)
- $IoU = \frac{|\hat{Y} \cap Y|}{|\hat{Y} \cup Y|} \approx 0.78$ for this example
- **mIoU** averages IoU over all classes, $\text{mIoU} = \frac{1}{C} \sum_{c=1}^{C} IoU_c$: robust to class imbalance, unlike pixel accuracy

SETR: Plain ViT Encoder for Segmentation

- Alternative to atrous convolution: scale context via self-attention instead of dilation
- **Tokenize**: split the $H \times W$ image into $p \times p$ patches ($p = 16$), linearly project each to $\mathbb{R}^C$, add position embeddings: $L = HW/p^2$ tokens total
- **Plain ViT encoder**: L_e Transformer layers, global self-attention from *layer 1*; no hierarchy, no multi-scale features (unlike CNN backbones)
- **Reshape**: encoder output $\in \mathbb{R}^{L \times C} \rightarrow \mathbb{R}^{(H/p) \times (W/p) \times C}$ feature map
- **PUP decoder**: since $p = 16 = 2^4$, 4 stages of (3×3 conv + 2× bilinear upsample) recover full resolution
- Simplest of the three ViT-based models here, but also the most compute-hungry: $O(L^2)$ self-attention over *all* L tokens, every layer



[*Zheng et al., 2021: SETR*]

SegFormer: Hierarchical Encoder, Lightweight Decoder

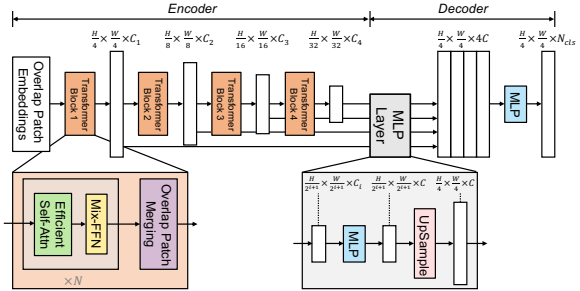


Figure: [*Xie et al., 2021: SegFormer*]

- **Hierarchical** MiT encoder: 4 stages, shrinking resolution like a CNN (feature maps at $\approx 1/4, 1/8, 1/16, 1/32$ of the input); multi-resolution features F_i, unlike SETR's single scale
- **All-MLP decoder**, per stage $i = 1..4$: project to a common channel size C, upsample to $1/4$ resolution, fuse, predict:

$$\hat{F}_i = \text{Upsample}\big(\text{Linear}(C_i, C)(F_i)\big), \quad M = \text{Linear}(C, N_{\text{cls}})\big(\text{Linear}(4C, C)(\text{Concat}(\hat{F}_1, \ldots, \hat{F}_4))\big)$$

SegFormer: Inside the Encoder Block

- **Overlapped Patch Merging**: overlapping windows between stages (unlike ViT's non-overlapping patchify) preserve local continuity
- **Efficient Self-Attention**: full attention over all N tokens is too costly at the high-resolution early stages, so reshape $K \in \mathbb{R}^{N \times C}$ to reduce its sequence length by a ratio R before attention, then project back to C channels:

$$\hat{K} = \text{Linear}(C \cdot R,\ C)\Big(\text{Reshape}\big(\tfrac{N}{R},\ C \cdot R\big)(K)\Big)$$

  (same projection for V), cutting attention cost from $O(N^2)$ to $O(N^2/R)$
- **Mix-FFN**: fixed positional embeddings do not generalize when train and test resolutions differ, so a 3×3 depthwise conv inside the MLP injects positional information directly instead:

$$\mathbf{x}_{\text{out}} = \text{MLP}\Big(\text{GELU}\big(\text{Conv}_{3\times 3}(\text{MLP}(\mathbf{x}_{\text{in}}))\big)\Big) + \mathbf{x}_{\text{in}}$$

DPT: ViT + Convolutional Reassemble

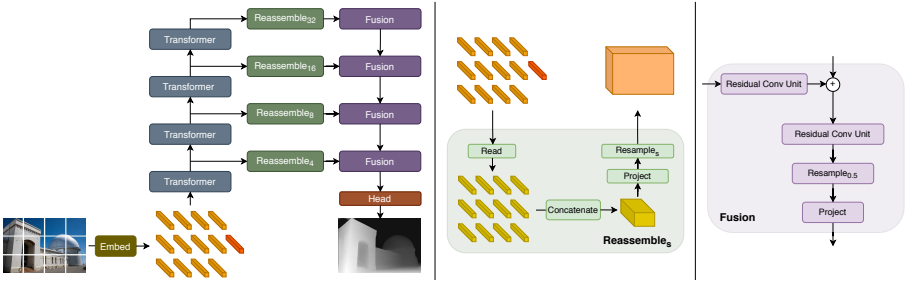


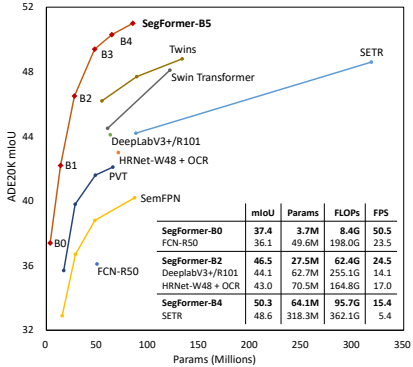
Figure: [*Ranftl et al., 2021: DPT*]

- **Reassemble**: turns ViT tokens from *multiple* layers back into feature maps at 4 different resolutions
- **Fusion**: a conv module progressively merges these multi-scale maps, state of the art for monocular depth estimation

DPT: ViT + Convolutional Reassemble

- **Reassemble**$_s$ (one per chosen ViT depth, scale $s \in \{1/4, 1/8, 1/16, 1/32\}$): **Read** (drop/project the readout token) $\rightarrow$ **Concatenate** (tokens back into an $h \times w$ grid) $\rightarrow$ **Resample**$_s$ (1×1 conv + resize)
- **Fusion** (finest-last): two residual conv units, then $2\times$ upsample + 1×1 project, summed with the next (finer) Reassemble's output
- Features from *multiple* ViT depths at *multiple* scales give CNN-like multi-scale fusion without a hierarchical encoder

Semantic Segmentation: Accuracy vs. Efficiency



	mIoU	Params	FLOPs	FPS
SegFormer-B0	**37.4**	**3.7M**	**8.4G**	**50.5**
FCN-R50	36.1	49.6M	198.0G	23.5
SegFormer-B2	**46.5**	**27.5M**	**62.4G**	**24.5**
DeeplabV3+/R101	44.1	62.7M	255.1G	14.1
HRNet-W48 + OCR	43.0	70.5M	164.8G	17.0
SegFormer-B4	**50.3**	**64.1M**	**95.7G**	**15.4**
SETR	48.6	318.3M	362.1G	5.4

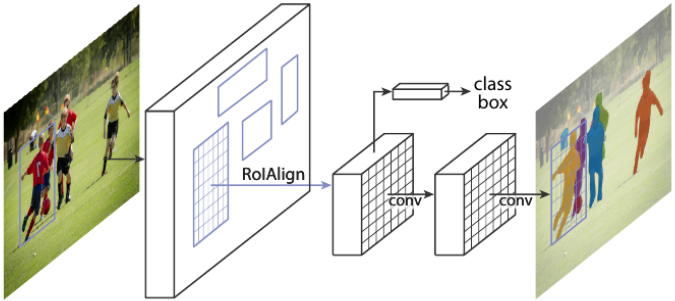
Figure: [*Xie et al., 2021: SegFormer, Fig. 1*]: ADE20K mIoU vs. parameter count

- **SegFormer-B2** (27.5M): 46.5 mIoU @ 24.5 FPS vs. **DeepLabV3+/R101** (62.7M): 44.1 mIoU @ 14.1 FPS

Instance & Panoptic Segmentation

Combining object localization with pixel-level understanding

Mask R-CNN: From Boxes to Instance Masks



[*He et al., 2017: Mask R-CNN, Fig. 1*]

- Adds a **mask head** (a tiny FCN) on top of every RoI in Faster R-CNN: same RPN, same anchors, same box head; box + class + per-instance mask *is* instance segmentation
- **RoIAlign**: bilinear sampling, so the mask aligns to the actual pixels
- **Which box is which**: GT boxes supervise RPN training; RPN **proposals** feed RoIAlign; GT **masks** supervise only the mask head; at **inference** both heads predict from scratch

RoIAlign: Bilinear Sampling in Detail

- **RoI Pooling** (Fast/Faster R-CNN) rounds the RoI boundary, then rounds each bin boundary, to integer feature-map cells: two roundings misalign the extracted features from the actual RoI
- **RoIAlign** keeps every coordinate continuous, with no rounding at all. Divide the RoI into $k \times k$ bins; sample r regularly-spaced *continuous* points (x, y) per bin; each point is read off the feature map by **bilinear interpolation** from its 4 neighboring cells:

$$F(x,y) = \sum_{i,j \in \{0,1\}} w_{ij}\, F[x_0{+}i,\, y_0{+}j], \quad x_0{=}\lfloor x \rfloor,\; y_0{=}\lfloor y \rfloor$$

$$w_{00} = (1{-}dx)(1{-}dy),\; w_{10} = dx(1{-}dy),\; w_{01} = (1{-}dx)dy,\; w_{11} = dx\,dy, \quad dx{=}x{-}x_0,\; dy{=}y{-}y_0$$

- Each bin's output is the average (or max) of its r sampled values. This sub-pixel accuracy is what lets the predicted mask align to the true object pixels

Mask R-CNN: Multi-Task Loss

- Per RoI, three losses trained jointly (same $\mathcal{L}_{\text{cls}}$, $\mathcal{L}_{\text{box}}$ as Faster R-CNN, plus a new mask term):

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}} + \mathcal{L}_{\text{mask}}$$

- For C classes, the mask head outputs C binary $m \times m$ masks per RoI (one per class). $\mathcal{L}_{\text{mask}}$ is the average per-pixel **binary cross-entropy**, computed *only* on the channel of the **ground-truth class**: the other $C-1$ channels do not contribute to the loss for that RoI
- **Why this matters**: mask prediction is **decoupled** from classification. Each class's mask is a per-pixel sigmoid, not a softmax competing against other classes, since the classification branch already decides *which* class; the mask branch only answers "object or background" for that one class, which works better empirically than a per-pixel softmax over all classes (as in semantic segmentation)

[*He et al., 2017: Mask R-CNN*]

Panoptic Segmentation: Unifying Everything



Image | Semantic (stuff) | Instance (things) | Panoptic

Figure: [*Kirillov et al., 2019: Panoptic Segmentation*]

- **Stuff** (sky, road: uncountable, no instances): one semantic segmentation head. **Things** (countable objects): one instance segmentation head (Mask R-CNN)
- **Merging, pixel by pixel**: (1) sort instance masks by confidence and paste them onto the canvas highest-confidence first, so each pixel is claimed by *at most one* instance, avoiding overlap; (2) any pixel not claimed by an instance falls back to the semantic head's prediction (a stuff class, or *void*)
- Every pixel ends up with exactly one (class, instance id) pair; instance id is meaningful only for things

Panoptic Quality (PQ)

- Over matched segments with IoU > 0.5 (TP), unmatched predictions (FP), unmatched ground truth (FN):

$$\text{PQ} = \underbrace{\frac{\sum_{(p,g)\in TP} \text{IoU}(p,g)}{|TP|}}_{\text{Segmentation Quality (SQ)}} \times \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{\text{Recognition Quality (RQ)}}$$

- **SQ**: how well matched segments overlap
- **RQ**: what fraction of segments were matched at all (like an F1 score for detection)

Recap: Classification vs. Detection vs. Segmentation

Task	**Output**	**Label (GT)**	**Training loss**	**At inference**
Classification	1 class for the whole image	class id	cross-entropy	arg max over softmax
Object detection	set of (box, class, score)	boxes + class per object	CE (cls) + smooth-L_1 (box), per anchor/RoI	NMS over candidate boxes
Semantic seg.	class per *pixel*	per-pixel class mask	per-pixel CE (+ Dice)	arg max per pixel
Instance seg.	box + binary mask per object	boxes + per-instance masks	CE + box + mask (Mask R-CNN)	NMS, then threshold mask per kept box
Panoptic seg.	class *and* instance ID per pixel	merged semantic + instance GT	semantic + instance losses, combined	merge semantic arg max with instance proposals

Summary

Compare architectures by *core mechanism*, not by publication order:

- **Localization**: exhaustive sliding windows (Viola-Jones) $\rightarrow$ hand-crafted region proposals (Selective Search) $\rightarrow$ *learned* proposals (RPN) $\rightarrow$ direct regression (YOLO)
- **Prediction representation**: proposal-based (Faster R-CNN), dense grid/one-stage (YOLO, SSD, FCOS), and query-based set prediction (DETR)
- **Handling object scale**: anchors (Faster R-CNN, YOLOv2+, SSD), multi-scale feature maps (SSD, YOLOv3+), and anchor-free per-location regression (FCOS, modern YOLO)
- **Output refinement**: greedy **NMS** for most detectors vs. DETR's end-to-end bipartite matching (no NMS needed)
- **Dense segmentation**: atrous convolution / **ASPP** (DeepLab) and ViT-based decoders (SETR, SegFormer, DPT) both push resolution and context further than plain FCN/U-Net
- **Unified scene understanding**: boxes (detection) + masks (Mask R-CNN) + per-pixel classes (semantic segmentation) combine into **panoptic** segmentation

Evaluation: **mAP** for detection, **mIoU** for segmentation, **PQ** $=$ SQ $\times$ RQ for panoptic.

References

- Viola & Jones (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. CVPR 2001.
- Felzenszwalb & Huttenlocher (2004). Efficient Graph-Based Image Segmentation. IJCV 2004.
- Uijlings, van de Sande, Gevers & Smeulders (2013). Selective Search for Object Recognition. IJCV 2013.
- Girshick et al. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation (R-CNN). CVPR 2014.
- Girshick (2015). Fast R-CNN. ICCV 2015.
- Ren, He, Girshick & Sun (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS 2015.
- He, Gkioxari, Dollár & Girshick (2017). Mask R-CNN. ICCV 2017.
- Redmon, Divvala, Girshick & Farhadi (2016). You Only Look Once (YOLO). CVPR 2016.
- Liu et al. (2016). SSD: Single Shot MultiBox Detector. ECCV 2016.
- Wang et al. (2024). YOLOv10: Real-Time End-to-End Object Detection. NeurIPS 2024.
- Tian, Shen, Chen & He (2019). FCOS: Fully Convolutional One-Stage Object Detection. ICCV 2019.
- Neubeck & Van Gool (2006). Efficient Non-Maximum Suppression. ICPR 2006.
- Everingham et al. (2010). The PASCAL Visual Object Classes (VOC) Challenge. IJCV 2010.
- Lin et al. (2014). Microsoft COCO: Common Objects in Context. ECCV 2014.
- Tan, Pang & Le (2020). EfficientDet: Scalable and Efficient Object Detection. CVPR 2020.
- Carion et al. (2020). End-to-End Object Detection with Transformers (DETR). ECCV 2020.
- Fang et al. (2021). You Only Look at One Sequence (YOLOS). NeurIPS 2021.
- Yu & Koltun (2016). Multi-Scale Context Aggregation by Dilated Convolutions. ICLR 2016.
- Chen et al. (2017). Rethinking Atrous Convolution for Semantic Image Segmentation (DeepLabv3).
- Chen et al. (2018). Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation (DeepLabv3+). ECCV 2018.
- Zhao et al. (2017). Pyramid Scene Parsing Network (PSPNet). CVPR 2017.
- Zheng et al. (2021). Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers (SETR). CVPR 2021.
- Xie et al. (2021). SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. NeurIPS 2021.
- Ranftl et al. (2021). Vision Transformers for Dense Prediction (DPT). ICCV 2021.
- Kirillov, He, Girshick, Rother & Dollár (2019). Panoptic Segmentation. CVPR 2019.
- Cordts et al. (2016). The Cityscapes Dataset for Semantic Urban Scene Understanding. CVPR 2016.
- Zhou et al. (2017). Scene Parsing through ADE20K Dataset. CVPR 2017.

Appendix

Supplementary derivations

Selective Search: Full Merge Pseudocode

Input: initial regions $R = \{r_1, \ldots, r_n\}$ (superpixels, e.g. Felzenszwalb–Huttenlocher), similarity function $s(\cdot, \cdot)$

1: $S \leftarrow \{\, s(r_i, r_j) : r_i, r_j \text{ adjacent} \,\}$ // all adjacent-pair similarities
2: **while** $S \neq \emptyset$ **do**
3: $(r_i, r_j) \leftarrow \arg\max S$ // most similar pair
4: $r_t \leftarrow r_i \cup r_j$; $R \leftarrow R \cup \{r_t\}$
5: remove from S every similarity involving r_i or r_j
6: $S \leftarrow S \cup \{\, s(r_t, r_k) : r_k \text{ adj. to } r_t \,\}$ // similarities for merged region
7: **end while**
8: **return** the bounding box of *every* region ever added to R

[*Uijlings et al., 2013, Algorithm 1*]

Computing AP: The Full Ranking Algorithm

Input: for one class, n detections ranked by confidence (descending), g ground-truth boxes.

$tp, fp \leftarrow$ zero arrays of length n; matched $\leftarrow \emptyset$
for $k = 1 \ldots n$ **do**
  $j^\star \leftarrow \arg\max_{j \notin \text{matched}} \text{IoU}(\text{det}_k, \text{gt}_j)$
  if $\text{IoU}(\text{det}_k, \text{gt}_{j^\star}) \geq 0.5$ **then**
    $tp[k] \leftarrow 1$; matched $\leftarrow$ matched $\cup \{j^\star\}$
  else
    $fp[k] \leftarrow 1$
  end if
end for
$TP_k \leftarrow \sum_{l \leq k} tp[l]$, $FP_k \leftarrow \sum_{l \leq k} fp[l]$ // cumulative sums
$P_k \leftarrow TP_k / (TP_k + FP_k)$, $R_k \leftarrow TP_k / g$ for $k = 1 \ldots n$

This produces exactly the P_k, R_k columns of the worked example on the main lecture slide. [*Lin et al., 2014: COCO; Everingham et al., 2010: PASCAL VOC*]

Viola-Jones: Combining Weak Classifiers

- **AdaBoost** combines many weak classifiers h_t, each a threshold on *one* Haar feature, into a single strong classifier:

$$H(x) = \text{sign}\Big(\sum_t \alpha_t \, h_t(x) \Big), \qquad \alpha_t = \tfrac{1}{2} \ln \tfrac{1-\epsilon_t}{\epsilon_t}$$

- ϵ_t is the weighted training error of h_t: lower error gives that weak classifier a larger vote α_t in the final sum

[*Viola & Jones, 2001*]

Selective Search: Histogram Details & Efficiency

- **Color histogram** c_i: for each of the 3 color channels, a 25-bin intensity histogram over region r_i's pixels, concatenated (75 bins) and L1-normalized
- **Texture histogram** t_i: Gaussian derivatives at 8 orientations, each channel, 10 bins per orientation; same intersection formula as color
- **Efficient to merge**: a merged region's histogram comes from its parts' histograms directly, no re-scanning pixels:

$$C_t = \frac{\text{size}(r_i)C_i + \text{size}(r_j)C_j}{\text{size}(r_i) + \text{size}(r_j)}$$

- **Diversification**: re-running the whole pipeline in multiple color spaces (RGB, HSV, Lab, ...) helps because each space is invariant to a *different, limited* set of imaging conditions (e.g. illumination, shadows)

[*Uijlings et al., 2013*]


1


2


3


4


5


6


7


8


9


10


11


12


13


14


15


16


17


18


19


20


21


22


23


24


25


26


27


28


29


30


31


32


33


34


35


36


37


38


39


40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62