Object Detection Basics
Deep Learning for Computer Vision

Nipun Batra · IIT Gandhinagar

Teaching computers to see and locate objects

Why Object Detection?

```
Imagine you're driving...

┌─────────────────────────────────────────────────┐
│                                                   │
│   [Car] <-- Your car sees this scene              │
│                                                   │
│       ┌───────────────────────────────────────┐  │
│       │  [Ped] pedestrian   [Car] car    [Bike] cyclist │  │
│       │        ↓                ↓            ↓    │  │
│       │   STOP!           SLOW DOWN      CAREFUL! │  │
│       │                                          │  │
│       │  Your car needs to know:                 │  │
│       │  ● WHAT is there? (classification)       │  │
│       │  ● WHERE is it? (localization)           │  │
│       │  ● HOW MANY? (counting)                  │  │
│       │                                          │  │
│       └───────────────────────────────────────┘  │
│                                                   │
└─────────────────────────────────────────────────┘
```

Classification alone says "there's a person somewhere" — **not enough!**

What You Will Learn Today

```
Part 1: The Core Problem
        Classification → Detection → Segmentation

Part 2: Bounding Boxes
        How we represent "WHERE is the object?"

Part 3: IoU (Intersection over Union)
        How we measure "Is this box correct?"

Part 4: NMS (Non-Maximum Suppression)
        How we handle "Too many boxes!"

Part 5: YOLO Architecture
        How real detectors work

Part 6: Training & Metrics
        How we train and evaluate detectors
```
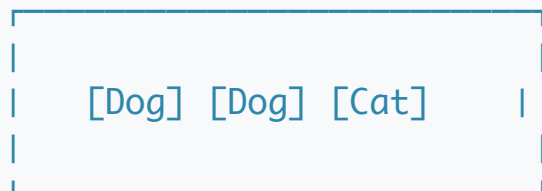
Part 1: The Core Problem
What IS Object Detection?

The Vision Task Hierarchy

Level 1: CLASSIFICATION   "What is in this image?"
_____

```
    ┌─────────────────────┐
    │                     │
    │   [Dog] [Dog] [Cat] │  ->  Output: "dog" (or "cat")
    │                     │
    └─────────────────────┘
```

Just ONE label for the whole image

Level 2: OBJECT DETECTION   "What + Where?"
_____

```
    ┌─────────────────────┐
    │  ┌───┐  ┌───┐  ┌───┐ │  →  Output: 3 boxes with labels
    │  │Dog│  │Dog│  │Cat│ │        dog (0.95) @ [10,20,50,80]
    │  └───┘  └───┘  └───┘ │        dog (0.92) @ [100,20,140,85]
    └─────────────────────┘         cat (0.88) @ [200,30,250,90]
```

Level 3: INSTANCE SEGMENTATION   "Exact shape of each object"
_____
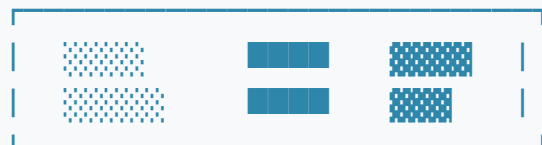
```
    ┌─────────────────────┐
    │  ░░░    ███    ░░░   │  →  Pixel-perfect mask for each
    │  ░░░    ███    ░░░   │        individual object
    └─────────────────────┘
```

Classification vs Detection: Key Difference

Classes: bus, stop sign, person
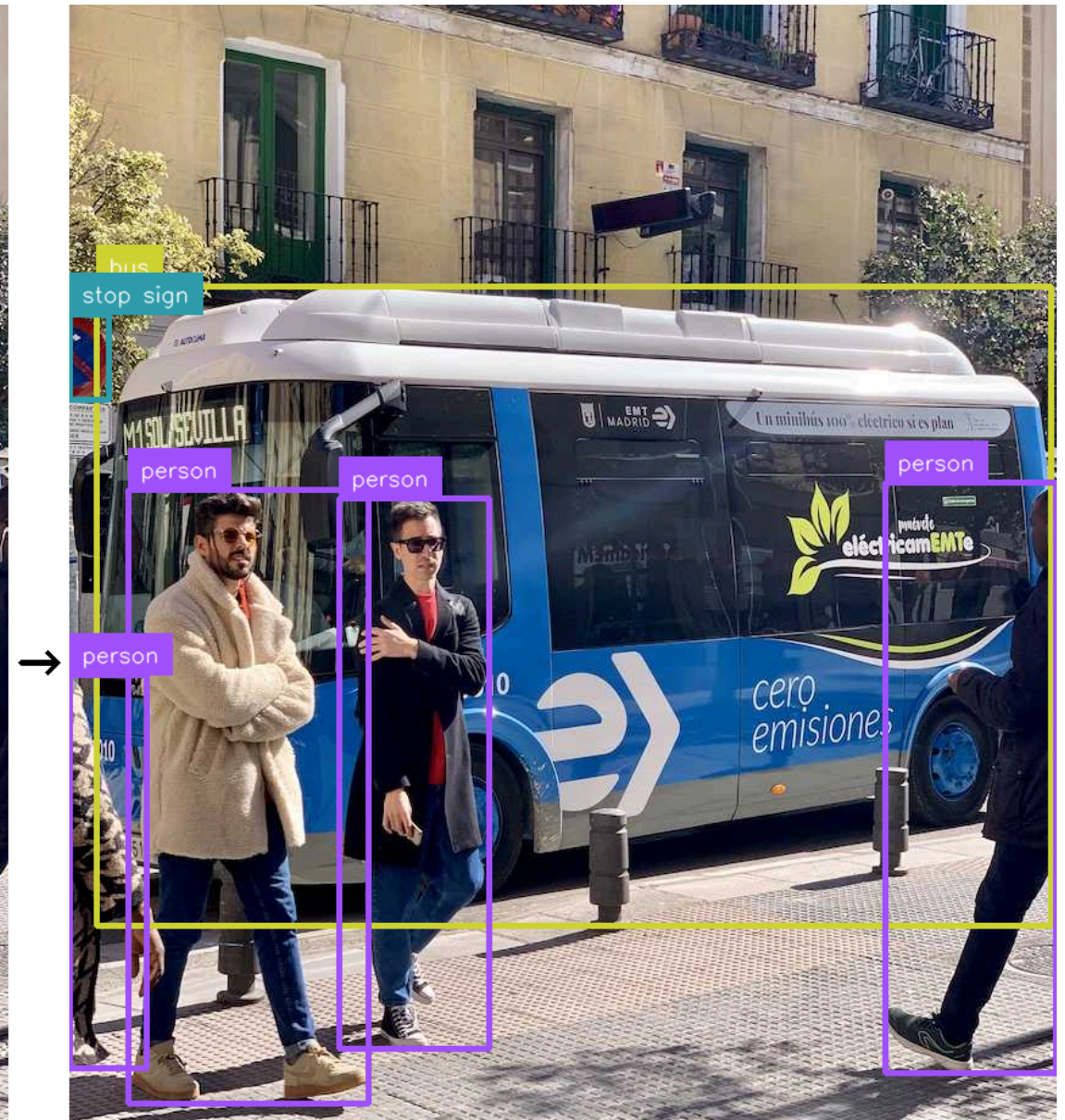
A More Detailed Comparison

Aspect	Classification	Detection	Segmentation
Output	Single label	Boxes + labels	Pixel masks
Multiple objects?	No (or multi-label)	Yes ✓	Yes ✓
Location?	No	Box ✓	Exact shape ✓
Typical use	"What is this?"	"Where are things?"	"Precise boundaries"
Difficulty	Easier	Medium	Harder
Speed	Fastest	Fast	Slower

Real applications: • Classification: "Is this X-ray normal?" • Detection: "Where are the tumors?" • Segmentation: "Exact tumor boundary for surgery planning"

Real-World Detection Applications

```
+-----------------------------------------------------------------+
|                 OBJECT DETECTION IN THE WILD                    |
+-----------------------------------------------------------------+
|                                                                 |
|  AUTONOMOUS DRIVING              SMARTPHONE CAMERAS              |
|  • Pedestrians, cars, signs      • Face detection for focus     |
|  • Traffic lights, lanes         • Scene understanding          |
|  • Cyclist detection             • AR filters positioning       |
|                                                                 |
|  RETAIL & INVENTORY              MEDICAL IMAGING                 |
|  • Shelf stock monitoring        • Tumor detection              |
|  • Checkout-free stores (Amazon Go) • Cell counting             |
|  • Customer flow analysis        • Organ localization           |
|                                                                 |
|  SECURITY & SURVEILLANCE         MANUFACTURING                  |
|  • Intrusion detection           • Defect detection             |
|  • Crowd monitoring              • Quality control              |
|  • License plate recognition     • Safety compliance            |
|                                                                 |
+-----------------------------------------------------------------+
```

Instance Segmentation: Beyond Boxes

DETECTION gives you BOXES:

```
Person
standing
```
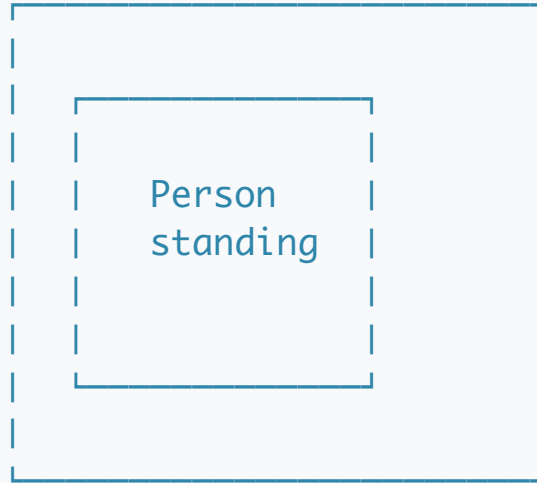
Box includes background!

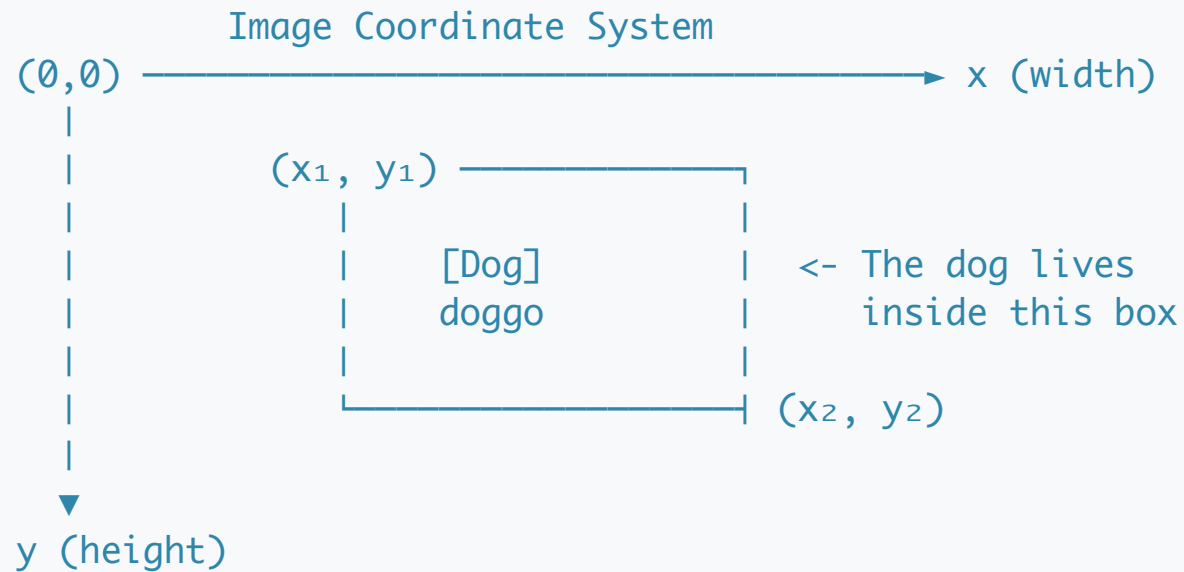SEGMENTATION gives you MASKS:
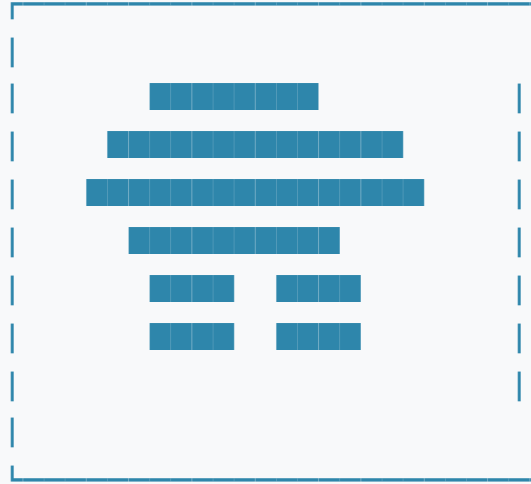
Every pixel classified!

Segmentation is needed for: video editing backgrounds, medical boundaries, precise robotics

Part 2: Bounding Boxes
How We Represent "WHERE"

What IS a Bounding Box?

A bounding box is a RECTANGLE that tightly contains an object.

```
                    Image Coordinate System
         (0,0) ─────────────────────────────────────►  x (width)
            |
            |          (x₁, y₁) ─────────────┐
            |             |                   |
            |             |                   |
            |             |     [Dog]         |    <- The dog lives
            |             |     doggo         |        inside this box
            |             |                   |
            |             └───────────────────┤ (x₂, y₂)
            |
            ▼
      y (height)
```

The box is defined by 4 numbers. But WHICH 4?

Different systems use different conventions — this causes many bugs!

The Three Main Formats

Format 1: CORNER FORMAT (x$_1$, y$_1$, x$_2$, y$_2$)
_____

- Top-left and bottom-right corners
- Used by: PyTorch, many APIs
- Easy for: drawing, IoU calculation
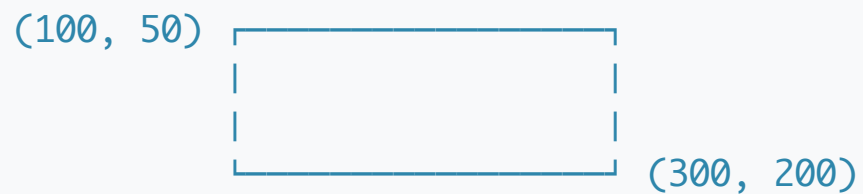
```
(100, 50)  ┌──────────────────┐
           │                  │
           │                  │
           └──────────────────┘ (300, 200)


        Box = [100, 50, 300, 200]
        Width = 300 - 100 = 200
        Height = 200 - 50 = 150
```

The Three Main Formats (continued)
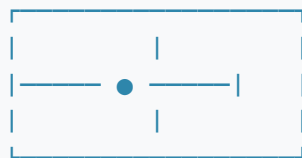
```
Format 2: CENTER FORMAT (cx, cy, w, h)
_____

• Center point + width + height
• Used by: YOLO, many papers
• Easy for: predicting from grid cells
```

```
        ┌─────────────────┐
        │        │        │
        │─────── ● ───────│   │    ● = center (200, 125)
        │        │        │        w = 200, h = 150
        └─────────────────┘
```

```
        Box = [200, 125, 200, 150]
        x₁ = 200 - 200/2 = 100
        y₁ = 125 - 150/2 = 50
```

$x_1 = 200 - 200/2 = 100$

$y_1 = 125 - 150/2 = 50$

```
Format 3: CORNER + SIZE (x, y, w, h)
_____

• Top-left corner + width + height
• Used by: COCO dataset format
• Easy for: drawing (x,y is start point)
```

```
    (100, 50) ┌──────────────┐
              │              │   w=200
              │              │   h=150
              └──────────────┘
```

```
        Box = [100, 50, 200, 150]
```

Format Conversion Cheat Sheet

```python
# Corner (x1,y1,x2,y2) → Center (cx,cy,w,h)
cx = (x1 + x2) / 2
cy = (y1 + y2) / 2
w = x2 - x1
h = y2 - y1

# Center (cx,cy,w,h) → Corner (x1,y1,x2,y2)
x1 = cx - w/2
y1 = cy - h/2
x2 = cx + w/2
y2 = cy + h/2

# Corner+Size (x,y,w,h) → Corner (x1,y1,x2,y2)
x1 = x
y1 = y
x2 = x + w
y2 = y + h
```

Common bug alert! Always check your dataset's format before training. COCO uses [x,y,w,h], PASCAL VOC uses [x1,y1,x2,y2], YOLO uses normalized [cx,cy,w,h]

Absolute vs Normalized Coordinates

```
ABSOLUTE COORDINATES (Pixels):

    Image size: 640 × 480 pixels
    Box: [100, 50, 300, 200]  ← pixel values

    Problem: What if image is resized to 320 × 240?
             Box values become invalid!

NORMALIZED COORDINATES (0 to 1):

    Image size: ANY
    Box: [0.156, 0.104, 0.469, 0.417]  ← proportions

    Conversion:
    x_norm = x_pixel / image_width
    y_norm = y_pixel / image_height

    Same box works for ANY image size!
```

YOLO uses normalized center format: [cx/W, cy/H, w/W, h/H] — all values between 0 and 1

Quick Format Quiz

```
Image: 640 × 480 pixels
Object is at: top-left (100, 50), bottom-right (300, 200)

What is the box in each format?


┌─────────────────────────────┬───────────────────────────────┐
| Format                      | Values                        |
├─────────────────────────────┼───────────────────────────────┤
| Corner (xyxy)               | [100, 50, 300, 200]           |
| Corner+Size (xywh)          | [100, 50, 200, 150]           |
| Center (cxcywh)             | [200, 125, 200, 150]          |
| Normalized xyxy             | [0.156, 0.104, 0.469, 0.417]  |
| Normalized cxcywh           | [0.312, 0.260, 0.312, 0.312]  |
└─────────────────────────────┴───────────────────────────────┘


Always verify which format your framework expects!
```

Part 3: Intersection over Union (IoU)
How Good Is a Detection?

The Core Question

```
Your model predicted a box. Ground truth is another box.
Is your prediction "correct"?


        ┌──────────────────┐
        │   Ground Truth   │
        │   (from human)   │
        │        ┌─────────┼────────┐
        │        │         │        │
        │        │  OVERLAP│        │
        │        │         │        │
        └────────┼─────────┘        │
                 │                  │
                 │    Prediction    │
                 │   (from model)   │
                 └──────────────────┘


Question: How do we measure if prediction is "close enough"?

Answer: IoU = Intersection / Union
```
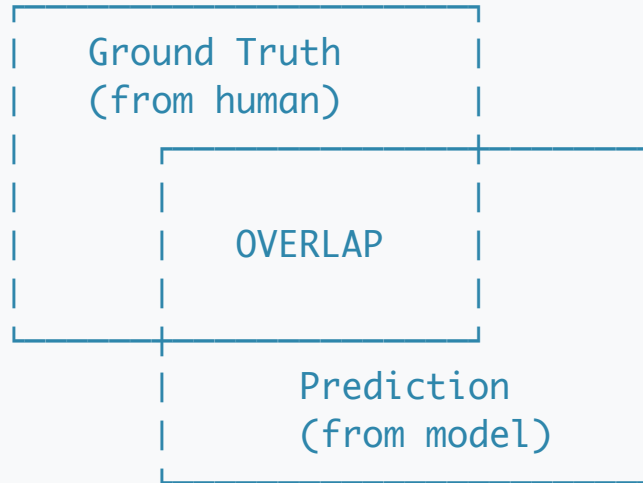
IoU: The Formula

$$IoU = \frac{\text{Area of OVERLAP}}{\text{Area of BOTH boxes (counted once)}}$$

```
| A |
|   |---------|
|   |▪▪▪▪|    |
|   |▪▪▪▪|    |
|---------|  B |
    |         |
    |---------|
```
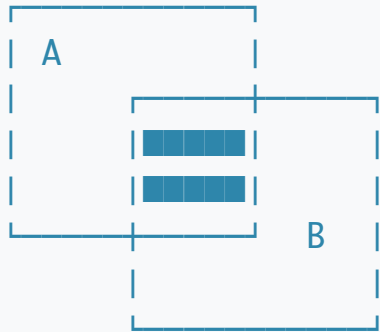
INTERSECTION = shaded area
UNION = A + B - Intersection

$$IoU = \frac{\text{Intersection}}{\text{Union}}$$

$$IoU = \frac{\text{Intersection}}{A + B - \text{Intersection}}$$

IoU: Visual Example
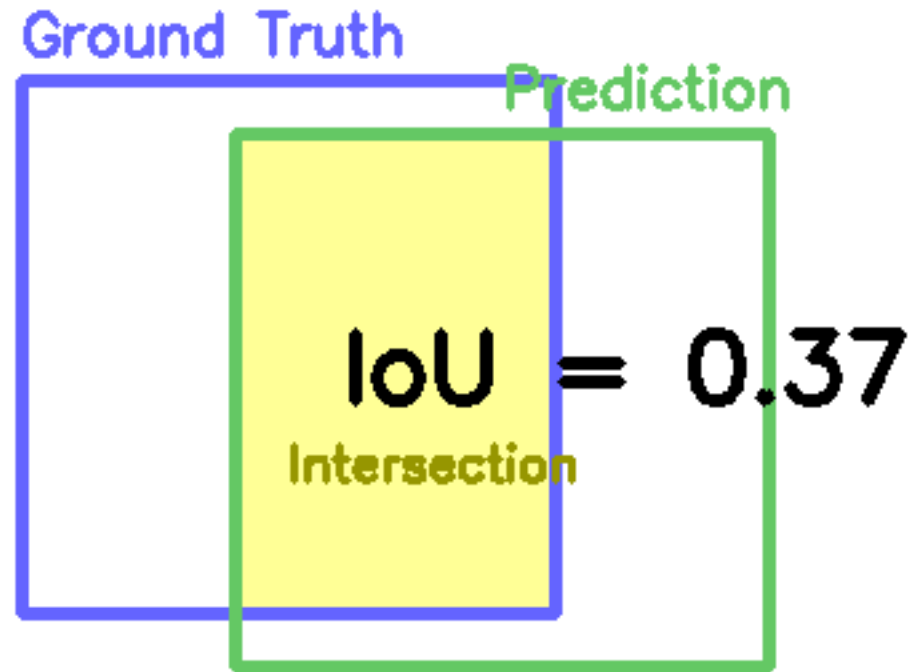
Computing IoU: Step by Step

```python
def compute_iou(box1, box2):
    """
    Boxes in format: [x1, y1, x2, y2]
    """
    # Step 1: Find intersection rectangle
    x1_inter = max(box1[0], box2[0])  # leftmost right edge
    y1_inter = max(box1[1], box2[1])  # topmost bottom edge
    x2_inter = min(box1[2], box2[2])  # rightmost left edge
    y2_inter = min(box1[3], box2[3])  # bottommost top edge

    # Step 2: Compute intersection area
    inter_width = max(0, x2_inter - x1_inter)
    inter_height = max(0, y2_inter - y1_inter)
    intersection = inter_width * inter_height

    # Step 3: Compute union area
    area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
    area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])
    union = area1 + area2 - intersection

    # Step 4: IoU = intersection / union
    return intersection / union if union > 0 else 0
```

IoU Thresholds in Practice

```
┌──────────────────────────────────────────────────────────┐
│                    IoU THRESHOLD GUIDE                     │
├───────────────────┬──────────────────────────────────────┤
│   IoU Value       │   What it means                       │
├───────────────────┼──────────────────────────────────────┤
│   1.0             │   Perfect match (never happens in practice) │
│   0.9+            │   Excellent — boxes nearly identical  │
│   0.75            │   Very good — used in strict evaluation │
│   0.50            │   Standard threshold — "correct" in most benchmarks │
│   0.25            │   Loose — used in some older benchmarks │
│   0.0             │   No overlap at all — completely wrong │
└───────────────────┴──────────────────────────────────────┘
```

```
The RULE in most systems:
    IoU ≥ 0.5  →  TRUE POSITIVE (TP) ✓  "You found the object!"
    IoU < 0.5  →  FALSE POSITIVE (FP) ✗  "Wrong detection"
```

Why 0.5 Is the Magic Number

```
Think about it visually:

IoU = 0.5 means: Intersection is 1/3 of each box's area

    ┌──────────┬──────────┐
    |    1/3   |    1/3   |    1/3
    |  A only  |  OVERLAP |    B only
    |          |          |
    └──────────┴──────────┘


    Union = 1/3 + 1/3 + 1/3 = 1
    Intersection = 1/3
    IoU = 1/3 ÷ 1 = 0.33? No wait...

Actually for IoU = 0.5:
    If Intersection = I, Union = 2I
    Because: A + B - I = 2I means A + B = 3I
    So each box is ~2I, overlapping by I

    Roughly: boxes overlap by about half their area
```
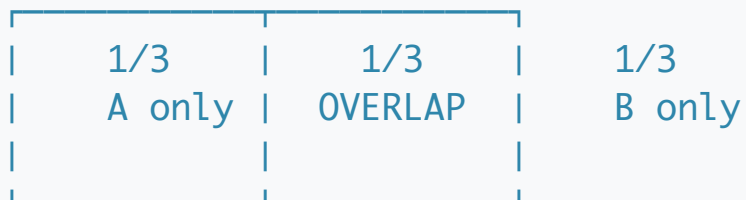
IoU = 0.5 is when the overlap is "significant enough" to say "yes, you found it"

Different Benchmarks, Different Thresholds

```
PASCAL VOC (older, easier):
    ● Single threshold: IoU ≥ 0.5
    ● One number: mAP@0.5
    ● 20 object classes

MS COCO (modern, stricter):
    ● Multiple thresholds: 0.50, 0.55, 0.60, ..., 0.95
    ● Reports: mAP@0.5, mAP@0.75, mAP@[.5:.95]
    ● 80 object classes
    ● Also evaluates different object sizes (small/medium/large)

Why COCO is harder:
    ● Must be accurate at multiple thresholds
    ● Average over 10 thresholds penalizes sloppy boxes
    ● Same model might get 60% on VOC but only 40% on COCO
```

Part 4: Non-Maximum Suppression (NMS)
Cleaning Up Duplicate Detections

The Problem: Too Many Boxes!

What the detector sees:

```
 _____
|                                                   |
|                                                   |
|        _____                        |
|       |                   |                       |
|       |  _____      |                        |
|       | |            |    |                        |
|       | |  _____   |  | |  <- Same dog, but 5 different boxes!  |
|       | | |[Dog]|  | |  | |        All have high confidence!      |
|       | | |_____|  | |  | |                                     |
|       | |            |    |                        |
|       |  _____      |                        |
|       |_____|                        |
|                                                   |
|                                                   |
|_____|
```
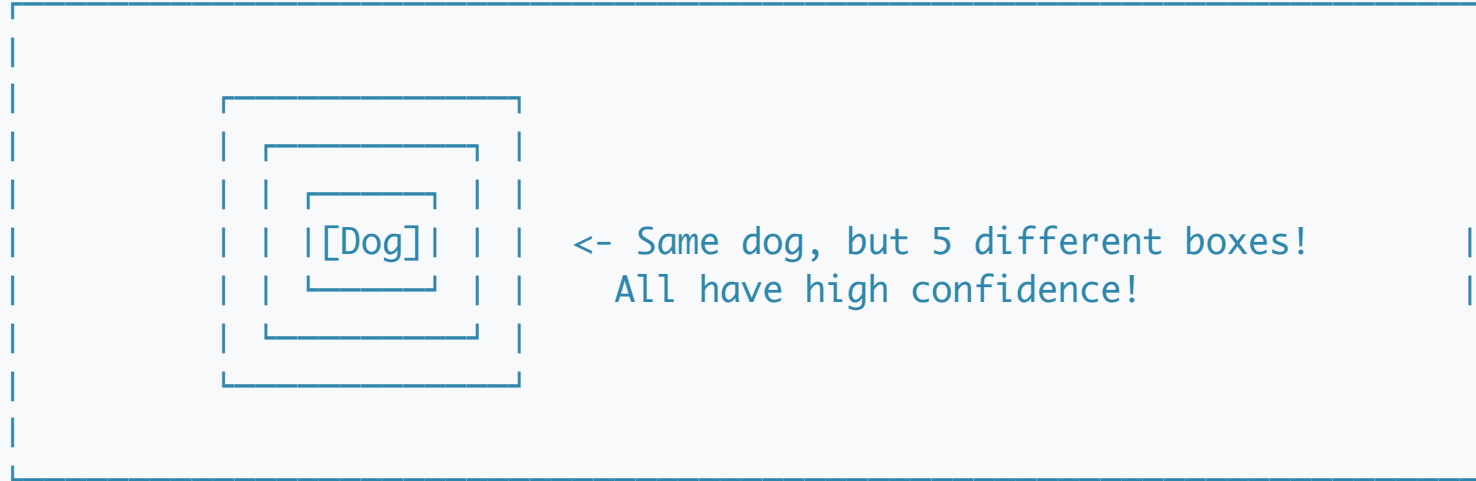
Boxes with confidences:
    Box 1: dog (0.95)
    Box 2: dog (0.93)
    Box 3: dog (0.91)
    Box 4: dog (0.87)
    Box 5: dog (0.85)

Problem: We want ONE box per object, not five!

Why Does This Happen?

```
Detectors check MANY locations:

┌─────────────────────────────────────────────────────────┐
│                                                          │
│   YOLO divides image into 7x7 = 49 cells                 │
│   Each cell predicts 2 boxes                             │
│   = 98 boxes per class                                   │
│                                                          │
│   Modern YOLO has 3 scales × (13² + 26² + 52²) × 3 anchors │
│   = ~10,000+ candidate boxes!                            │
│                                                          │
│   Many of these will fire on the same object:            │
│                                                          │
│    ┌───┬───┬───┬───┐                                     │
│    │   │ X │ X │   │    X = cells that detect the dog     │
│    ├───┼───┼───┼───┤    Each X outputs a box              │
│    │   │ X │ X │   │    All boxes are similar, all correct │
│    └───┴───┴───┴───┘    But we only want ONE               │
│                                                          │
└─────────────────────────────────────────────────────────┘
```
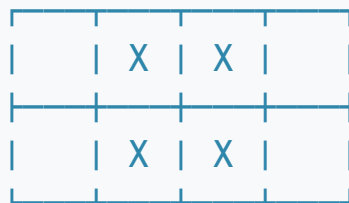
27

NMS: The Intuition

```
Non-Maximum Suppression = "Keep the best, remove the rest"

Like choosing the best photo from burst mode:

📷

📷

📷

📷

📷
→
📷
(keep best one)

Algorithm intuition:
    1. Sort all boxes by confidence (best first)
    2. Pick the best box → definitely keep it
    3. Remove all boxes that overlap too much with it
       (they're probably detecting the same object)
    4. Repeat with remaining boxes


                    ┌─────────┐
                    |0.95 |  ← Keep this (highest confidence)
                    └─────────┘

                         ↓
    Remove these if IoU > threshold:
    [0.93, 0.91, 0.87, 0.85] ← All overlapping, all removed
```

NMS: Before and After

Before NMS → NMS → After NMS

0.95

4 overlapping boxes → 1 best box kept
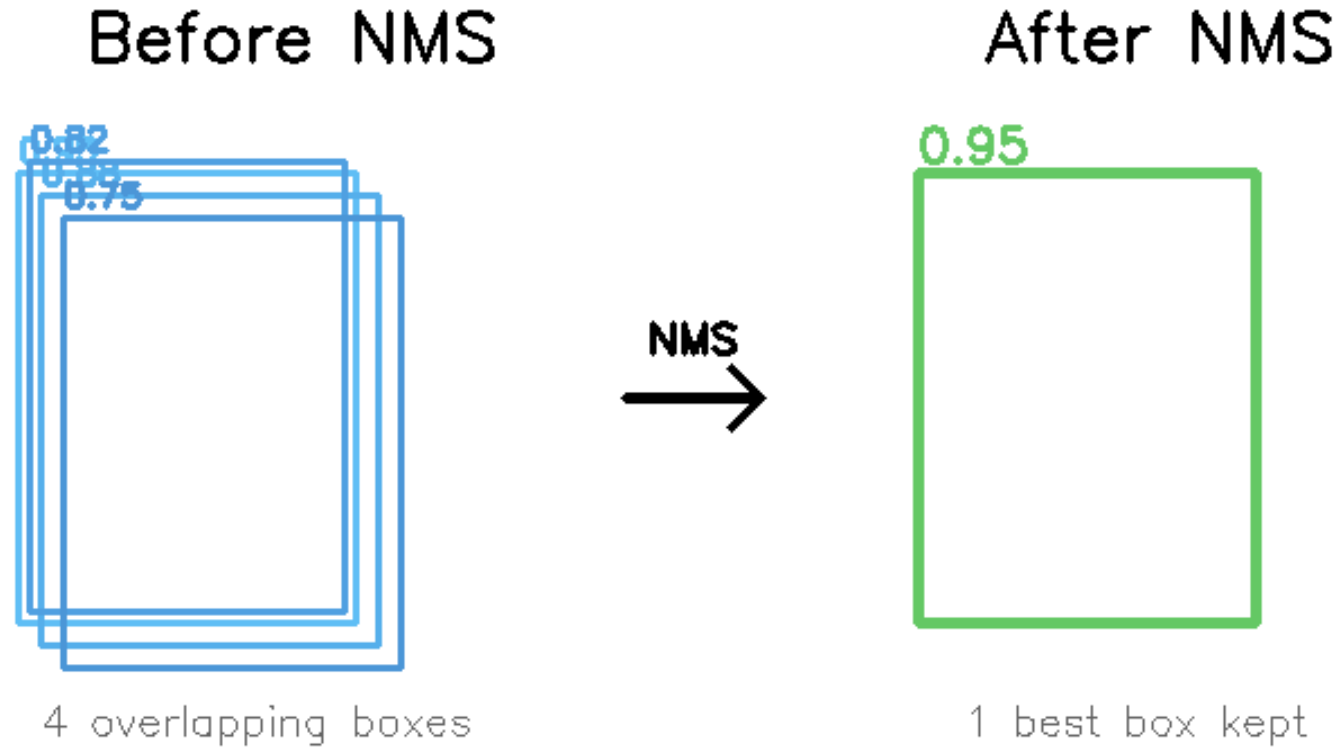
The Algorithm:

1. **Sort** all boxes by confidence (descending)

2. **Take** the best box, add to "keep" list

NMS: The Code

```python
def nms(boxes, scores, iou_threshold=0.5):
    """

    boxes: List of [x1, y1, x2, y2]
    scores: Confidence for each box
    Returns: Indices of boxes to keep
    """
    # Sort by score (descending)
    order = scores.argsort()[::-1]

    keep = []

    while len(order) > 0:
        # Take the best remaining box
        best_idx = order[0]
        keep.append(best_idx)

        # Compute IoU with all remaining boxes
        remaining = order[1:]
        ious = compute_iou_batch(boxes[best_idx], boxes[remaining])

        # Keep only boxes with low overlap (different objects)
        mask = ious <= iou_threshold
        order = remaining[mask]

    return keep
```

NMS Parameters

```
IoU THRESHOLD controls how aggressive NMS is:

    threshold = 0.3 (aggressive)
    _____

    • Removes boxes even with small overlap
    • Might accidentally merge two close objects into one
    • Use when objects are far apart

    threshold = 0.5 (standard)
    _____

    • Good balance for most cases
    • Default in most frameworks

    threshold = 0.7 (lenient)
    _____

    • Keeps more boxes
    • Better when objects overlap (crowded scenes)
    • Might keep some duplicates

Rule of thumb: Start with 0.5, adjust based on your data
```

NMS Variants

```
STANDARD NMS:
    ● Apply separately per class
    ● Simple, fast, widely used

SOFT-NMS:
    ● Instead of removing, reduce confidence of overlapping boxes
    ● Better for crowded scenes
    ● score = score × (1 - IoU) or score × exp(-IoU²/σ)

CLASS-AGNOSTIC NMS:
    ● Apply across all classes
    ● Prevents same location detecting "dog" AND "cat"

BATCHED NMS:
    ● Optimized for GPU
    ● Process all classes in parallel

DIoU-NMS:
    ● Uses Distance-IoU instead of IoU
    ● Considers center distance, not just overlap
    ● Better for overlapping objects
```

Part 5: How Detectors Work
The YOLO Architecture

Two Families of Detectors

```
TWO-STAGE DETECTORS            ONE-STAGE DETECTORS
(Accurate but slower)          (Fast, good enough)

Stage 1: "Where might             Single pass:
         objects be?"             "Here are all objects
Stage 2: "What are they?"          with their locations"


|Image| →  | RPN | →  |Class|    |Image| →  | All |

         ↓                                  ↓
     ~2000 boxes                      Direct output
         ↓                                  ↓
     Classify each                   Boxes + Classes

Examples:                         Examples:
● R-CNN, Fast R-CNN               ● YOLO (v1-v8)
● Faster R-CNN                    ● SSD
● Mask R-CNN                      ● RetinaNet

~5-10 FPS                         ~30-100+ FPS
```
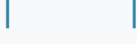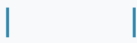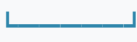
YOLO: You Only Look Once

```
The revolutionary idea (2015):

BEFORE YOLO:                      YOLO:
_____          _____

Run classifier 1000s of times     Run CNN ONCE
at different locations/scales      Output everything directly


  ┌──────┐   → Is there object here?     ┌──────┐
  │      │   → Is there object here?     │      │   →  All boxes +
  │ IMG  │   → Is there object here?     │ IMG  │      All classes +
  │      │   → Is there object here?     │      │      All confidences
  └──────┘   → ... (repeat 1000x)        └──────┘      (ONE forward pass)


Very slow (seconds per image)     Very fast (milliseconds)
```
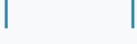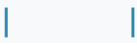
YOLO = "You Only Look Once" — the entire detection in a single neural network pass

YOLO Core Idea: Grid Division

YOLO divides the image into an S×S grid (e.g., 7×7):
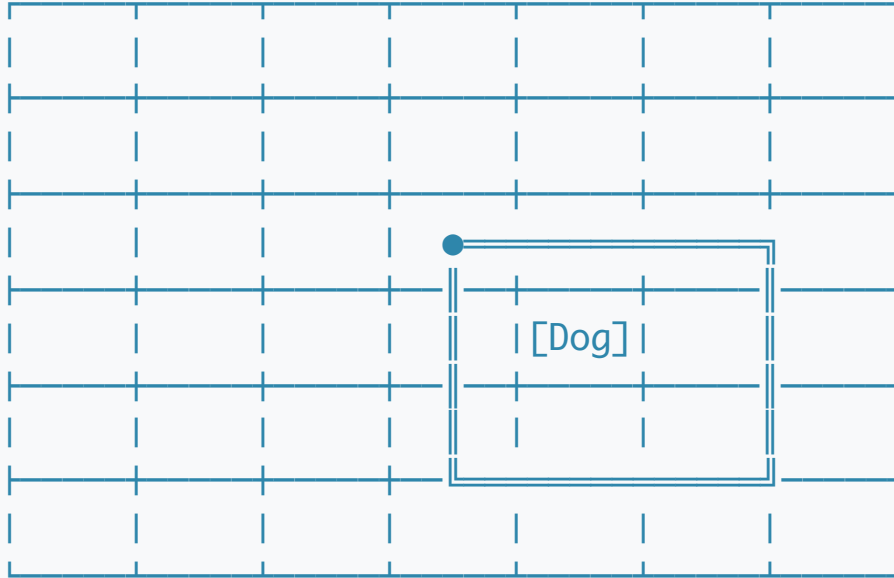

```
                            ↑
        The cell containing the CENTER of the dog
        is "responsible" for detecting that dog
```

[Dog]

What Each Cell Predicts

Each grid cell outputs:

```
For EACH BOUNDING BOX (B boxes per cell):
─────────────────────────────────────────

    • x, y      = center offset (relative to cell)
    • w, h      = width, height (relative to image)
    • conf      = P(object) × IoU with ground truth

For the CELL (shared across boxes):
────────────────────────────────────

    • P(class₁), P(class₂), ..., P(classC)

OUTPUT SHAPE:
──────────────────────────────

  S × S × (B × 5 + C)

  Example: 7 × 7 × (2 × 5 + 20) = 7 × 7 × 30
           = 1470 numbers describing all detections
```

YOLO Prediction Visualization

```
One cell's prediction (2 boxes, 20 classes):

        Box 1                    Box 2                    Classes
   ┌──────────────┐         ┌──────────────┐         ┌──────────────────┐
   | x: 0.3       |         | x: 0.6       |         | P(person): 0.01  |
   | y: 0.4       |         | y: 0.5       |         | P(car): 0.02     |
   | w: 0.5       |         | w: 0.3       |         | P(dog): 0.92     |
   | h: 0.7       |         | h: 0.4       |         | P(cat): 0.03     |
   | conf: 0.85   |         | conf: 0.12   |         | ...              |
   └──────────────┘         └──────────────┘         └──────────────────┘

         ↓                        ↓
     High conf!               Low conf,                      ↓
     This is the              ignore              Most likely: DOG
     main detection

Final: Box 1 predicts "dog" with confidence 0.85 × 0.92 = 0.78
```

The YOLO Network Architecture (v1)

```
┌────────────────────────────────────────────────────────┐
│                  YOLO v1 Architecture                  │
├────────────────────────────────────────────────────────┤
│                                                        │
│  INPUT: 448 × 448 × 3 (RGB image)                      │
│     |                                                  │
│     ▼                                                  │
│  ┌──────────────────────────────────────┐             │
│  │  24 CONVOLUTIONAL LAYERS             │             │
│  │  (Feature extraction - like image filters) │        │
│  │                                      │             │
│  │  Conv → Pool → Conv → Pool → ... → Conv   │          │
│  └──────────────────────────────────────┘             │
│                                                        │
│     |                                                  │
│     ▼                                                  │
│  ┌──────────────────────────────────────┐             │
│  │  2 FULLY CONNECTED LAYERS            │             │
│  │  (Turn features into predictions)    │             │
│  │                                      │             │
│  │  4096 → 7 × 7 × 30                   │             │
│  └──────────────────────────────────────┘             │
│                                                        │
│     |                                                  │
│     ▼                                                  │
│  OUTPUT: 7 × 7 × 30 tensor (all predictions)          │
│                                                        │
└────────────────────────────────────────────────────────┘
```
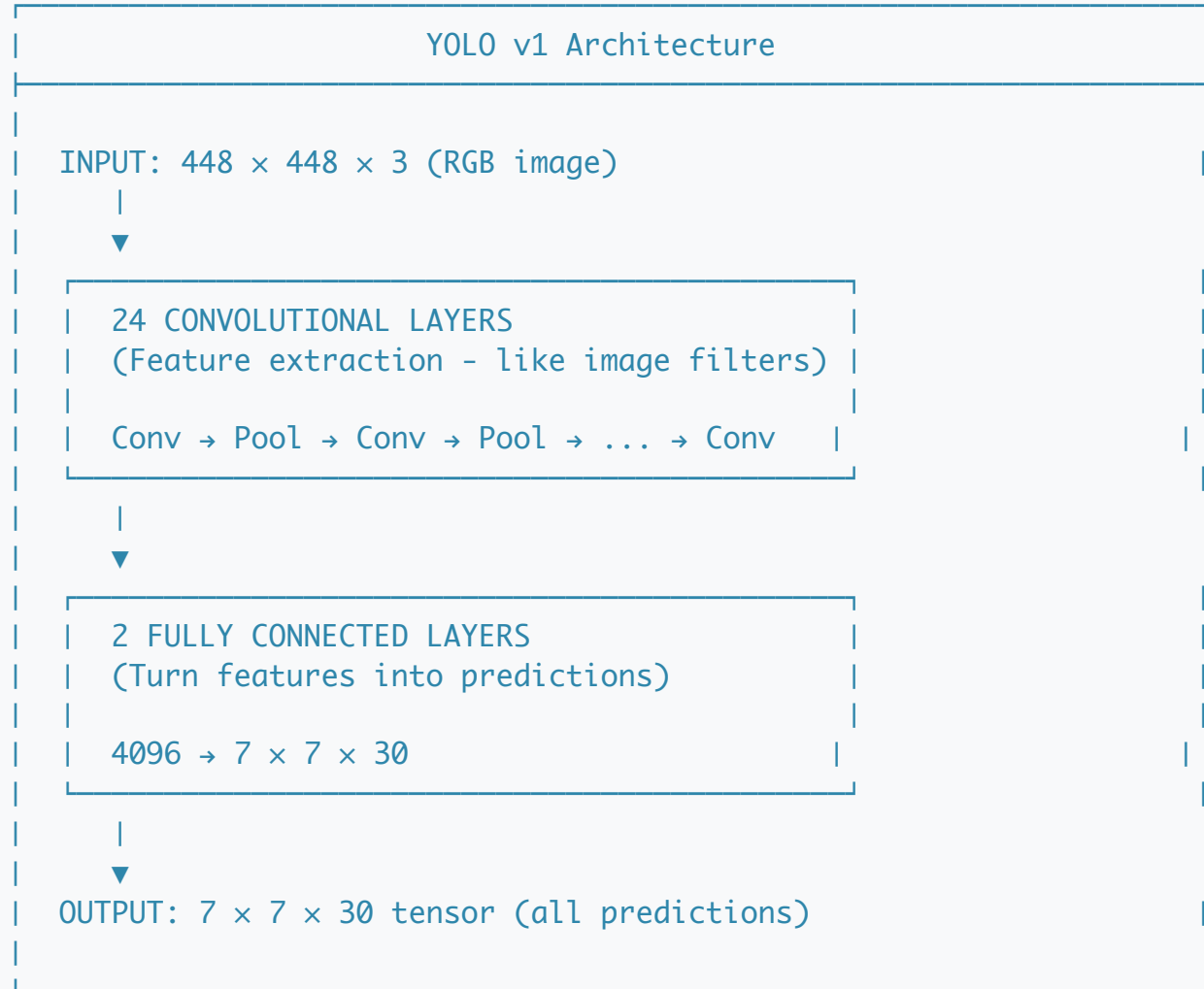
Anchor Boxes: Handling Multiple Objects Per Cell

```
PROBLEM: What if TWO objects have centers in the SAME cell?

    ┌───┬───┬───┐
    │   │   │   │
    ├───┼───┼───┤
    │   │D C│   │  <- Both dog AND cat center here!
    ├───┼───┼───┤      But cell can only predict
    │   │   │   │      one class in YOLO v1
    └───┴───┴───┘


SOLUTION: Anchor boxes (YOLO v2+)

    Pre-define box SHAPES (anchors):

    ┌───┐   ┌───────┐   ┌─┐
    │   │   │       │   │ │
    │   │   └───────┘   │ │
    └───┘               │ │
                        │ │
    Tall    Wide        │ │
    anchor  anchor      └─┘
                        Very tall

    Each anchor predicts separately!
    Now cell can detect person (tall) AND car (wide)
```
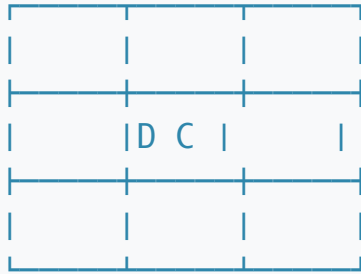
Anchor Boxes: How They Work

```
Each cell has K anchor boxes (e.g., K=5 in YOLOv2, K=9 in YOLOv3)

 ┌─────────────────────────────────────────────────────────────┐
 │                                                               │
 │  ANCHOR SHAPES (learned from data using K-means clustering):  │
 │                                                               │
 │   Anchor 1     Anchor 2     Anchor 3     Anchor 4    Anchor 5 │
 │   (small)      (tall)       (wide)       (medium)    (large)  │
 │                                                               │
 │    ┌──┐         ┌──┐       ┌─────────┐   ┌───┐      ┌─────┐   │
 │    │  │         │  │       │         │   │   │      │     │   │
 │    └──┘         │  │       └─────────┘   │   │      │     │   │
 │                 │  │                     └───┘      │     │   │
 │                 └──┘                                └─────┘   │
 │                                                               │
 │  PREDICTION: Instead of predicting raw (x,y,w,h),             │
 │              predict OFFSETS from anchor:                     │
 │                                                               │
 │              predicted_w = anchor_w × exp(t_w)                │
 │              predicted_h = anchor_h × exp(t_h)                │
 │                                                               │
 │  This makes learning easier! Network just learns adjustments. │
 │                                                               │
 └─────────────────────────────────────────────────────────────┘
```

YOLO Evolution: v1 → v8

YOLOv1 (2015): Original breakthrough
- 7×7 grid, 2 boxes/cell, 45 FPS
- Simple but limited

YOLOv2 (2016): Better and faster
- Anchor boxes, batch normalization
- Multi-scale training

YOLOv3 (2018): Big improvement
- 3 scales (13×13, 26×26, 52×52)
- Better small object detection
- Darknet-53 backbone

YOLOv4 (2020): Bag of tricks
- Many augmentations (Mosaic, CutMix)
- CSPDarknet backbone

YOLOv5-v8 (2020-2023): Ultralytics versions
- PyTorch native
- Easy to use API
- State-of-the-art speed/accuracy

Modern YOLO (v5/v8): Key Features

```
┌─────────────────────────────────────────────────────────────┐
│                    YOLO v5/v8 INNOVATIONS                     │
├─────────────────────────────────────────────────────────────┤
│                                                              │
│  1. MULTI-SCALE DETECTION                                    │
│     • Small objects: 52×52 grid (detailed)                   │
│     • Medium objects: 26×26 grid                             │
│     • Large objects: 13×13 grid (coarse)                     │
│                                                              │
│  2. FEATURE PYRAMID NETWORK (FPN)                            │
│     • Combines low-level (edges) and high-level (semantics)  │
│     • Better detection at all sizes                          │
│                                                              │
│  3. PATH AGGREGATION NETWORK (PAN)                           │
│     • Bottom-up path for better localization                 │
│                                                              │
│  4. MOSAIC AUGMENTATION                                      │
│     • Combine 4 images → more objects per batch              │
│     • Better generalization                                  │
│                                                              │
│  5. AUTO-ANCHOR                                              │
│     • Automatically compute best anchors for your data       │
│                                                              │
└─────────────────────────────────────────────────────────────┘
```

YOLO Model Sizes

```
+-----------------------------------------------------------------------+
|                      YOLOv8 Model Variants                            |
+-----------------------------------------------------------------------+
|  Model   |  Parameters  |   Size    |  mAP (COCO)  |   Speed (GPU)     |
+-----------------------------------------------------------------------+
|  v8n     |      3.2M     |    6 MB   |     37.3     |   ~1.0 ms         |
|  v8s     |     11.2M     |   22 MB   |     44.9     |   ~1.5 ms         |
|  v8m     |     25.9M     |   52 MB   |     50.2     |   ~2.5 ms         |
|  v8l     |     43.7M     |   87 MB   |     52.9     |   ~4.0 ms         |
|  v8x     |     68.2M     |  136 MB   |     53.9     |   ~6.0 ms         |
+-----------------------------------------------------------------------+
```

```
n = nano (edge devices, phones)
s = small (good balance)
m = medium
l = large
x = extra large (best accuracy)
```

Start with YOLOv8n for prototyping, move to larger models if needed.

Part 6: Training & Evaluation
How We Train and Measure Detectors

The Detection Loss Function

Detection models optimize MULTIPLE objectives simultaneously:

```
TOTAL LOSS = λ₁·L_box + λ₂·L_obj + λ₃·L_class

  L_box (Localization Loss)
  "Is the box in the right place?"
  ● Mean squared error on (x, y, w, h)
  ● Or IoU-based loss (GIoU, DIoU, CIoU)


  L_obj (Objectness Loss)
  "Is there an object here?"
  ● Binary cross-entropy
  ● Confidence should match IoU with ground truth


  L_class (Classification Loss)
  "What class is it?"
  ● Cross-entropy for each class
```

Understanding the Box Loss

```
ORIGINAL YOLO (MSE Loss):
─────────────────────────────────────────────

L_box = (x - x̂)² + (y - ŷ)² + (√w - √ŵ)² + (√h - √ĥ)²

Why square root of w,h?
    ● Large boxes shouldn't dominate
    ● Error of 10px matters more for 20px box than 200px box
    ● √ compresses the range

MODERN YOLO (IoU-based Loss):
─────────────────────────────────────────────

L_box = 1 - IoU(predicted, ground_truth)

Or better variants:
    ● GIoU: Adds penalty for gap between boxes
    ● DIoU: Adds penalty for center distance
    ● CIoU: DIoU + aspect ratio penalty

These directly optimize what we care about: overlap!
```

Precision and Recall: Detection Version

```
For DETECTION, we need to define TP/FP/FN differently:

TRUE POSITIVE (TP):
    ● Detection exists
    ● Matches a ground truth box with IoU ≥ threshold
    ● Correct class

FALSE POSITIVE (FP):
    ● Detection exists BUT:
      - Doesn't match any ground truth (IoU too low)
      - OR correct box but wrong class
      - OR duplicate detection (already matched)

FALSE NEGATIVE (FN):
    ● Ground truth box exists
    ● No detection matched it
```

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

"Of my detections,   "Of the actual objects,
 how many are correct?"   how many did I find?"

Precision vs Recall Trade-off

```
As we lower the CONFIDENCE THRESHOLD:

High threshold (0.9):                Low threshold (0.3):
_____                 _____

● Only very confident detections     ● Many detections (even uncertain)
● Might miss some objects            ● Might include wrong detections
● HIGH PRECISION, LOW RECALL         ● LOW PRECISION, HIGH RECALL


Example on 10 real dogs:

Threshold=0.9:                       Threshold=0.3:
● Found: 4 dogs (all correct)        ● Found: 12 "dogs" (9 correct, 3 wrong)
● Precision = 4/4 = 100%             ● Precision = 9/12 = 75%
● Recall = 4/10 = 40%                ● Recall = 9/10 = 90%
```

```
┌─────────────────────────────────────────────────┐
│    P                                             │
│    r  |*                                         │
│    e  | *                                        │
│    c  |  *                                       │
│    i  |   *    ← Precision-Recall Curve          │
│    s  |    **                                    │
│    i  |      ***                                 │
│    o  |       ****                               │
│    n  └──────────────*******────────────── Recall│
│                                                  │
└─────────────────────────────────────────────────┘
```
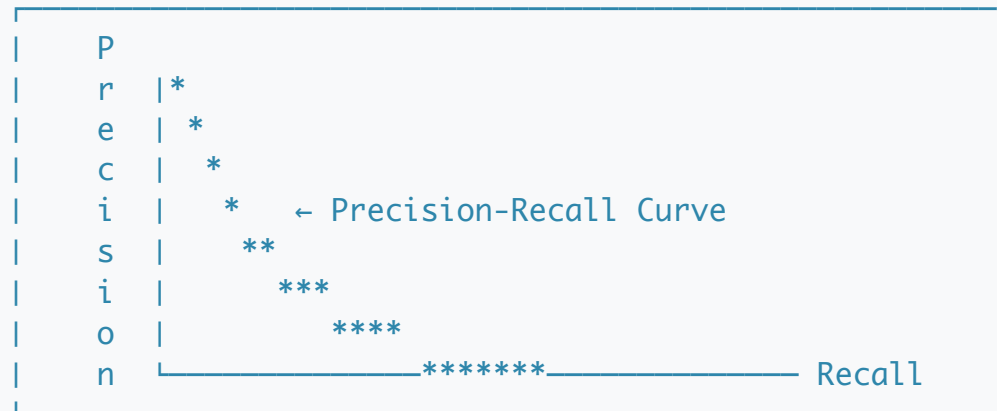
Average Precision (AP)

```
AP = Area under the Precision-Recall curve

   Precision
   1.0 ┤███████
       |█████████████
       |███████████████████
       |█████████████████████████
   0.5 ┤████████████████████████████████
       |█████████████████████████████████████
       |██████████████████████████████████████████
   0.0 ┴────────────────────────────────────
       0.0                            1.0
              Recall
```
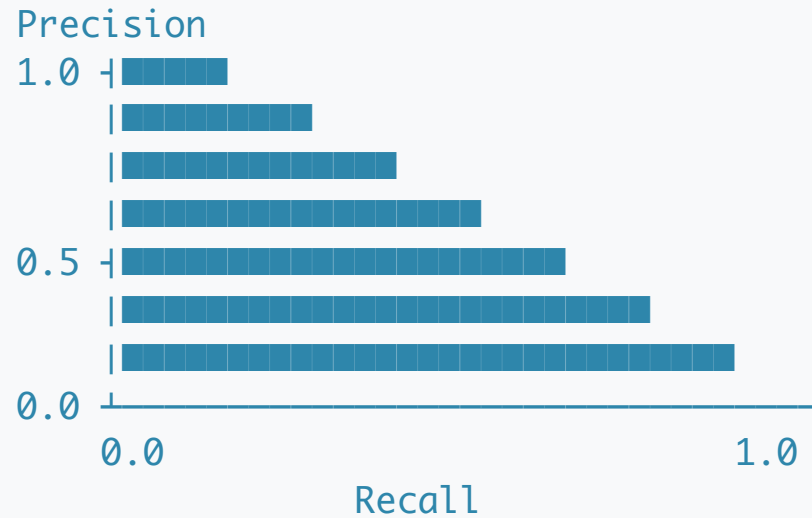
AP = Area of this curve (shaded region)

Higher AP = better detector (good precision at all recall levels)

In practice, we compute AP by:
1. Sort detections by confidence
2. At each detection, compute precision and recall so far
3. Interpolate and compute area under curve

Mean Average Precision (mAP)

```
mAP = Mean of AP across all classes

┌─────────────────────────────────────────────────────┐
│                                                       │
│   For each class:                                     │
│   ───────────────                                     │
│                                                       │
│   AP_person = 0.85                                    │
│   AP_car = 0.92                                        │
│   AP_dog = 0.78                                        │
│   AP_cat = 0.71                                        │
│                                                       │
│   ...                                                 │
│   AP_class_80 = 0.65                                  │
│                                                       │
│                        AP₁ + AP₂ + ... + APc          │
│   mAP@0.5 = ──────────────────────────────────        │
│                        C (number of classes)         │
│                                                       │
│   COCO also reports:                                  │
│   ● mAP@0.75 (stricter IoU threshold)                 │
│   ● mAP@[0.5:0.95] (average over 10 thresholds: 0.5, 0.55, ..., 0.95)│
│                                                       │
└─────────────────────────────────────────────────────┘
```

$$mAP@0.5 = \frac{AP_1 + AP_2 + ... + AP_c}{C \text{ (number of classes)}}$$

mAP Calculation Example

```
Let's compute mAP step by step:

Dataset: 5 images with dogs and cats
Ground truth: 3 dogs, 2 cats

Model predictions (sorted by confidence):

  ┌─────┬────────┬────────┬──────────┬─────────┐
  | #   | Class  | Conf   | Matched? | Status |
  ├─────┼────────┼────────┼──────────┼─────────┤
  | 1   | dog    | 0.95   | Yes      | TP      |
  | 2   | dog    | 0.90   | Yes      | TP      |
  | 3   | cat    | 0.85   | Yes      | TP      |
  | 4   | dog    | 0.80   | No       | FP      |
  | 5   | cat    | 0.75   | Yes      | TP      |
  | 6   | dog    | 0.70   | Yes      | TP      |
  └─────┴────────┴────────┴──────────┴─────────┘


For dogs: TP=3, FP=1, FN=0 (found all 3)
For cats: TP=2, FP=0, FN=0 (found all 2)

AP_dog = (compute P-R curve area)
AP_cat = (compute P-R curve area)
mAP = (AP_dog + AP_cat) / 2
```

Data Augmentation for Detection

```
IMPORTANT: When transforming the image, ALSO transform the boxes!

 ------------------------------------------------------------
|                                                            |
|  HORIZONTAL FLIP:                                          |
|  ---------------                                           |
|   Original            Flipped                              |
|    -------------       -------------                       |
|   |  ----      |      |        ----  |                     |
|   | |Dog|      |  ->  |        |Dog| |                     |
|   |  ----      |      |        ----  |                     |
|    -------------       -------------                       |
|                                                            |
|   Box: [10,20,60,80]   Box: [W-60, 20, W-10, 80]          |
|                                                            |
|  SCALE/RESIZE:                                             |
|  ------------                                              |
|   Scale factor = s                                         |
|   New box: [x*s, y*s, w*s, h*s]                            |
|                                                            |
|  CROP:                                                     |
|  -----                                                     |
|   Remove boxes that fall outside crop                      |
|   Clip boxes that partially overlap                        |
|   Adjust coordinates relative to crop origin               |
|                                                            |
 ------------------------------------------------------------
```
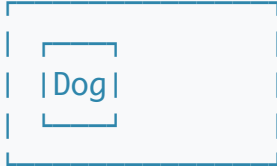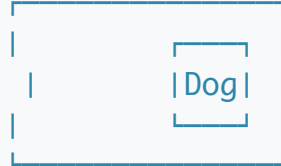
Mosaic Augmentation (YOLO v4+)

```
Combine 4 images into 1 training sample:

    ┌───────────────┬───────────────┐
    │    Image 1    │    Image 2    │
    │ D1    D2      │      C1       │
    │               │    D3         │
    ├───────────────┼───────────────┤
    │    Image 3    │    Image 4    │
    │   C2          │    D4 C3      │
    │      C4       │               │
    └───────────────┴───────────────┘

            ↓
       Single training image with
       4× the objects!

BENEFITS:
● More objects per batch
● Objects in unusual contexts
● Better generalization
● Reduces need for large batch sizes
```

Training Tips for Detection

```
┌──────────────────────────────────────────────────────┐
│              PRACTICAL TRAINING TIPS                   │
├──────────────────────────────────────────────────────┤
│                                                        │
│  1. START WITH PRETRAINED WEIGHTS                      │
│      • Use COCO pretrained model                       │
│      • Fine-tune on your dataset                       │
│      • Much faster than training from scratch          │
│                                                        │
│  2. BALANCE YOUR DATASET                               │
│      • Roughly equal examples per class                │
│      • Or use focal loss to handle imbalance           │
│                                                        │
│  3. ANCHOR BOX ANALYSIS                                │
│      • Run k-means on your box sizes                   │
│      • Use auto-anchor feature in YOLOv5/v8            │
│                                                        │
│  4. IMAGE SIZE MATTERS                                 │
│      • Larger images = better small object detection   │
│      • But slower training/inference                  │
│      • 640×640 is common default                       │
│                                                        │
│  5. AUGMENTATION IS KEY                                │
│      • Mosaic, MixUp, color jitter                     │
│      • More augmentation = better generalization       │
│                                                        │
└──────────────────────────────────────────────────────┘
```

Common Detection Benchmarks

```
┌─────────────────────────────────────────────────────────────┐
│                 POPULAR DATASETS & BENCHMARKS                 │
├──────────────────┬──────────────────────────────────────────┤
│                  │                                          │
│  PASCAL VOC      │  ● 20 classes, ~10K images               │
│  (2007, 2012)    │  ● Classic benchmark                     │
│                  │  ● mAP@0.5 metric                        │
│                  │                                          │
├──────────────────┼──────────────────────────────────────────┤
│                  │                                          │
│  MS COCO         │  ● 80 classes, ~120K images              │
│  (2014-present)  │  ● Modern standard benchmark             │
│                  │  ● mAP@[0.5:0.95] metric (harder)        │
│                  │  ● Small/medium/large object splits      │
│                  │                                          │
├──────────────────┼──────────────────────────────────────────┤
│                  │                                          │
│  Open Images     │  ● 600 classes, 1.7M images              │
│                  │  ● Largest, most diverse                 │
│                  │                                          │
├──────────────────┼──────────────────────────────────────────┤
│                  │                                          │
│  Custom          │  ● Your domain-specific data             │
│                  │  ● Label with Roboflow, CVAT, LabelImg   │
│                  │                                          │
└──────────────────┴──────────────────────────────────────────┘
```

State-of-the-Art Results (2024)

```
MS COCO leaderboard (mAP@[0.5:0.95]):


┌─────────────────────────┬─────────┬─────────┬───────────┐
│  Model                  │  mAP    │  Params │  Speed    │
├─────────────────────────┼─────────┼─────────┼───────────┤
│  DINO (Transformer-based) │  63.3 │  218M   │  Slow     │
│  Focal-DINO             │  58.4   │   -     │  Slow     │
│  Co-DETR                │  66.0   │  218M   │  Slow     │
│  YOLOv8-X               │  53.9   │  68M    │  Fast     │
│  YOLOv9-E               │  55.6   │  60M    │  Fast     │
│  YOLO-World             │  45.0*  │  46M    │  Fast     │
└─────────────────────────┴─────────┴─────────┴───────────┘

   * Zero-shot (no COCO training)

Key insight: Transformer detectors (DINO, DETR) now beat CNNs in accuracy
but YOLO family still dominates real-time applications.
```

Complete Detection Pipeline

```
┌─────────────────────────────────────────────────┐
│              END-TO-END DETECTION FLOW            │
├─────────────────────────────────────────────────┤
│                                                   │
│  INPUT: Image (640×640×3)                         │
│      |                                            │
│      ▼                                            │
│   ┌──────────────────────────────────┐            │
│   │  BACKBONE: Extract features      │            │
│   │  (Conv layers, ResNet, CSPDarknet, etc.)      │
│   └──────────────────────────────────┘            │
│                                                   │
│      |                                            │
│      ▼                                            │
│   ┌──────────────────────────────────┐            │
│   │  NECK: Combine multi-scale features           │
│   │  (FPN, PAN, BiFPN)               │            │
│   └──────────────────────────────────┘            │
│                                                   │
│      |                                            │
│      ▼                                            │
│   ┌──────────────────────────────────┐            │
│   │  HEAD: Predict boxes + classes at each scale  │
│   │  Output: ~8000 candidate boxes   │            │
│   └──────────────────────────────────┘            │
│                                                   │
│      |                                            │
│      ▼                                            │
│   ┌──────────────────────────────────┐            │
│   │  POST-PROCESSING:                │            │
│   │  1. Filter by confidence (> 0.25) │           │
│   │  2. Apply NMS per class          │            │
│   │  3. Output final detections      │            │
│   └──────────────────────────────────┘            │
│                                                   │
│      |                                            │
│      ▼                                            │
│  OUTPUT: List of [class, confidence, x1, y1, x2, y2]  │
│                                                   │
└─────────────────────────────────────────────────┘
```

Getting Started: YOLO in 3 Lines

```python
# Install: pip install ultralytics

from ultralytics import YOLO

# Load a pretrained model
model = YOLO('yolov8n.pt')

# Run detection on an image
results = model('your_image.jpg')

# Display results
results[0].show()

# Access detections programmatically
for box in results[0].boxes:
    cls = int(box.cls[0])          # Class index
    conf = float(box.conf[0])      # Confidence
    xyxy = box.xyxy[0].tolist()    # [x1, y1, x2, y2]

    print(f"Class: {model.names[cls]}, Conf: {conf:.2f}")
    print(f"Box: {xyxy}")
```

That's it! You now have a working object detector. Try it on your own photos!

Training on Custom Data

```python
# Step 1: Prepare your dataset (YOLO format)
# data/
#   images/
#     train/  val/
#   labels/
#     train/  val/
# Each label file: class_id cx cy w h (normalized)

# Step 2: Create data.yaml
"""
path: ./data
train: images/train
val: images/val
names:
  0: cat
  1: dog
  2: bird
"""

# Step 3: Train!
from ultralytics import YOLO

model = YOLO('yolov8n.pt')  # Start from pretrained

results = model.train(
    data='data.yaml',
    epochs=50,
    imgsz=640,
    batch=16
)

# Step 4: Use your trained model
model = YOLO('runs/detect/train/weights/best.pt')
model.predict('test_image.jpg')
```

Summary: Key Takeaways

```
┌─────────────────────────────────────────────────────────────────┐
│                          KEY CONCEPTS                           │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  1. DETECTION = CLASSIFICATION + LOCALIZATION                   │
│     Find WHAT objects and WHERE they are                        │
│                                                                 │
│  2. BOUNDING BOX FORMATS VARY                                   │
│     Always check: corner vs center, absolute vs normalized      │
│                                                                 │
│  3. IoU MEASURES OVERLAP QUALITY                                │
│     IoU ≥ 0.5 is standard threshold for "correct"              │
│                                                                 │
│  4. NMS REMOVES DUPLICATES                                       │
│     Keep highest confidence, remove overlapping                 │
│                                                                 │
│  5. YOLO IS FAST AND PRACTICAL                                  │
│     One-stage detection, grid-based, real-time capable          │
│                                                                 │
│  6. mAP IS THE GOLD STANDARD METRIC                            │
│     Average precision across classes and IoU thresholds         │
│                                                                 │
│  7. START WITH PRETRAINED MODELS                               │
│     Fine-tune on your data for best results                     │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

Resources for Learning More

PAPERS:
───

- YOLO v1: "You Only Look Once" (Redmon et al., 2015)
- YOLO v3: "YOLOv3: An Incremental Improvement" (Redmon, 2018)
- Faster R-CNN: "Faster R-CNN" (Ren et al., 2015)
- DETR: "End-to-End Object Detection with Transformers" (2020)

CODE & TUTORIALS:
───

- Ultralytics YOLOv8: https://github.com/ultralytics/ultralytics
- Roboflow Blog: https://blog.roboflow.com/
- PyTorch Detection: https://pytorch.org/vision/stable/models.html

DATASETS:
───

- COCO: https://cocodataset.org/
- Open Images: https://storage.googleapis.com/openimages/
- Roboflow Universe: https://universe.roboflow.com/ (100K+ datasets)

What's Next?

```
NOW YOU CAN:
_____

✓ Understand how object detection differs from classification
✓ Work with bounding boxes in different formats
✓ Implement and understand IoU calculation
✓ Apply Non-Maximum Suppression
✓ Use YOLO for real-time detection
✓ Evaluate detectors with mAP

NEXT STEPS:
_____

→ Try detection on your own images/videos
→ Fine-tune YOLO on a custom dataset
→ Explore instance segmentation (Mask R-CNN, YOLOv8-seg)
→ Learn about transformer-based detectors (DETR, DINO)
→ Deploy detection models on edge devices
```

Thank You!

Object Detection opens the door to...

- Self-driving cars
- Augmented reality
- Medical diagnosis
- Smart retail
- Security systems
- Robotics

And so much more!

Questions?

Appendix: Common Pitfalls

```
┌──────────────────────────────────────────────────────────────┐
│                    THINGS THAT GO WRONG                       │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  1. WRONG BOX FORMAT                                         │
│     ● Model expects YOLO format, you gave COCO format        │
│     ● Solution: Always verify format in documentation        │
│                                                              │
│  2. FORGETTING TO NORMALIZE                                  │
│     ● Pixel coordinates don't work for different image sizes │
│     ● Solution: Use normalized coordinates or resize consistently │
│                                                              │
│  3. NMS THRESHOLD TOO AGGRESSIVE                             │
│     ● Two close objects merged into one detection            │
│     ● Solution: Increase NMS threshold (0.5 → 0.7)           │
│                                                              │
│  4. SMALL OBJECTS MISSED                                     │
│     ● Default models optimized for medium/large objects      │
│     ● Solution: Use larger image size, or specialized models │
│                                                              │
│  5. CLASS IMBALANCE                                          │
│     ● 1000 cars, 10 motorcycles → model ignores motorcycles  │
│     ● Solution: Oversample rare classes, use focal loss      │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

Appendix: IoU Loss Variants

```
┌─────────────────────────────────────────────────────────┐
│                 IoU-BASED LOSS FUNCTIONS                │
├─────────────────────────────────────────────────────────┤
│                                                         │
│  Standard IoU Loss:                                     │
│      L = 1 - IoU                                        │
│      Problem: No gradient when boxes don't overlap      │
│                                                         │
│  GIoU (Generalized IoU):                                │
│      L = 1 - IoU + |C - Union| / |C|                    │
│      C = smallest enclosing box                         │
│      Handles non-overlapping boxes!                     │
│                                                         │
│  DIoU (Distance IoU):                                   │
│      L = 1 - IoU + d² / c²                              │
│      d = center distance, c = diagonal of enclosing box │
│      Faster convergence                                 │
│                                                         │
│  CIoU (Complete IoU):                                   │
│      L = 1 - IoU + d² / c² + αv                         │
│      α, v = aspect ratio penalty terms                  │
│      Best overall performance                           │
│                                                         │
└─────────────────────────────────────────────────────────┘
```