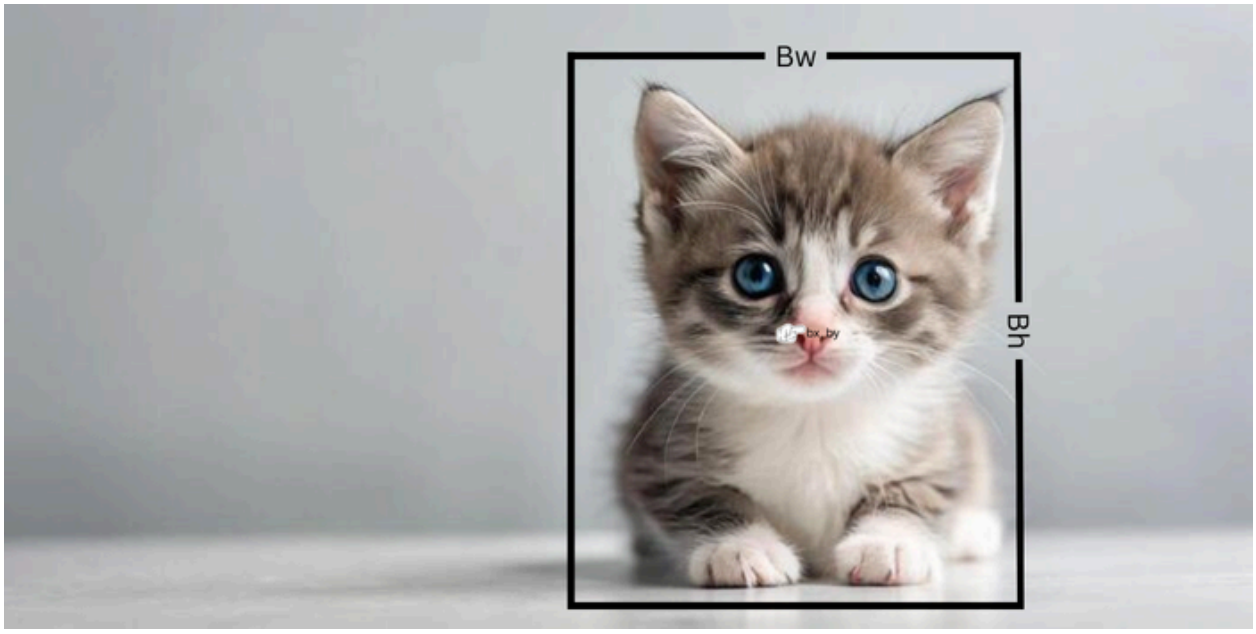
Object Localization:



output prediction vector:

[Pc, bx, by, bh, bw, c1, c2, c3]

🔹 Meaning of Each Component

Symbol	What it represents	Type
Pc	Objectness score — probability that an object exists in this cell	Scalar
bx	Predicted bounding box **center x** relative to the cell	Scalar
by	Predicted bounding box **center y** relative to the cell	Scalar
bh	Predicted bounding box **height**	Scalar
bw	Predicted bounding box **width**	Scalar
c1, c2, c3	Class probabilities (e.g., 3 classes)	Scalars

So this vector has **1 + 4 + 3 = 8 values**.

If Pc = 1

Pc = 1
bx = 0.4
by = 0.6
bw = 0.3
bh = 0.5
[c1, c2, c3] = [0.1, 0.7, 0.2] → Class 2 highest

If Pc = 0
Pc = 0
bx, by, bw, bh = don't matter
[c1, c2, c3] = don't matter

Square loss according to YOLOv1:

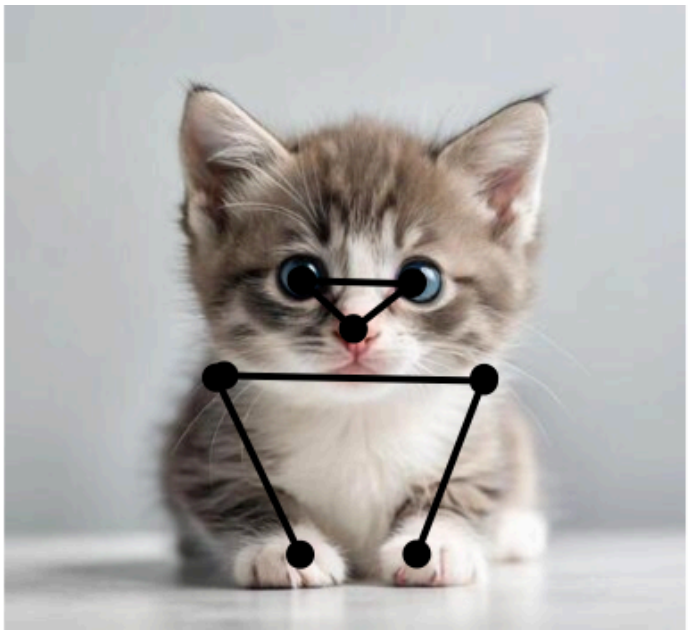
$$\text{Loss} = \lambda_{coord} * \Sigma[(b_x - \hat{b}_x)^2 + (b_y - \hat{b}_y)^2 + (b_w - \hat{b}_w)^2 + (b_h - \hat{b}_h)^2] + (P_c - \hat{P}_c)^2 + \Sigma_{classes}(c_i - \hat{c}_i)^2$$

Where:

Symbol	Meaning
$\lambda coord$	Extra weight for box accuracy
bx, by, bw, bh	Predicted box
Pc	Predicted object presence
ci	Predicted class probabilities
^	Ground truth

Condition	What Loss Applies?	Why
$P\hat{c} = 1$ (object exists)	Box loss + Class loss + Pc loss	Model must learn full prediction
$P\hat{c} = 0$ (no object)	Only Pc loss	Model should predict background correctly

Landmark Detection:



Landmark detection means identifying **specific key points** on an object (not just a bounding box).

Examples:

- Face: eyes, nose, mouth corners, chin
- Hand: finger joints
- Body: hips, knees, shoulders (pose estimation)
- Cars: wheel centers, headlights

Output

[[left_eye_x, left_eye_y],

[right_eye_x, right_eye_y],

[nose_x, nose_y],

[mouth_left_x, mouth_left_y],

[mouth_right_x, mouth_right_y]]

Convolutional Implementation Using Sliding Window for Object Detection

Classical object detection used a sliding window approach, where a fixed-size window moves across the image and each cropped region is sent to a classifier. This method is slow because the classifier repeatedly processes many overlapping regions.

Convolutional Neural Networks (CNNs) solve this inefficiency by converting the sliding-window operation into a single convolutional pass. In CNNs, the same set of filters slides over the image, meaning each convolution operation naturally evaluates local image regions—exactly like sliding windows. Fully connected layers of a classifier can be replaced by 1×1 convolution layers, allowing the network to take the full image as input and output a heatmap where each cell represents the classifier's decision for each window location.

Thus, instead of cropping thousands of windows, the CNN performs all evaluations at once. This makes convolutional sliding-window detection fast, efficient, and the foundation for modern detectors such as Faster R-CNN, SSD, and YOLO.

Intersection over Union (IoU)

Intersection over Union (IoU) is a metric used in object detection to measure how well a predicted bounding box matches the ground-truth box. It is defined as the ratio between the area of overlap and the area of union of the two boxes:

$$IoU = \frac{Intersection(Predicted\_Box, Ground\_Truth\_Box)}{Area(Predicted\_Box) + Area(Ground\_Truth\_Box) - Intersection}$$

The value ranges from 0 to 1, where:

- 1 means a perfect match between predicted and ground-truth boxes
- 0 means no overlap

Non-max Suppression:

NMS Algorithm (Step-by-Step):

1. Start with all predicted boxes and their confidence scores.
2. Select the box with the highest confidence score.
3. Remove all other boxes that have IoU greater than a chosen threshold (e.g., 0.5) with this box.
4. Repeat until no boxes remain.

Anchor Boxes

Anchor boxes are predefined bounding boxes of different shapes and sizes placed at each location of a feature map in object detection models like YOLO, SSD, and Faster R-CNN.

They help the model detect objects of varying sizes and aspect ratios. For each anchor box, the model predicts:

- Adjustments to fit the object (position and size)
- Class probabilities

By using multiple anchor boxes per location, the detector can handle multiple objects in the same region and improve detection accuracy.

Key point: Anchor boxes act as templates that are adjusted to match the actual objects in the image.

Keep in mind that a grid size of 13 * 13 means dividing the image into 13 rows by 13 columns, and each grid cell is image_shape / grid_size = pixels. And modern object detection algorithms like YOLO don't use sliding window concept, a classical method.

YOLO Algo

1. Input image divided into a grid (S × S)
2. Each grid cell predicts:
    - Bounding box → center (b_x, b_y), width b_w, height b_h
    - Objectness score P_c → how likely an object is present
    - Class probabilities for C classes
3. Confidence Score:

Confidence = P_c × IoU(pred, truth)

4. During inference:
    - Filter low-confidence boxes
    - Apply Non-Max Suppression (NMS) to remove duplicates

🔹 Output Format per Box

(P_c, b_x, b_y, b_w, b_h, p_1, p_2, ..., p_C)

Where:

- P_c = Objectness/confidence score
- b_x, b_y = Bounding box center coordinates
- b_w, b_h = Bounding box width and height
- p_1, p_2, ..., p_C = Class probabilities for C classes

🔹 Loss Function (Multi-Task)

The YOLO loss function consists of three main components:

✔ Localization loss (box center & size)
✔ Confidence loss
✔ Classification loss

Combined as a weighted sum to balance object vs. background detection:

$$\text{Loss} = \lambda_{coord} * \Sigma[(b_x - \hat{b}_x)^2 + (b_y - \hat{b}_y)^2 + (b_w - \hat{b}_w)^2 + (b_h - \hat{b}_h)^2]$$

$$+ (P_c - \hat{P}_c)^2$$

$$+ \Sigma_{classes}(c_i - \hat{c}_i)^2$$

🔹 Anchors & Multi-Scale Detection (Modern YOLO)

Anchor boxes help detect objects of various shapes and sizes.

YOLOv3/v4/v5/v7/v8 use 3 detection scales:

Grid Size	Object Size	Use Case
13×13	Large objects	Detecting big objects in the scene
26×26	Medium objects	Detecting medium-sized objects
52×52	Small objects	Detecting fine-grained, small objects