Non-Maximum Suppression (NMS) in Object Detection:

Object detection models like YOLO, SSD, and Faster R-CNN often predict multiple bounding boxes for a single object. This redundancy needs to be filtered out to produce clean, precise outputs. This is where **Non-Maximum Suppression (NMS)** comes in — a simple yet crucial technique to refine detections.

Non-Maximum Suppression

Non-Maximum Suppression is a key step in object detection models to remove overlapping bounding boxes and keep only the most confident ones. Non-Maximum Suppression is a post-processing algorithm used to select the best bounding box among overlapping candidates. It works by:

1. **Sorting** all boxes by their confidence scores.
2. **Selecting** the box with the highest score.
3. **Removing** boxes that have a high overlap (IoU) with the selected box.
4. **Repeating** the process until no boxes remain.

This ensures only the most relevant detections are retained.

Need for NMS

When a model detects multiple bounding boxes for the same object, NMS:

- Removes redundant boxes
- Keeps the one with the highest confidence score

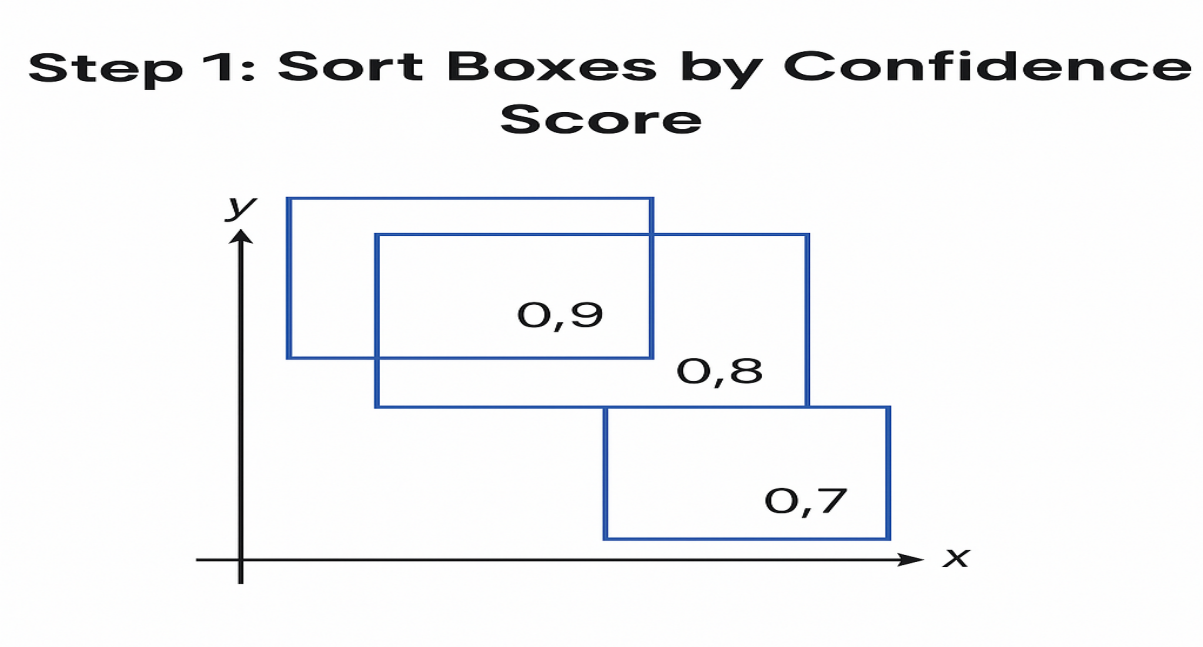
Non-Maximum Suppression (NMS) Procedure

Given:

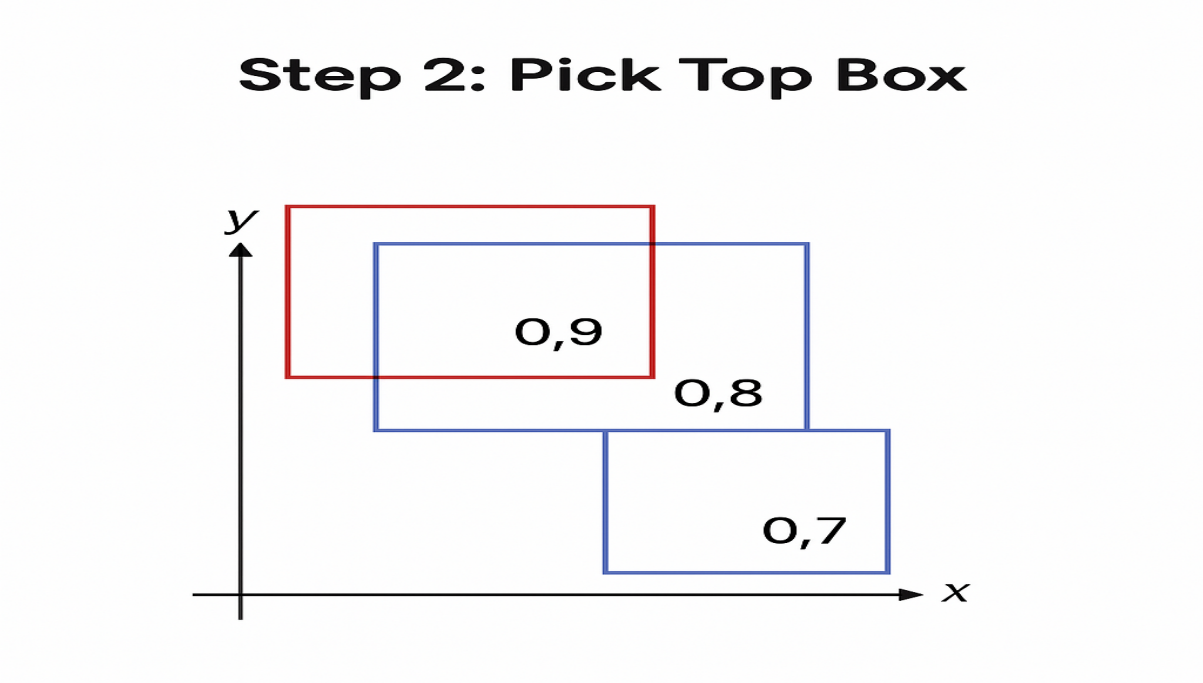
- A list of **bounding boxes**: [[x1, y1, x2, y2], ...]
- A list of **confidence scores**: [score1, score2, ...]
- An **IoU threshold** (e.g., 0.5)

Step-by-Step Algorithm:

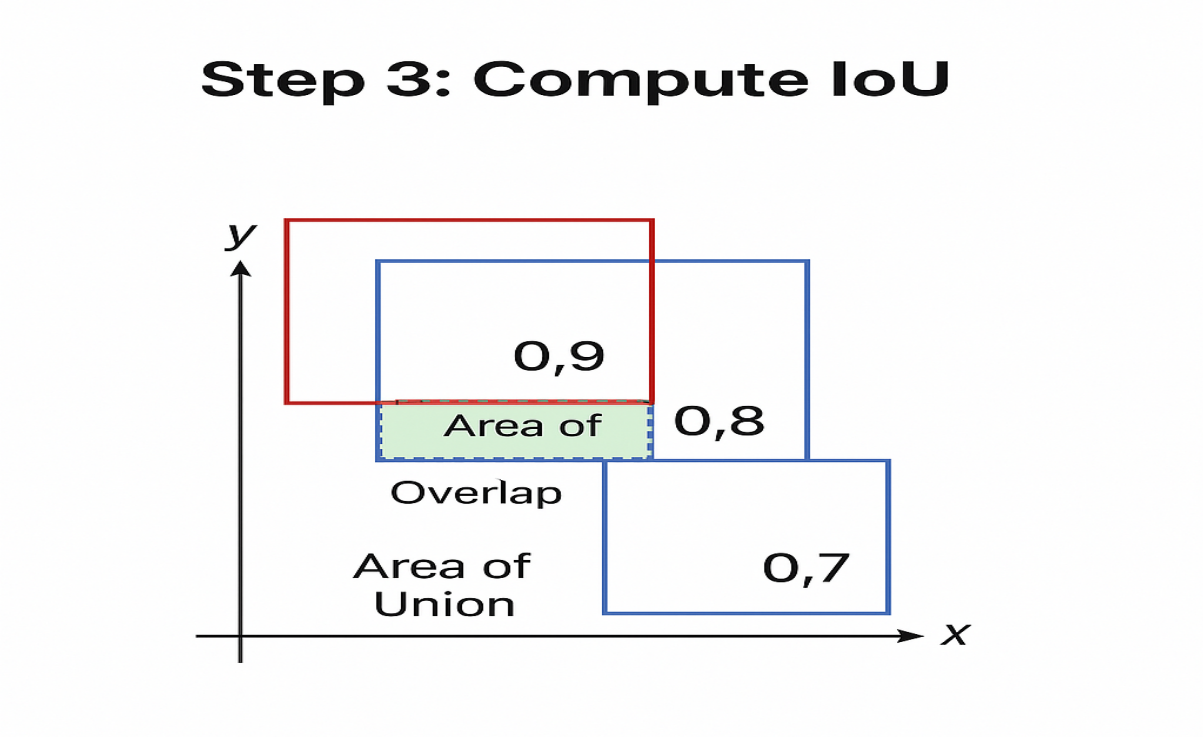
1. **Sort Boxes by Confidence Score (Descending)**
    - Start by sorting all the boxes based on their associated confidence scores in descending order.



2. **Initialize an Empty List for Keeping Selected Boxes**
    - selected_boxes = []



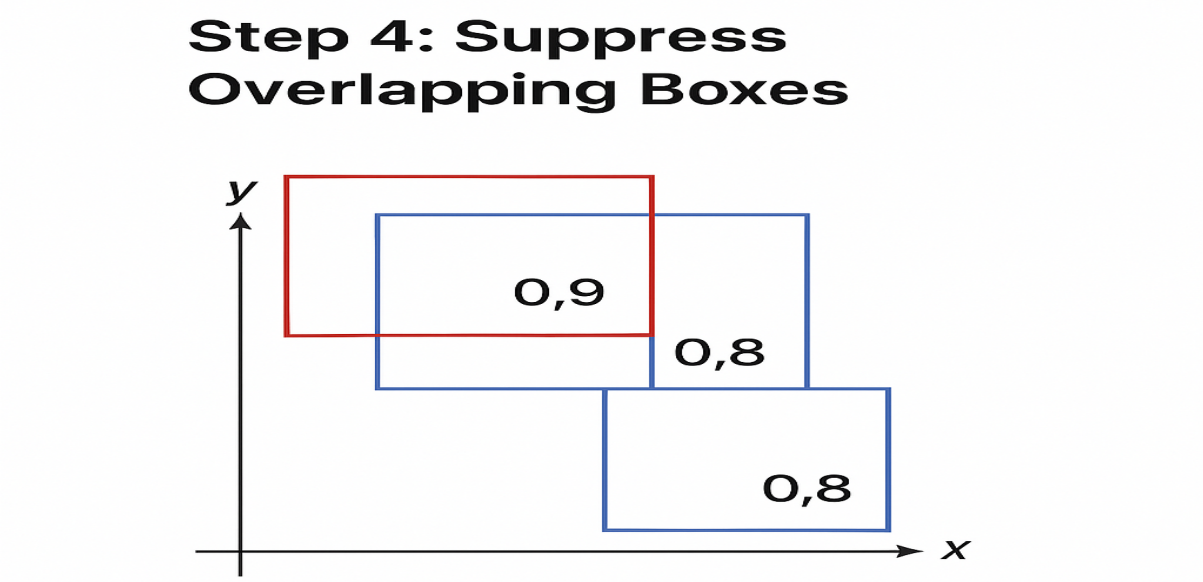
3. **Iterate Through the Boxes:**
    - Pick the box with the **highest confidence score** and **add it** to selected_boxes.



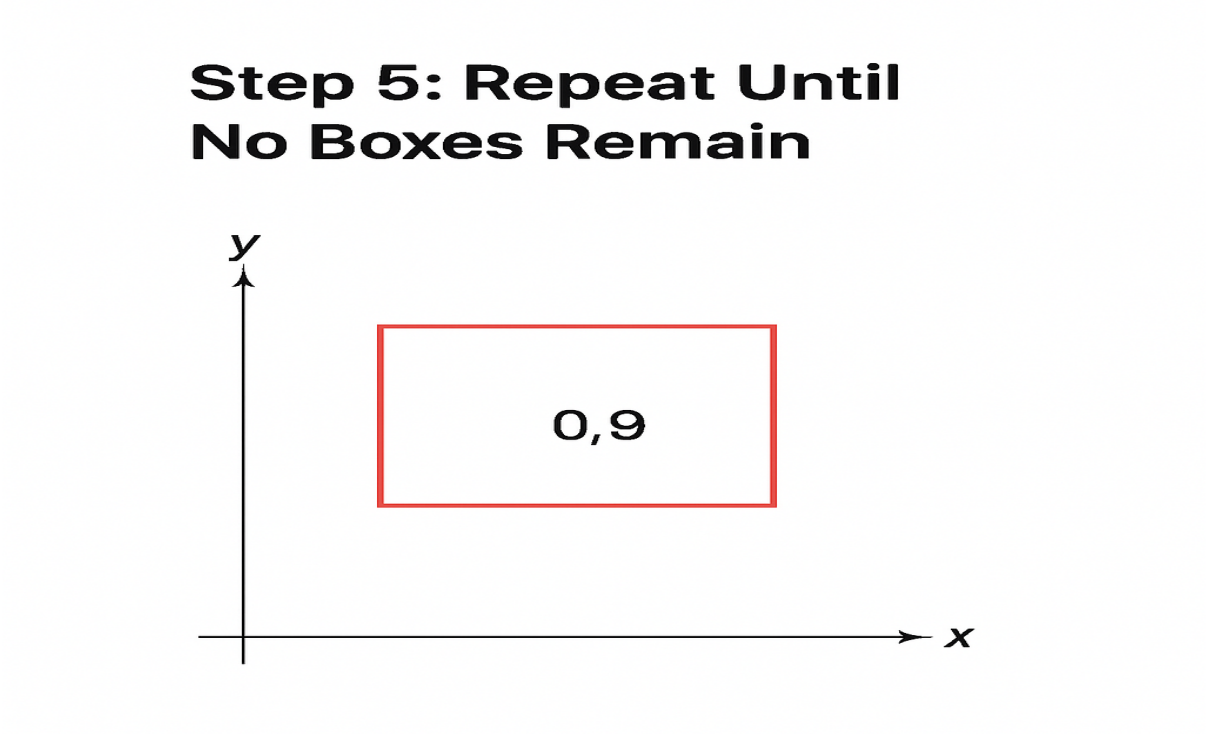
4. **Compute IoU** (Intersection over Union):
    - For the current highest-scoring box, **calculate IoU** with every other box.

IoU formula:

IoU=Area of Overlap/Area of Union



5. **Suppress Overlapping Boxes:**
    - Discard all boxes with **IoU greater than the threshold** with the current box (they overlap too much and are considered redundant).



6. **Repeat:**
    - Repeat the above process with the remaining boxes until none are left.

Standard NMS vs Soft-NMS

Feature	Standard NMS	Soft-NMS
Suppression	Hard removal of overlapping boxes	Reduces score based on overlap
Accuracy	May miss close detections	Better at keeping near-overlapping boxes
Use Case	Faster, simple use cases	More accurate in crowded scenes

NMS Algorithm

```python
import numpy as np

def iou(box1, box2):
    x1 = max(box1[0], box2[0])
    y1 = max(box1[1], box2[1])
    x2 = min(box1[2], box2[2])
    y2 = min(box1[3], box2[3])

    inter_area = max(0, x2 - x1) * max(0, y2 - y1)

    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])

    union_area = box1_area + box2_area - inter_area

    return inter_area / union_area if union_area != 0 else 0

def non_max_suppression(boxes, scores, iou_threshold):
    indices = np.argsort(scores)[::-1]  # Sort scores in descending order
    keep = []

    while len(indices) > 0:
        current = indices[0]
        keep.append(current)
        rest = indices[1:]

        filtered = []
        for i in rest:
            if iou(boxes[current], boxes[i]) < iou_threshold:
                filtered.append(i)

        indices = np.array(filtered)

    return keep
```

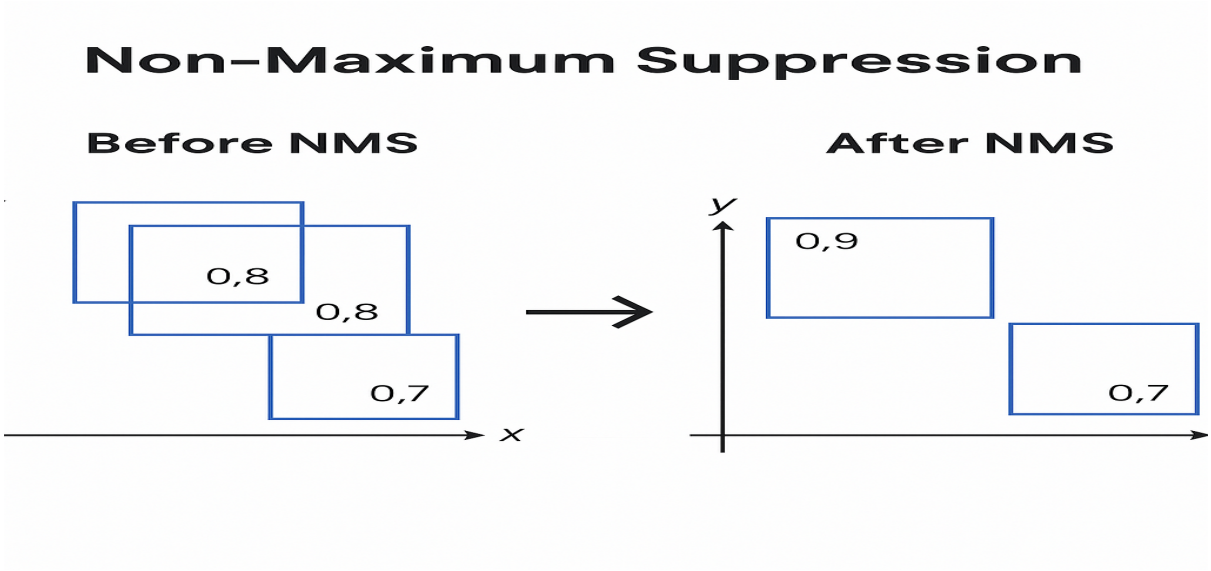
```python
[7] boxes = [
        [100, 100, 210, 210],
        [105, 105, 215, 215],
        [250, 250, 400, 400]
    ]
    scores = [0.9, 0.8, 0.7]

    iou_threshold = 0.5
    selected_indices = non_max_suppression(boxes, scores, iou_threshold)
    selected_boxes = [boxes[i] for i in selected_indices]

    print("Selected Boxes:", selected_boxes)
```

```
Selected Boxes: [[100, 100, 210, 210], [250, 250, 400, 400]]
```

Box 2 is suppressed because it has high overlap with box 1, and a lower confidence score.

Visualizing NMS

- Before NMS: multiple overlapping boxes
- After NMS: only the highest scoring, least overlapping boxes remain



Using NMS in Deep Learning Frameworks

```
[8]  #Pytorch
     boxes = [
         [100, 100, 210, 210],
         [105, 105, 215, 215],
         [250, 250, 400, 400]
     ]
     scores = [0.9, 0.8, 0.7]

     iou_threshold = 0.5
     selected_indices = non_max_suppression(boxes, scores, iou_threshold)
     selected_boxes = [boxes[i] for i in selected_indices]

     print("Selected Boxes:", selected_boxes)
```

```
Selected Boxes: [[100, 100, 210, 210], [250, 250, 400, 400]]
```

```
#Tensorflow
import tensorflow as tf

boxes = tf.constant([[100, 100, 210, 210], [105, 105, 215, 215], [250, 250, 400, 400]], dtype=tf.float32)
scores = tf.constant([0.9, 0.8, 0.7])

selected_indices = tf.image.non_max_suppression(
    boxes=boxes, scores=scores, max_output_size=3, iou_threshold=0.5)
print("Selected Boxes:", tf.gather(boxes, selected_indices))
```

```
Selected Boxes: tf.Tensor(
[[100. 100. 210. 210.]
 [250. 250. 400. 400.]], shape=(2, 4), dtype=float32)
```

Conclusion

Non-Maximum Suppression is a cornerstone of object detection post-processing. Understanding its internals can help in:

- Fine-tune object detectors
- Optimize precision vs recall trade-offs
- Debug overlapping detection issues

Whether using a custom implementation or a framework's built-in version, mastering Non-Maximum Suppression (NMS) is essential for building efficient object detection pipelines.

Reference

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 580–587.

[2] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Soft-NMS – Improving Object Detection With One Line of Code," *arXiv preprint arXiv:1704.04503*, 2017. [Online]. Available: https://arxiv.org/abs/1704.04503

[3] TorchVision, "torchvision.ops.nms," *PyTorch Documentation*. [Online]. Available: https://pytorch.org/vision/stable/ops.html#torchvision.ops.nms

[4] TensorFlow, "tf.image.non_max_suppression," *TensorFlow API Documentation*. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/image/non_max_suppression

[5] S. Malik, "Non-Maximum Suppression: Theory and Implementation in Python," *LearnOpenCV*, 2021. [Online]. Available: https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pythonnms/

[6] J. Hall, "What is Non-Maximum Suppression (NMS) in Object Detection?" *Roboflow Blog*, 2022. [Online]. Available: https://blog.roboflow.com/non-maximum-suppression/