ROB 498/599: Deep Learning for Robot Perception (DeepRob)

Lecture 13: NMS IoU; PoseCNN

02/24/2025

https://deeprob.org/w25/

ROBOTICS

Today

- Feedback and Recap (5min)
- Canvas Quiz question (15min)
  - IoU threshold and NMS
- PoseCNN (50min)
- Summary and Takeaways (5min)

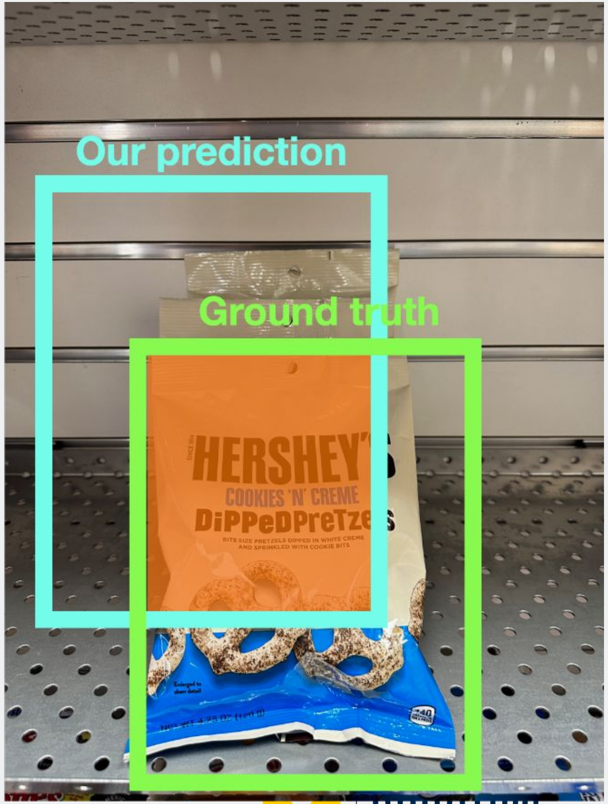
Final Project Group Sign-Up (2-4 people per group):
https://docs.google.com/spreadsheets/d/1FjWAjJ8p26xZmZaqsW4lew8H4iKe0FA78Q30eZ38g7A/edit?usp=sharing

Recap: IoU (Intersection over Union)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU) (Also called "Jaccard similarity" or "Jaccard index"):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$



Recap: Non-Max Suppression (NMS)

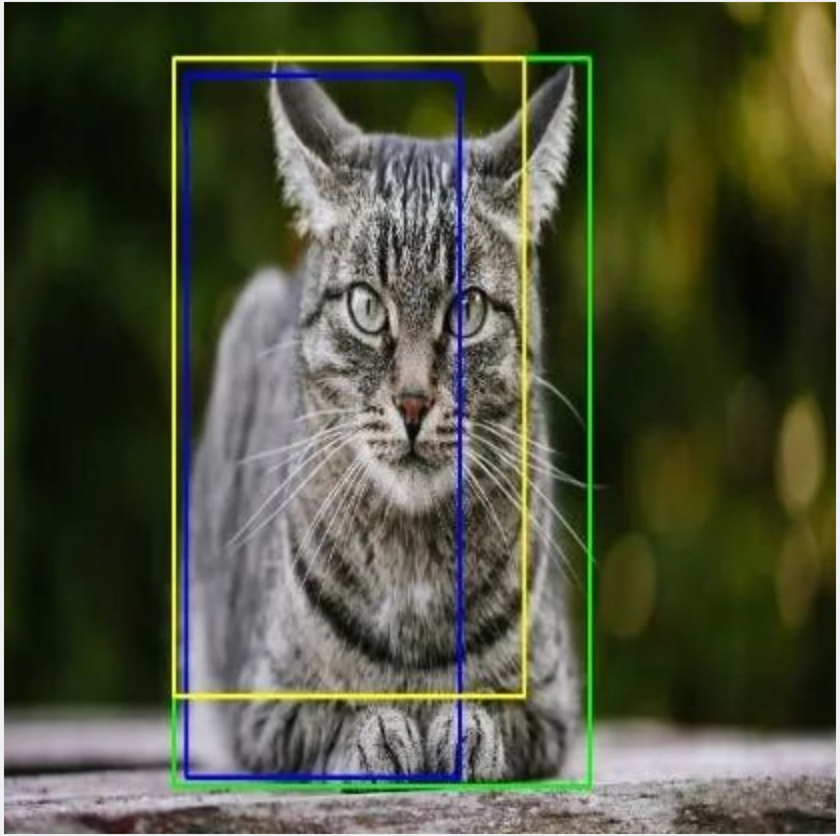
Problem: Object detectors often output many overlapping detections

Solution: Post-process raw detections using Non-Max Suppression (NMS)

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU> threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

IoU(,) = 0.8
IoU(,) = 0.03
IoU(,) = 0.05

Example:



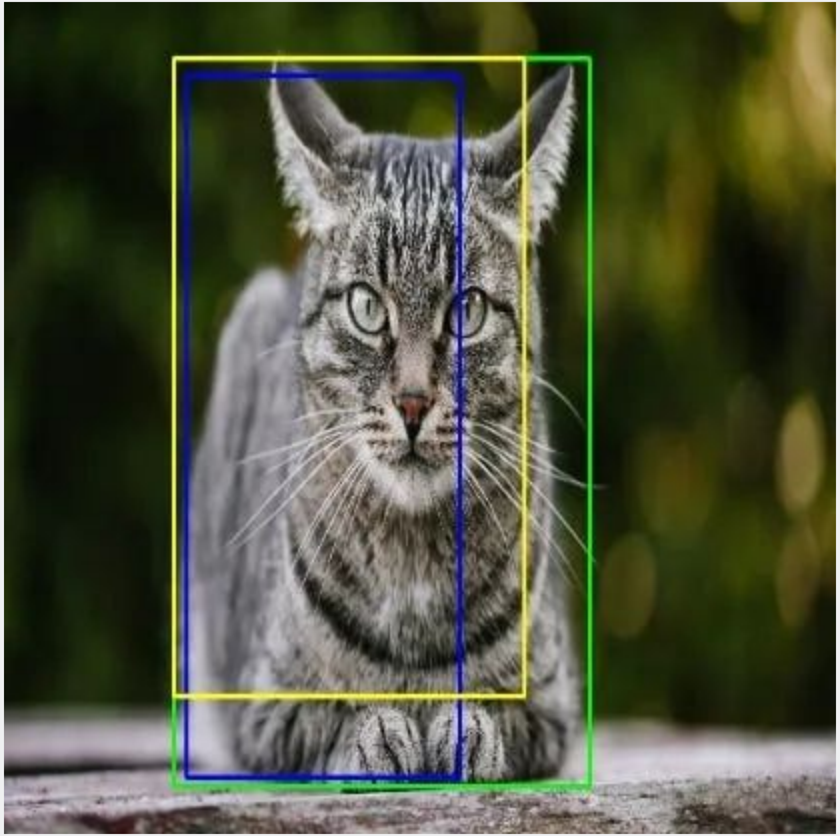
green_box = [x1, y1, x2, y2, ”Cat”, 0.9]

yellow_box = [x5, y5, x6, y6, ”Cat”, 0.75]

blue_box = [x3, y3, x4, y4, ”Cat”, 0.85]

Example:

green_box = [x1, y1, x2, y2, "Cat", 0.9]
yellow_box = [x5, y5, x6, y6, "Cat", 0.75]
blue_box = [x3, y3, x4, y4, "Cat", 0.85]



NMS Algorithm

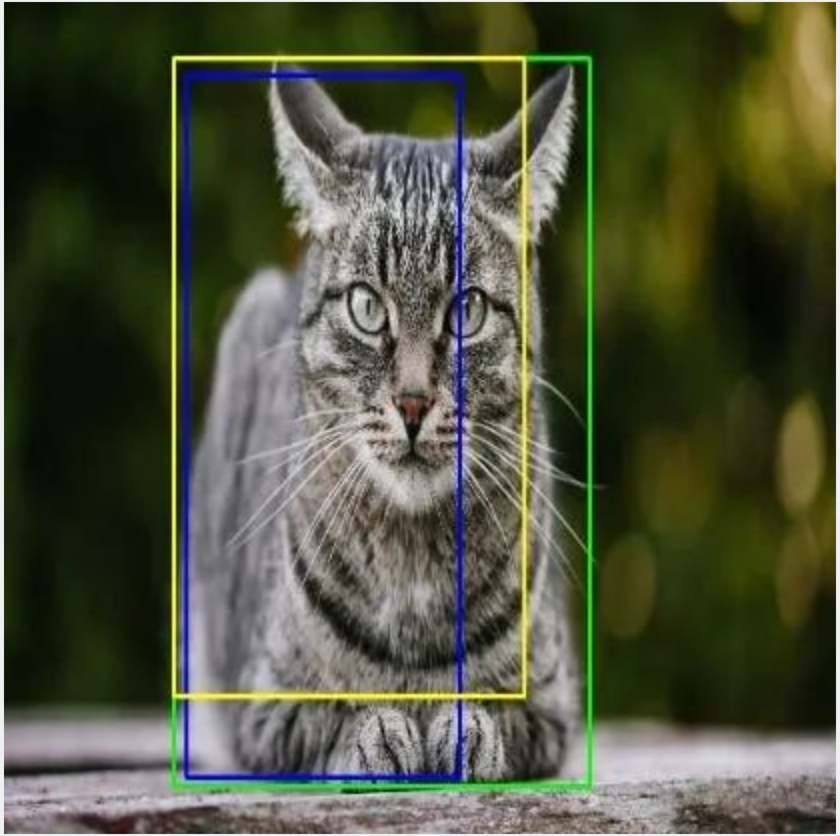
Stage 1 - initial removal of boxes

1. Sort the confidence

   bbox_list = [green_box, blue_box, yellow_box]

Example:

green_box = [x1, y1, x2, y2, "Cat", 0.9]
yellow_box = [x5, y5, x6, y6, "Cat", 0.75]
blue_box = [x3, y3, x4, y4, "Cat", 0.85]



NMS Algorithm:

Stage 1 - initial removal of boxes

1. Sort the confidence

   bbox_list = [green_box, blue_box, yellow_box]

2. Set a confidence threshold

   Let's say, confidence threshold= 0.8

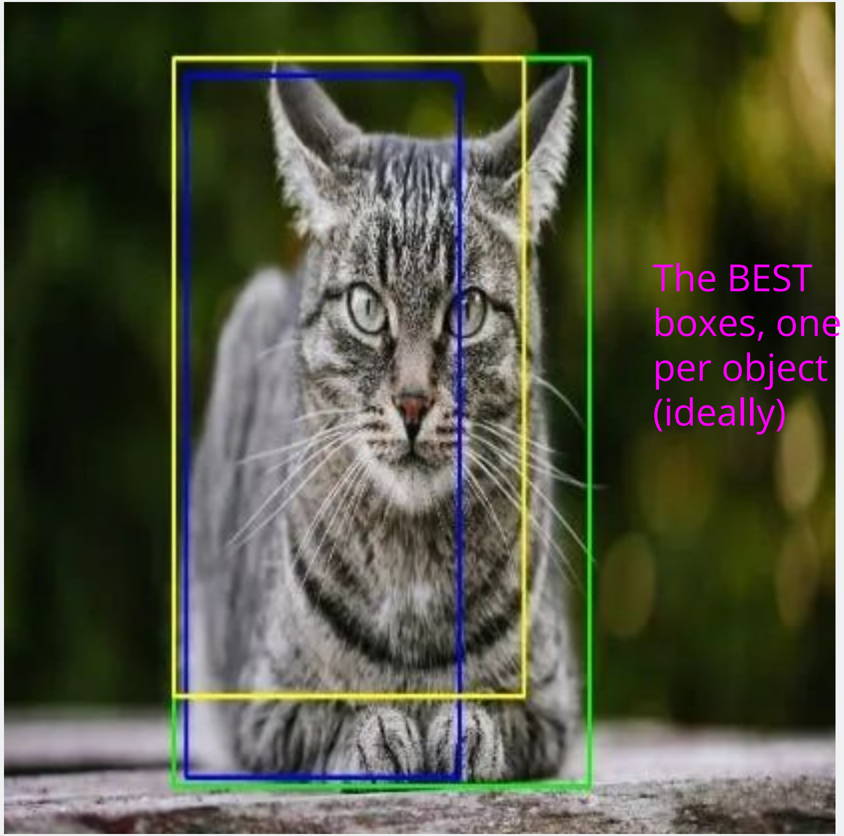
   Any box that has a confidence below this threshold will be **removed**.

bbox_list = [green_box, blue_box]

ROBOTICS

Example:

green_box = [x1, y1, x2, y2, "Cat", 0.9]
yellow_box = [x5, y5, x6, y6, "Cat", 0.75]
blue_box = [x3, y3, x4, y4, "Cat", 0.85]



NMS Algorithm:

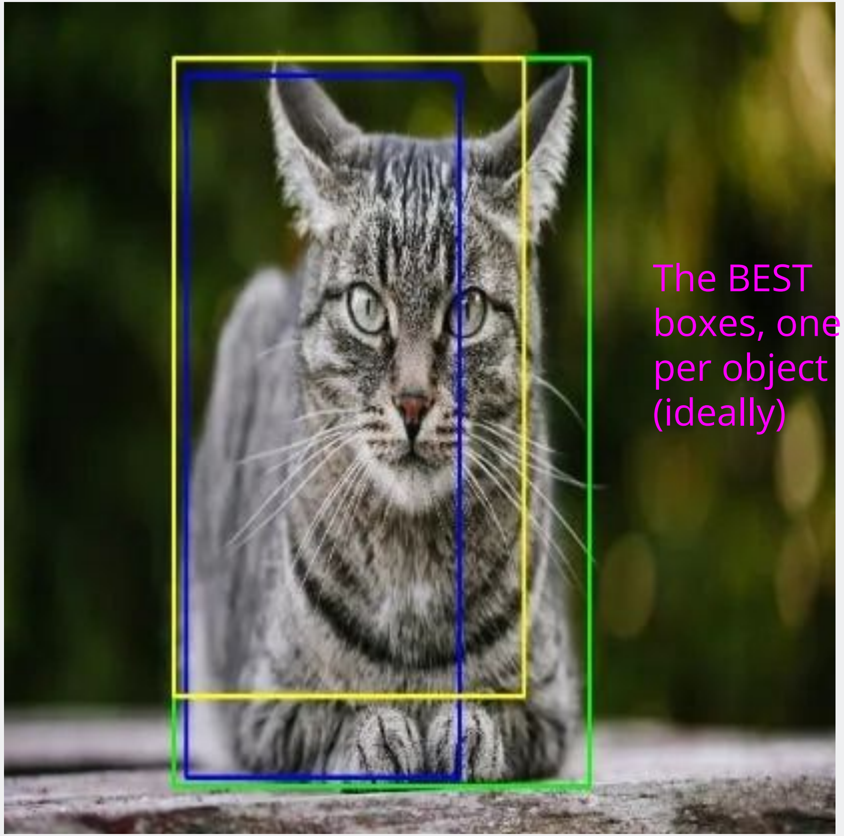
Stage 2 - IoU comparison of Boxes

1. Start a new list. Start with the highest confidence box.

bbox_list_new = [green_box]

Example:

green_box = [x1, y1, x2, y2, "Cat", 0.9]
yellow_box = [x5, y5, x6, y6, "Cat", 0.75]
blue_box = [x3, y3, x4, y4, "Cat", 0.85]



NMS Algorithm:

Stage 2 - IoU comparison of Boxes

1. Start a new list. Start with the highest confidence box.

   bbox_list_new = [green_box]

2. Set an IoU threshold (e.g., 0.5)

*IoU threshold: determine when two boxes **overlap** too much and likely represent the same object.

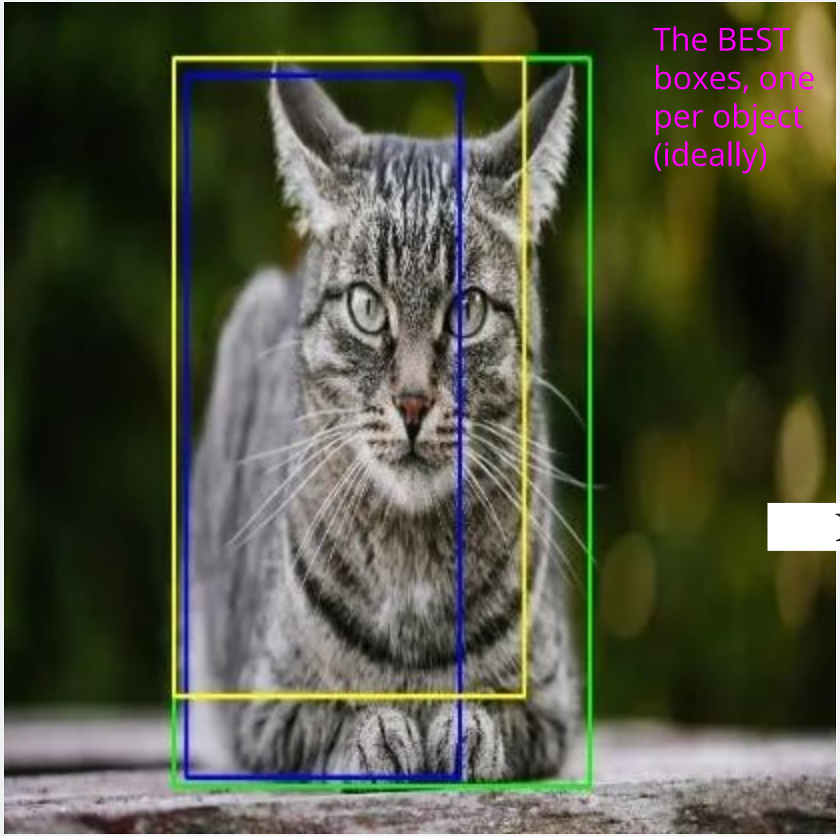
Example:

green_box = [x1, y1, x2, y2, "Cat", 0.9]
yellow_box = [x5, y5, x6, y6, "Cat", 0.75]
blue_box = [x3, y3, x4, y4, "Cat", 0.85]

NMS Algorithm: **Stage 2** - IoU comparison of Boxes

1. Start a new list. Start with the highest confidence box.

bbox_list_new = [green_box]



2. Set an IoU threshold (e.g., **0.5**)

*IoU threshold: determine when two boxes **overlap** too much and likely represent the same object.

3. Compare IoU with remaining boxes

Calculate the **IoU** of the green box with all remaining boxes of the same class.

Note: bbox_list = [green_box, blue_box] (from stage 1)

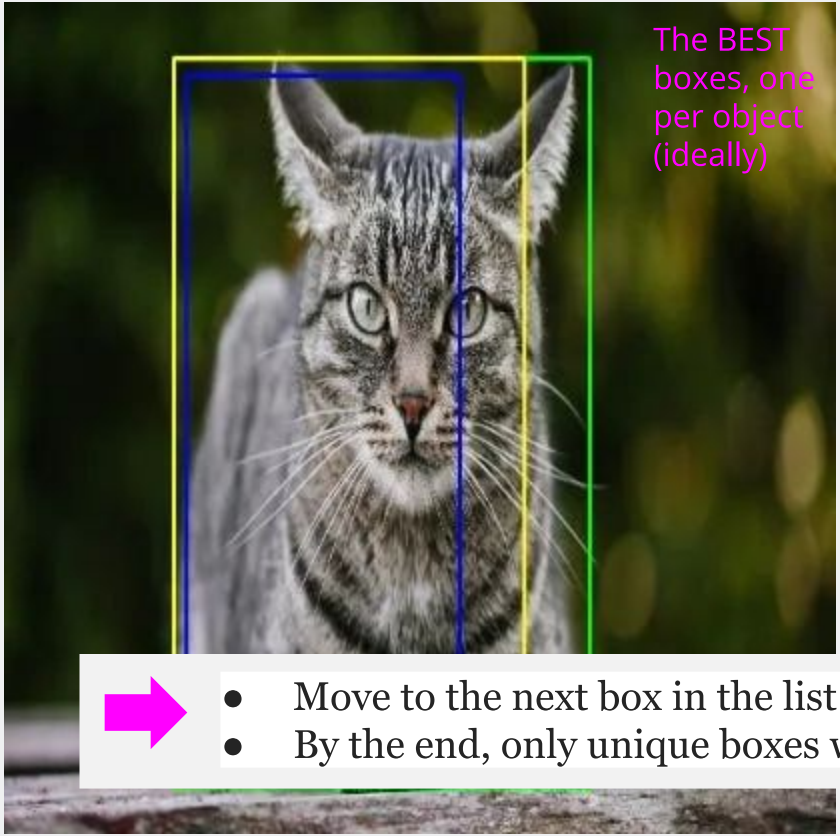
If IoU (green_box, blue_box)>0.5, means they have significant overlap (likely detect the same object)

M | ROBOTICS

Example:

green_box = [x1, y1, x2, y2, "Cat", 0.9]
yellow_box = [x5, y5, x6, y6, "Cat", 0.75]
blue_box = [x3, y3, x4, y4, "Cat", 0.85]

NMS Algorithm: **Stage 2** - IoU comparison of Boxes



1. Start a new list. Start with the highest confidence box.

bbox_list_new = [green_box]

2. Set an IoU threshold (e.g., **0.5**)

*IoU threshold: determine when two boxes **overlap** too much and likely represent the same object.

3. Compare IoU with remaining boxes

IoU (green_box, blue_box)>0.5

4. Remove the lower confidence box

   Remove the blue_box

5. Repeat for all boxes

- Move to the next box in the list and repeat the process until all boxes have been checked.
- By the end, only unique boxes with high confidence will remain in `bbox_list_new`.

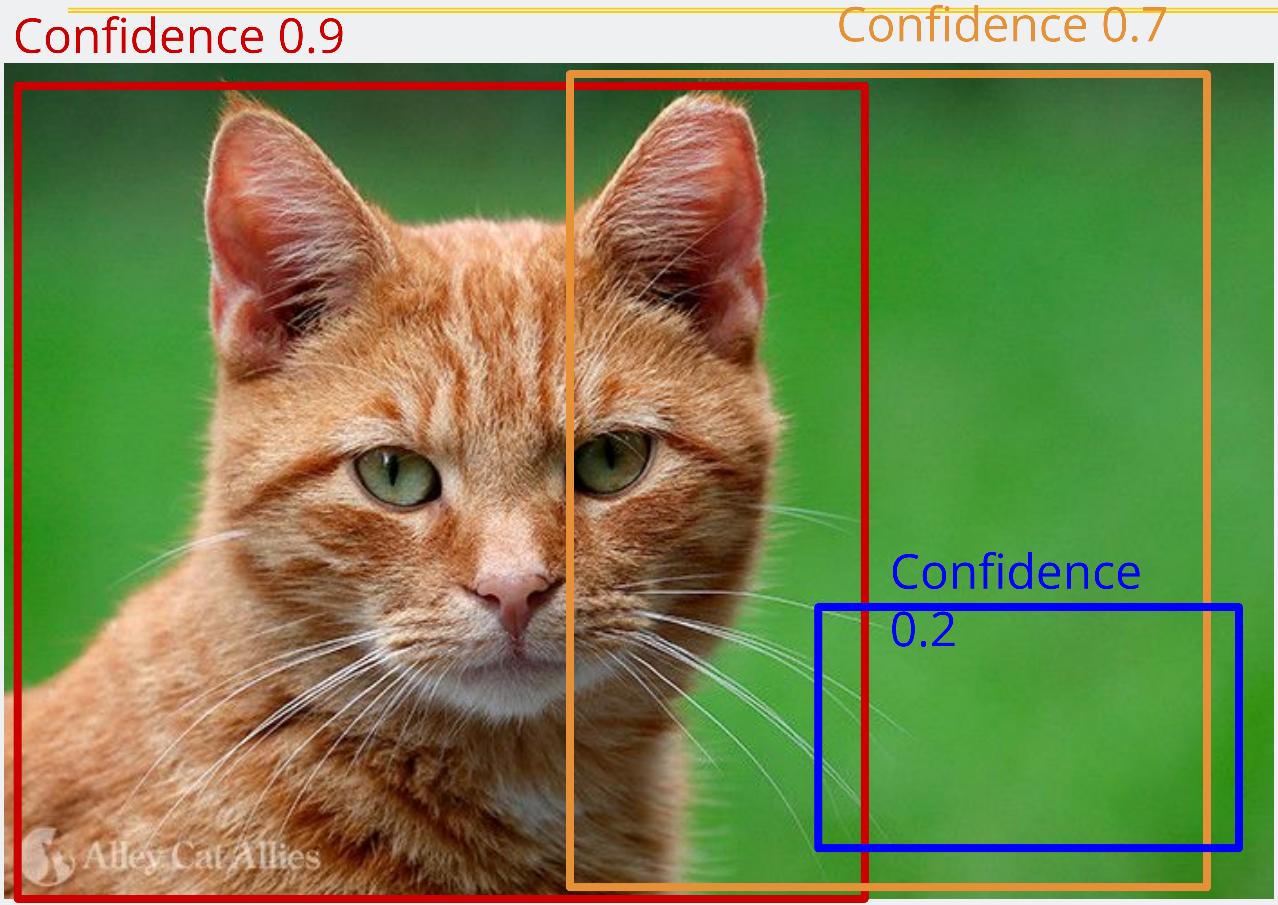
M | ROBOTICS

From the original R-CNN paper

https://arxiv.org/pdf/1311.2524

"Given all scored regions in an image, we apply a greedy non-maximum suppression (for each class independently) that <u>rejects</u> a region if it has an intersection-over-union (IoU) overlap with a <u>higher scoring</u> selected region <u>larger</u> than a learned threshold."

Example:



(compare to red)

If IoU threshold = 0.1 (low)
reject orange and blue boxes

Final detection = red box

Precision = TP/(TP+FP) = 1/(1+0) = 1
Recall = TP/(TP+FN) = 1/(1+0) = 1

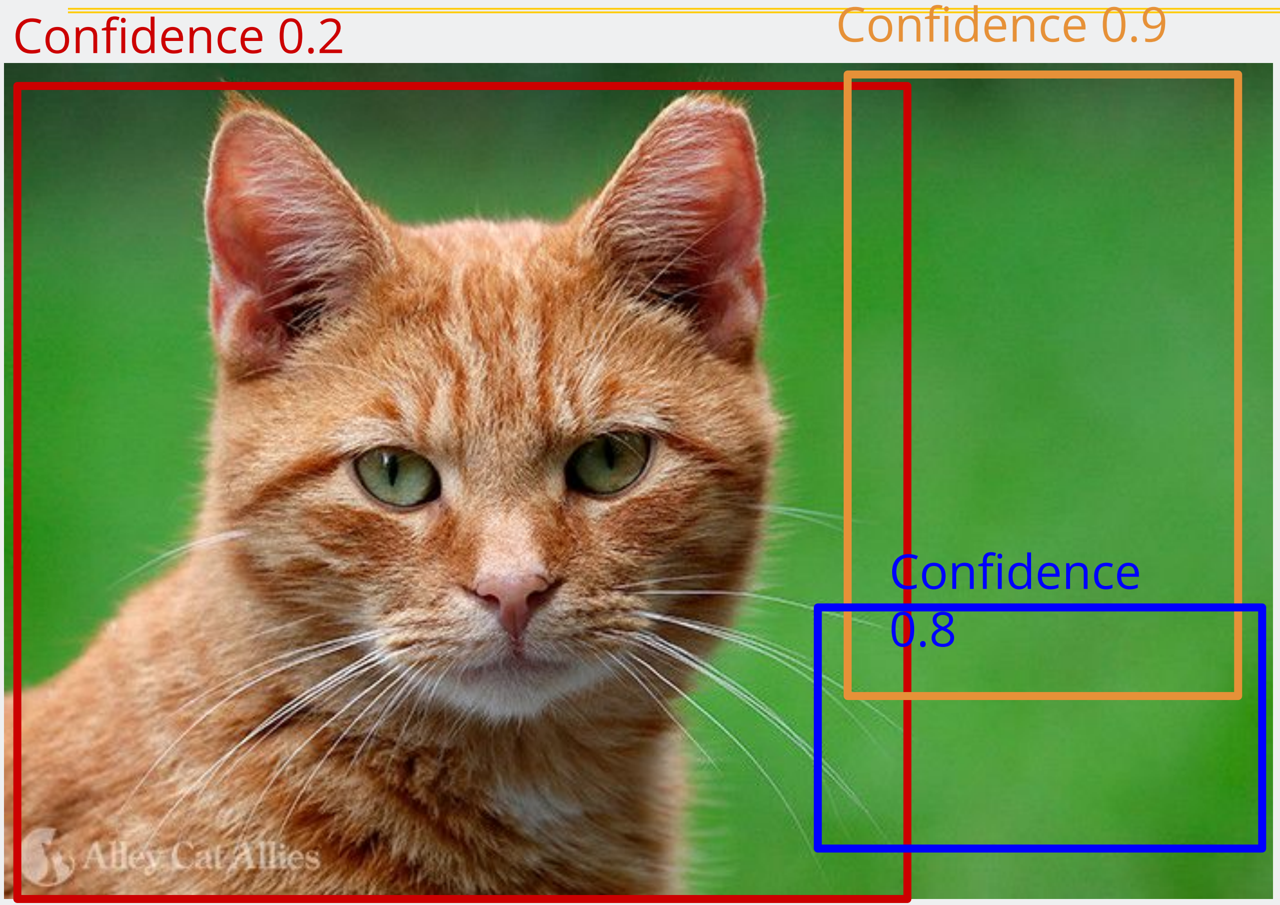
If IoU threshold = 0.9 (high)
NOT reject orange and blue boxes

Final detection = red, orange, blue box

Precision = TP/(TP+FP) = 1/(1+2) = 1/3
Recall = TP/(TP+FN) = 1/(1+0) = 1

M ROBOTICS

Example:



(compare to orange)

If IoU threshold = 0.1 (low)
reject red and blue boxes

Final detection = orange box

Precision = TP/(TP+FP) = 0/(0+1) = 0
Recall = TP/(TP+FN) = 0/(0+0) = 0

If IoU threshold = 0.9 (high)
NOT reject red and blue boxes

Final detection = orange, blue, red box

Precision = TP/(TP+FP) = 1/(1+2) = 1/3
Recall = TP/(TP+FN) = 1/(1+0) = 1

ROBOTICS

Additional Reading

- https://medium.com/@abhishekjainindore24/non-maximal-suppression-in-object-detection-nms-028ce2be6cdc

- https://github.com/ultralytics/ultralytics/issues/9150
- https://github.com/ultralytics/ultralytics/issues/8428

"Lower IoU threshold means stricter overlap criteria, potentially leading to more aggressive suppression of close detections."

- https://docs.ultralytics.com/reference/utils/ops/#ultralytics.utils.ops.non_max_suppression "iou_thres - The IoU threshold below which boxes will be filtered out during NMS. Valid values are between 0.0 and 1.0."

- https://arxiv.org/pdf/1705.02950

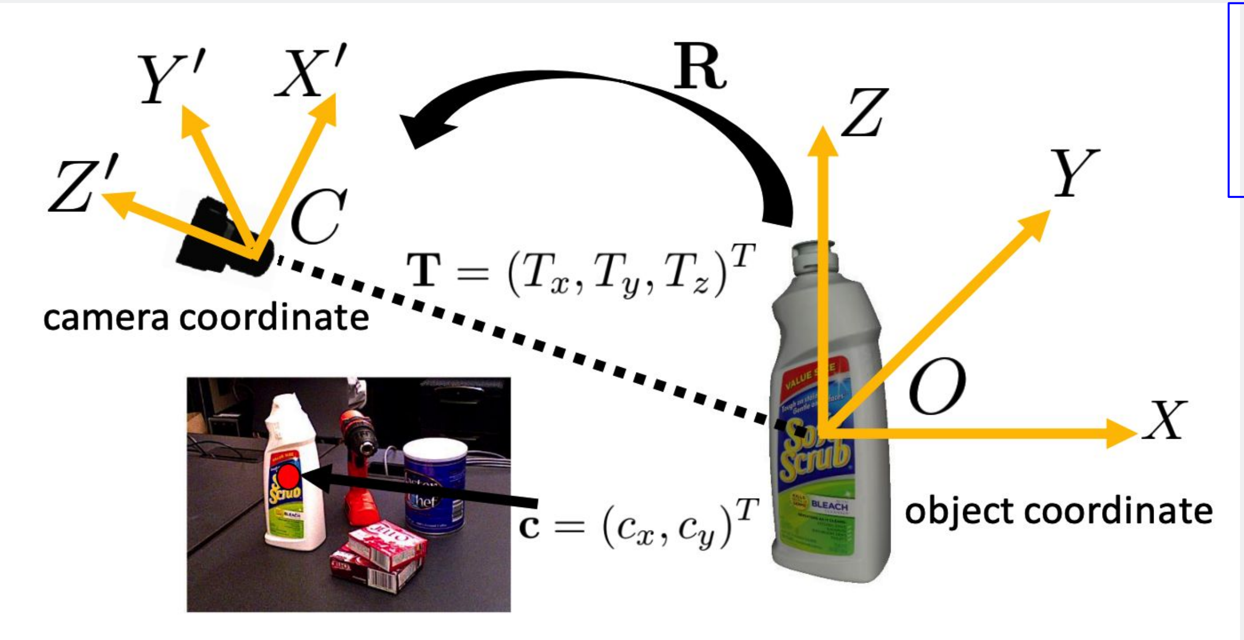
```
torchvision.ops.nms(boxes: Tensor, scores: Tensor, iou_threshold: float) → Tensor [SOURCE]
```

https://pytorch.org/vision/main/generated/torchvision.ops.nms.html

PoseCNN: **6D** Pose Estimation

- **P3 released, Due March 9, 2025 Start NOW!!!**
- **PoseCNN will be part of P4.**

https://arxiv.org/pdf/1711.00199



3D Translation T
3D Rotation R

2D → 3D

Task: determining the six degree-of-freedom (6D) pose of an object in 3D space based on RGB images

PoseCNN: **6D** Pose Estimation

https://arxiv.org/pdf/1711.00199

YCB-Video dataset

https://www.ycbbenchmarks.com/

Fig. 5. The subset of 21 YCB Objects selected to appear in our dataset.

PoseCNN: **6D** Pose Estimation

https://arxiv.org/pdf/1711.00199

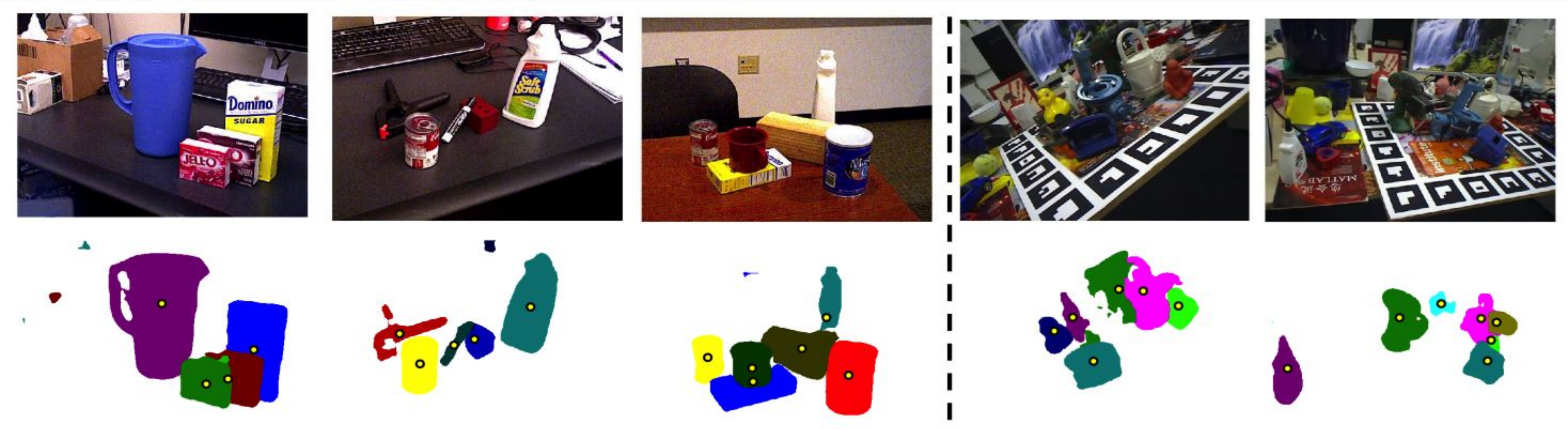
LINEMOD
dataset

https://bop.felk.cvut.cz/datasets/

PoseCNN: **6D** Pose Estimation

https://arxiv.org/pdf/1711.00199

Dataset Examples (given in paper)



Top: RGB image.
Bottom: labels and centers.

PoseCNN: 6D Pose Estimation

PROPS-POSE dataset (will be provided again with P4)

https://deeprob.org/w24/datasets/props-pose/

Download here:
https://drive.google.com/file/d/15rhwXhzHGKtBcxJAYMWJG7gN7BLLhyAq/view?usp=sharing

PoseCNN: **6D** Pose Estimation

Useful Functions

(refer back to this in P4!)

torch.nn.init.kaiming_normal_(*tensor*, *a=0*, *mode='fan_in'*, *nonlinearity='leaky_relu'*, *generator=None*) [SOURCE]

Initialize weight to kaiming_normal

Fill the input *Tensor* with values using a Kaiming normal distribution.

torch.nn.init.zeros_(*tensor*) [SOURCE]

Initialize bias to zero

upsample/interpolate

torch.nn.functional.interpolate(*input*, *size=None*, *scale_factor=None*, *mode='nearest'*, *align_corners=None*, *recompute_scale_factor=None*, *antialias=False*) [SOURCE]

PoseCNN: 6D Pose Estimation

*(refer back to this in P4!)

Useful Functions

`p3_helper/loss_Rotation` (to be used as rotation loss)

`p3_helper/IOUselection`

PoseCNN Forward pass **training**

```
pred_filtered_bbxs = IOUselection(bbox, gt_bbx, threshold=0.10)
```

If the size of `pred_filtered_bbxs` is larger than 0:

```
    quaternion = self.rotationBranch(.....)
    predRot,label_pred = self.estimateRotation(quaternion, pred_filtered_bbxs)
    gtRot = self.gtRotation(pred_filtered_bbxs, input_dict)
    loss_dict['loss_R'] = loss_Rotation(predRot, gtRot, label_pred,
    self.models_pcd)
```

`p3_helper/HoughVoting`

PoseCNN Forward pass **inference**

`bboxes` from SegmentationBranch

```
pred_centers, pred_depths = HoughVoting(segmentation, translation)
```

```
output_dict = self.generate_pose(predRot, pred_centers, pred_depths, bboxes)
```

ROBOTICS