

PRACTICE BOOK

Chapter 5

Object Detection and Robotic Decision Making

From Vision Model Output to a Useful Robot Decision

How to Use This Chapter

This chapter is a practical reference companion for Week 5. It follows the supplied workshop sequence and turns the code into small experiments that you can copy, run, inspect and modify. The aim is not to memorise Faster R-CNN or particular PyTorch calls. The aim is to understand how a perception model becomes one component of a robotics system.

The one idea to remember

A detector produces evidence. The engineering system must decide what to trust, what the evidence means, what action is appropriate, what to do when evidence is missing, and how to verify performance.

The Week 5 engineering question

Do not stop at: “Can the model detect the object?” Ask:

1. What exactly does the detector return?
2. Which predictions are relevant and reliable enough for this task?
3. How do I select the target when there are several candidates?
4. Where is the target in image space, and do I actually need 3D position?
5. What should the robot do with this information?
6. What happens when perception fails or becomes uncertain?
7. How do I know the whole perception system is good enough?

Chapter learning outcomes

- Explain the three core outputs of an object detector: boxes, labels and scores.
- Load and use a pretrained Faster R-CNN detector in the Week 5 software environment.
- Understand why preprocessing, device placement, eval mode and no_grad matter.
- Use confidence thresholds as explicit application-level filtering rules.
- Explain why confidence is evidence, not a guarantee of correctness.
- Compare perception models using task requirements, latency and hardware constraints.
- Separate perception, target selection, localisation and robot decision logic.
- Compute a bounding-box centre and distinguish image coordinates from physical 3D coordinates.
- Design explicit SEARCH/STOP/WAIT behaviour when no reliable target is available.
- Explain IoU, true/false positives, precision and recall.
- Use a reusable engineering skeleton for later vision-based robotics projects.

Workshop Map: What Are You Building?

Stage	Input	Output	Question to ask
Detect	Image	boxes + labels + scores	What evidence did the model produce?
Filter	Raw detections	candidate detections	Which predictions are relevant enough?
Select	Candidates	one target / none	Which object matters to the task?
Locate	Target box	image centre (cx, cy)	Where is it in the image?
Decide	Image location	action/state	What should the robot do?
Act	Action	robot command	How is the command executed safely?
Verify	Predictions + references	metrics/tests	Was the result reliable?

A useful software architecture is to keep these stages separate. A detector should not secretly decide the robot action. A target-selection function should not run the neural network again. An evaluation function should measure performance rather than changing the model during evaluation.

Connection to earlier weeks

Week 1 emphasised building a solution carefully. Week 2 introduced AI as an engineering assistant. Week 3 emphasised trusting perception only after validation and stress-testing. Week 4 developed camera/image geometry. Week 5 combines these ideas: a learned detector produces image-space evidence that must be filtered, interpreted, connected to decisions and evaluated.

Part 1 - Environment, Hardware and Input

Start with the environment and input, not the neural network. This is a small habit with large value: many apparent “AI problems” are actually input, file, package or device problems.

1 Check the environment

Copy-and-run: inspect the environment before debugging the model.

```
import sys
import torch
import torchvision

print("Python:", sys.version)
print("PyTorch:", torch.__version__)
print("Torchvision:", torchvision.__version__)
```

2 Select the device

```
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")

print("Using device:", device)
```

Key insight

The device is part of the computation contract. The model and its input tensors must be on compatible devices. Do not treat CPU/GPU selection as an unrelated setup detail.

3 Inspect the image

```
from PIL import Image
import matplotlib.pyplot as plt

image = Image.open(image_files[0]).convert("RGB")

print("Image size:", image.size)

plt.figure(figsize=(8, 5))
plt.imshow(image)
plt.axis("off")
plt.show()
```

- Check that the file is the image you think it is.
- Check image width and height.
- Look for the objects you expect the pretrained model to recognise.
- Remember that a pretrained detector has a fixed label set.

Common mistake

A pretrained detector is not a general-purpose object recogniser. If the required object class is not in the model's training label set, changing the confidence threshold will not solve the problem.

Part 2 - Load the Detector and Inspect Raw Predictions

This is the most important coding section. Do not immediately hide the detector output behind a final image. First inspect the raw structured result.

1 Load Faster R-CNN in this environment

```
from torchvision.models.detection import fasterrcnn_resnet50_fpn

def load_faster_rcnn(device):
    # torchvision 0.7.0 uses the older pretrained=True API.
    model = fasterrcnn_resnet50_fpn(pretrained=True)

    # Move model parameters to CPU or GPU.
    model = model.to(device)

    # Switch from training behaviour to inference/evaluation behaviour.
    model.eval()

    return model
```

Line	Meaning	Why it matters
pretrained=True	Loads an already trained detector.	We are using the model as a perception component, not training it from scratch.
model.to(device)	Moves model parameters to CPU/GPU.	Input tensors must be on the same device.
model.eval()	Uses evaluation/inference behaviour.	Essential when using a trained model for prediction.

2 The inference boundary

```
from torchvision import transforms

def detect(model, image, device):
    transform = transforms.ToTensor()

    # PIL image -> tensor, then move tensor to the model device.
    x = transform(image).to(device)

    # No gradients are needed during inference.
    with torch.no_grad():
        result = model([x])[0]

    # Move results to CPU for inspection and visualisation.
    return {key: value.cpu() for key, value in result.items()}
```

Why model([x])?

The detector API accepts a list of images. Even when you process one image, the input is [x], not just x. The [0] then selects the result dictionary for the first image.

3 Inspect boxes, labels and scores

```
result = detect(model, image, device)

print("Output keys:", list(result.keys()))
print("Number of raw predictions:", len(result["scores"]))

for i in range(min(5, len(result["scores"]))):
    print("Prediction", i)
    print("  box  :", result["boxes"][i].tolist())
```

```
print("  label:", int(result["labels"][i]))
print("  score:", float(result["scores"][i]))
```

Output	Typical meaning	Engineering interpretation
boxes	[x1, y1, x2, y2]	Where the model believes the object is in image coordinates.
labels	integer class ID	What object category the model predicts.
scores	floating-point score	How strongly the model supports that candidate.

Do not confuse score with certainty

A score such as 0.90 is not a guarantee that the prediction is correct, nor is it a guarantee that a robot action will succeed. It is evidence produced by the model. The application must decide how much evidence is sufficient.

4 Labels, Confidence and Visualisation

Convert class IDs into names

In torchvision 0.7.0 the convenient newer weights.meta['categories'] interface is unavailable. Use the supplied COCO category list in the workshop notebook.

```
# Example: map a numeric label to a readable class name.
name = COCO_CATEGORIES[int(label)]
print(name)
```

Do not memorise the entire category list. Understand the mapping: the model returns a numeric ID; the application needs a human-readable class name.

Confidence threshold experiment

```
THRESHOLDS = [0.30, 0.50, 0.80]
```

```
for threshold in THRESHOLDS:
    keep = result["scores"] >= threshold
    count = int(keep.sum())

    print("Threshold:", threshold)
    print("Accepted detections:", count)
```

Lower threshold	Higher threshold
Accepts more candidate detections.	Accepts fewer candidate detections.
May improve recall.	May reduce false positives.
May admit more false positives.	May miss weaker true detections.
More permissive.	More selective.

Important engineering rule

Do not say “0.80 is better than 0.50” simply because 0.80 is a larger number. The appropriate threshold depends on the task and the costs of false positives and false negatives.

Visualise accepted detections

```
from PIL import ImageDraw

def annotate(image, result, categories, threshold):
    output = image.copy()
    draw = ImageDraw.Draw(output)
```

```
for box, label, score in zip(
    result["boxes"], result["labels"], result["scores"]
):
    score_value = float(score)

    if score_value < threshold:
        continue

    x1, y1, x2, y2 = box.tolist()
    name = categories[int(label)]

    draw.rectangle(
        (x1, y1, x2, y2),
        outline="red",
        width=3
    )

    draw.text(
        (max(0, int(x1)), max(0, int(y1) - 15)),
        f"{name} {score_value:.2f}",
        fill="red"
    )

return output
```

Debugging habit

Always inspect the image after changing a threshold. The console tells you what the model returned; the visualisation tells you whether the spatial interpretation looks sensible.

Part 3 - Model Choice and Runtime Trade-offs

A robot is a real-time system. Model selection is therefore not simply an accuracy competition. The model must satisfy the task, hardware and timing constraints.

Requirement	Question
Detection quality	Are the predictions good enough for the task?
Latency	Can the robot react quickly enough?
Hardware	Can the platform run the model?
Memory	Does the model fit available memory?
Energy	Is the computational cost acceptable?
Failure cost	Which mistake is more damaging: false positive or false negative?

Week 5 environment note

The official workshop compares Faster R-CNN with SSDLite320-MobileNetV3. Your PyTorch 1.6.0 / torchvision 0.7.0 environment does not expose the SSDLite model. Do not install random APIs into the workshop code just to force the comparison. First understand the engineering objective: compare perception quality and computational cost.

1 Measure inference time correctly

```
from time import perf_counter

def timed_inference(model, image, device):
    # Warm-up avoids mixing one-time setup cost into the measurement.
    detect(model, image, device)

    times = []

    for _ in range(5):
        if device.type == "cuda":
            torch.cuda.synchronize()

        start = perf_counter()
        detect(model, image, device)

        if device.type == "cuda":
            torch.cuda.synchronize()

        times.append(perf_counter() - start)

    return times
```

Code	Why
perf_counter()	High-resolution wall-clock timing.
Warm-up inference	Avoids treating one-time setup as steady-state runtime.
torch.cuda.synchronize()	Ensures asynchronous GPU work has finished before timing.
Several runs	A single timing measurement is noisy.

Interpretation

Report what you observed. Do not claim that one model is universally faster or better. Hardware, image size, software versions and workload all affect timing.

2 A useful experiment

- Use the same image for every run.

- Warm up the model.
- Measure several inference runs.
- Report the mean (and, if useful, minimum/maximum).
- Change one factor at a time.
- Explain what the timing means for a robot task.

Part 4 - From Detection to Robot Decision

Up to this point the computer has said: “Here are some possible objects.” Robotics begins when we decide which candidate matters and what the robot should do.

1 Filter by class and confidence

```
valid = []

for detection in detections:
    if detection["class"] != TARGET_CLASS:
        continue

    if detection["score"] < CONF_THRESHOLD:
        continue

    valid.append(detection)
```

2 Select a target explicitly

```
if len(valid) == 0:
    target = None
else:
    target = max(valid, key=lambda d: d["score"])
```

Do not memorise “highest confidence” as the universal answer

Highest confidence is one possible policy. Other policies include closest to image centre, largest box, specific class priority, closest physical distance using depth, or a task-specific score. The important engineering requirement is that the policy is explicit, justified and tested.

Policy	Possible reason	Important assumption
Highest confidence	Strongest model evidence.	Score is useful for ranking candidates.
Closest to image centre	Easy visual approach rule.	Centre proximity relates to task success.
Largest box	Rough proximity cue.	Object size correlates with distance in the scene.
Specific class priority	Task has preferred object.	Class identity is reliable.
Closest physical distance	Need nearest object.	Depth/3D estimate is available and trustworthy.

3 Compute image-space target centre

```
x1, y1, x2, y2 = target["box"]

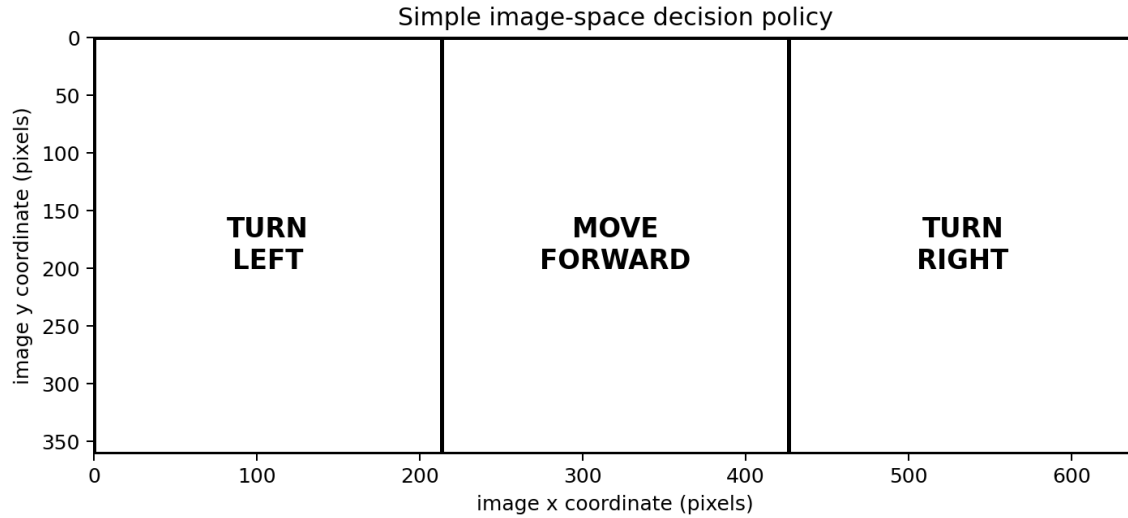
cx = (x1 + x2) / 2
cy = (y1 + y2) / 2

print("Target centre:", cx, cy)
```

Critical distinction

The bounding-box centre is an image coordinate. It is not automatically the physical centre of the object, and it is not a 3D robot coordinate. To obtain physical position you may need camera calibration, depth and coordinate-frame transformations.

4 Convert image location into a simple action



Example left/centre/right policy in image space.

```
def choose_action(target_center, image_width):  
    if target_center is None:  
        return "SEARCH"  
  
    cx, cy = target_center  
  
    if cx < image_width / 3:  
        return "TURN LEFT"  
    elif cx > 2 * image_width / 3:  
        return "TURN RIGHT"  
    else:  
        return "MOVE FORWARD"
```

Architecture insight

Perception -> information -> decision -> action. Keeping these boundaries separate makes the software easier to test and replace. You can change the detector without rewriting the action policy.

Failure-First Design

A robust perception system must define what happens when evidence is weak or missing. “No target” is not an exception to the design; it is one of the normal states of a perception system.

Situation	Possible response	Engineering question
No target	SEARCH / STOP / WAIT	What should the robot do with no evidence?
Confidence too low	Reject prediction	How much uncertainty can the task tolerate?
Several targets	Apply selection policy	Which target is relevant and why?
Target leaves image	SEARCH / TURN	How should the robot recover?
Unexpected class	Ignore / alternate perception	Should this object affect behaviour?
Unstable detections	Require temporal consistency	How much evidence over time is enough?

Robotics principle

Uncertainty should influence behaviour. A system that knows when it does not have a reliable target can be more useful than a system that always produces an action.

A small but important test

```
# Test the failure branch explicitly.
action = choose_action(None, image_width=640)
print(action)

# Expected:
# SEARCH
```

Mini-experiment: threshold changes behaviour

- Choose one image containing several objects.
- Run the detector once and store the raw result.
- Apply thresholds 0.30, 0.50 and 0.80 without rerunning the model.
- For each threshold, select a target.
- Compute the target centre.
- Apply the action rule.
- Record whether the final action changes.

Key question: If a small change in a threshold changes the robot action, what does that tell you about the importance of perception uncertainty?

Part 5 - How Do We Know the Detector Is Good Enough?

A successful-looking image is useful evidence, but it is not rigorous evaluation. We need a reference and measurable criteria.

1 Ground truth versus prediction

A ground-truth box is a reference annotation for where the object actually is. The detector produces a predicted box. The two boxes can then be compared geometrically.

2 Intersection over Union (IoU)

The formula is:

$$\text{IoU} = \text{area}(\text{intersection}) / \text{area}(\text{union})$$

High IoU means the predicted region overlaps the reference strongly. Low IoU means poor localisation. IoU is therefore different from the model's confidence score.

```
def box_iou(box_a, box_b):
    ax1, ay1, ax2, ay2 = box_a
    bx1, by1, bx2, by2 = box_b

    ix1 = max(ax1, bx1)
    iy1 = max(ay1, by1)
    ix2 = min(ax2, bx2)
    iy2 = min(ay2, by2)

    iw = max(0.0, ix2 - ix1)
    ih = max(0.0, iy2 - iy1)

    intersection = iw * ih

    area_a = max(0.0, ax2 - ax1) * max(0.0, ay2 - ay1)
    area_b = max(0.0, bx2 - bx1) * max(0.0, by2 - by1)

    union = area_a + area_b - intersection

    if union <= 0:
        return 0.0

    return intersection / union
```

Important

For the workshop, a prediction can be treated as a match when the class is correct and IoU is at least 0.50. This is a chosen evaluation rule, not a law of nature.

3 TP, FP and FN

Outcome	Meaning
TP - true positive	The system correctly detects a target.
FP - false positive	The system reports a target that should not be accepted.
FN - false negative	A real target is missed.

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

Metric	Question it answers	Failure emphasised
Precision	When the system predicts a target, how often is it correct?	False positives

Recall	Of the real targets, how many did it find?	False negatives
IoU	How well does the predicted box overlap the reference?	Localisation error
Runtime	Can perception run fast enough?	Latency / responsiveness

Which metric matters most?

It depends on the robot. Missing an obstacle can be more dangerous than a false alarm. In another task, selecting the wrong grasp target may be more costly than missing an irrelevant object. Evaluation must reflect failure consequences.

Evaluation Practice and Mini-Experiments

1 Manual IoU exercise

Use these boxes in the form [x1, y1, x2, y2]. Predict the IoU qualitatively before calculating it.

Case	Ground truth	Prediction	Your prediction
A	[10, 10, 50, 50]	[10, 10, 50, 50]	_____
B	[10, 10, 50, 50]	[20, 20, 60, 60]	_____
C	[10, 10, 50, 50]	[80, 80, 120, 120]	_____

Expected reasoning: A should be 1.0; C should be 0.0; B should lie between them.

2 Precision/recall exercise

Suppose a test produces TP = 8, FP = 2 and FN = 3.

- Calculate precision.
- Calculate recall.
- Which type of error is represented by FP?
- Which type is represented by FN?
- For an obstacle detector, which error might deserve more attention and why?

Worked check: precision = $8/(8+2) = 0.80$; recall = $8/(8+3)$ approximately 0.73.

3 Evaluation checklist

- Use more than one image.
- Include easy and difficult examples.
- Vary viewpoint, lighting, clutter or object size when relevant.
- Record the confidence threshold instead of hiding it.
- Inspect false positives and false negatives.
- Measure runtime if the robot has timing constraints.
- Evaluate the complete decision pipeline if perception controls behaviour.
- Record failure cases, not just successful examples.

Debugging: Find the First Wrong Stage

When the system fails, do not immediately rewrite the entire program. Find the first stage where the output becomes wrong.

- Can I load the image?
- Is the image in the expected format and size?
- Does the model run without errors?
- What are the raw boxes, labels and scores?
- Are the class IDs mapped correctly?
- Does filtering keep the detections I expect?
- Does target selection choose the intended object?
- Is the bounding-box centre correct?
- Does the left/centre/right action rule behave correctly?
- What happens when there is no target?
- Have I tested an example where perception fails?
- Have I evaluated the prediction against a reference?

Symptom	First check	Do not do this
No detections	Input image, class set, raw scores	Immediately lower threshold without understanding why.
Wrong object selected	Target-selection policy	Assume the detector must be wrong.
Action goes wrong	Centre calculation and image width	Rewrite the detector.
Too many detections	Threshold and class filter	Randomly change several parameters.
Missing target	Threshold, class, scene difficulty	Claim that a higher threshold is always safer.
Slow system	Timing method and hardware	Judge speed from one run.
Good-looking box but poor localisation	IoU against ground truth	Use confidence as a localisation metric.

Debugging rule

Inspect one boundary at a time: image -> model output -> filtered detections -> selected target -> image position -> action. The first incorrect boundary is usually the best place to investigate.

Reusable Robotics Vision Skeleton

This is the most transferable part of the chapter. When you later build another application, the model and task may change, but the software responsibilities can remain similar.

```
# -----
# Configuration
# -----
CONF_THRESHOLD = 0.50
TARGET_CLASS = "person"

# -----
# Perception
# -----
def load_model(device):
    ...

def load_image(path):
    ...

def preprocess(image):
    ...

def detect(model, image_tensor):
    ...

# -----
# Interpretation
# -----
def filter_detections(result, threshold, target_class):
    ...

def select_target(detections):
    ...

def bounding_box_center(box):
    ...

# -----
# Decision
# -----
def choose_action(target_center, image_width):
    ...

# -----
# Evaluation
# -----
def evaluate(predictions, references):
    ...

# -----
# Main pipeline
# -----
def main():
    model = load_model(device)
    image = load_image("example.jpg")
    image_tensor = preprocess(image)

    result = detect(model, image_tensor)
    detections = filter_detections(
        result, CONF_THRESHOLD, TARGET_CLASS
    )

    target = select_target(detections)
```

```

if target is None:
    action = "SEARCH"
else:
    centre = bounding_box_center(target["box"])
    action = choose_action(centre, image.width)

print("Action:", action)

```

Function	Responsible for	Should NOT be responsible for
load_model()	Loading/configuring the model	Choosing robot actions
detect()	Running inference	Selecting final action
filter_detections()	Class/confidence constraints	Hiding the decision rule
select_target()	Choosing relevant candidate	Running neural-network inference
bounding_box_center()	Image-space localisation	Claiming to provide 3D position
choose_action()	Task decision	Running neural-network inference
evaluate()	Measuring performance	Changing the model during evaluation

Why modularity matters

If perception and decision logic are mixed together, every model change can force a rewrite. If they are separated, you can replace Faster R-CNN with another detector while keeping the same filtering, selection and action interfaces.

Connection to Camera Geometry and Future Robotics

Week 4 taught you how to reason from 3D camera points to image pixels and back when depth is known. Week 5 gives you a learned way to obtain useful image-space evidence from real images.

Representation	Meaning	Typical information needed
Pixel / image space	Where is the target in the image?	Bounding box / centre
Camera frame	Where is it relative to the camera?	Calibration + depth / geometry
Robot/world frame	Where is it relative to the robot/environment?	Camera-to-robot/world transform

A bounding-box centre at (cx, cy) is therefore not enough to command a robot arm to a 3D point. A later system might combine detection with depth, camera calibration and a known camera-to-robot transform.

How the Week 5 skeleton can grow

Future capability	Extension
Video	Run perception repeatedly on frames.
Tracking	Associate detections across time.
Temporal filtering	Require stable evidence before acting.
Depth	Convert image location into a 3D estimate.
Navigation	Use objects as targets, landmarks or obstacles.
Manipulation	Combine detection with depth/pose and grasp planning.
Real robot	Replace printed actions with a validated robot-control interface.

Using AI as an Engineering Assistant

AI can speed up implementation, explanation and debugging, but generated code is a proposed solution, not proof of correctness. Give AI a well-defined component and keep responsibility for the architecture and tests.

Weak prompt	Better engineering prompt
"Write an object detection robot."	"Implement filter_detections() for this detector output. Keep TARGET_CLASS and score ≥ 0.50 . Explain assumptions."
"Fix my robotics code."	"Review only the target-selection logic. Identify logical errors first; do not rewrite unrelated functions."
"Make this faster."	"Profile this inference pipeline and identify the likely bottleneck. Preserve the model and output format."
"Add navigation."	"Extend the existing perception-to-decision interface with a left/centre/right policy. Keep perception code unchanged."

Useful prompts to copy

- Explain what boxes, labels and scores mean in this detector output. Do not rewrite the code.
- Review this filtering rule and identify cases where it could accept a wrong target or reject a correct target.
- Suggest five test images that would challenge this object detector and explain what each test checks.
- Review my target-selection policy. List its assumptions and suggest two alternative policies.
- Explain whether this bounding-box centre is an image coordinate, camera coordinate or robot coordinate.
- Review my IoU function and suggest edge cases to test.
- Given these TP, FP and FN values, calculate precision and recall and explain which failure is more important for my stated robot task.
- Do not rewrite the program. Identify the first stage where the output becomes inconsistent with the expected behaviour.

AI rule

Ask AI to work on a clearly defined component. You remain responsible for understanding the architecture, checking assumptions, testing the result and deciding whether the solution is appropriate.

Student Mini-Challenges

Challenge 1 - Threshold sensitivity

Use one image and thresholds 0.20, 0.30, 0.50, 0.70 and 0.90. Record the number of accepted detections and the selected target.

Threshold	# detections	Selected class	Selected score	Action
0.20	_____	_____	_____	_____
0.30	_____	_____	_____	_____
0.50	_____	_____	_____	_____
0.70	_____	_____	_____	_____
0.90	_____	_____	_____	_____

Explain: At what point does the system become too permissive or too selective for your task?

Challenge 2 - Change the selection policy

Replace highest-confidence selection with one of the following:

- closest to the image centre;
- largest bounding box;
- highest-confidence target of a preferred class.

Test at least three images. Record one case where the new policy gives a different target and explain why.

Challenge 3 - Deliberate failure

Create or find an image where the detector has difficulty. Do not immediately tune the model. Instead record:

Observe	Diagnose	Improve
What does the output look like?	Which assumption or evidence is failing?	What is the smallest change you would test first?
_____	_____	_____
_____	_____	_____

Challenge 4 - End-to-end reasoning

Suppose the robot must approach a bottle. Write the complete pipeline from image input to robot action. Include at least one uncertainty/failure branch and one evaluation method.

```

IMAGE
->
DETECT
->
FILTER
->
SELECT TARGET
->
LOCATE IN IMAGE
->
DECIDE
->
ACT
->
VERIFY

```

Failure branch: _____

Evaluation: _____

Self-Check

1. What are the three main outputs of an object detector?

2. Why does the detector receive a list [x] when there is only one image?

3. Why do we use `model.eval()` during inference?

4. Why do we use `torch.no_grad()`?

5. Why is the confidence threshold applied after inference?

6. Why is confidence not the same as correctness?

7. Why should target selection be a separate function?

8. What is the difference between a bounding-box centre and a 3D position?

9. What should happen when no reliable target is available?

10. What does IoU measure?

11. How are precision and recall different?

12. Why is one successful image not enough to validate a detector?

13. Why might a slightly less accurate model be a better robotics choice?

14. What is the first stage you should inspect when the final robot action is wrong?

15. How would you extend this Week 5 pipeline to use depth?

Quick Answer Guide

1. Boxes, labels and scores.
2. The detection API accepts a batch/list of images; [0] selects the first image's result.
3. It switches the model into inference/evaluation behaviour rather than training behaviour.
4. Gradients are not needed for prediction, so disabling them reduces unnecessary computation/memory use.
5. The neural network can produce raw candidates once; different application policies can be tested without rerunning inference.
6. A score is model evidence, not a guarantee that the object is correct or that the robot action will succeed.
7. It keeps perception replaceable and makes the decision rule independently testable.
8. A box centre is (cx, cy) in image pixels. A 3D position needs depth and coordinate transforms/calibration.
9. Use an explicit fallback such as SEARCH, STOP or WAIT, depending on the task.
10. The overlap between predicted and reference boxes: intersection area divided by union area.
11. Precision emphasises false positives; recall emphasises false negatives.
12. A single example can be lucky. Reliability requires multiple, varied tests and measurable reference-based evaluation.
13. It may satisfy latency, memory, energy or hardware constraints better.
14. Trace the pipeline from image -> raw model output -> filtering -> selection -> localisation -> action and find the first incorrect stage.
15. Combine the image-space detection with depth and camera calibration, then transform the camera-frame point into the robot/world frame.

Before You Trust a Vision-Based Robot

- I know what classes my model can recognise.
- I have inspected the raw boxes, labels and scores.
- I can explain what each output means.
- I have chosen and justified a confidence threshold.
- I have an explicit target-selection policy.
- I know whether my coordinates are image-space, camera-frame or robot/world coordinates.
- I have defined what happens when the target is missing.
- I have considered false positives and false negatives.
- I have considered inference speed and hardware constraints.
- I have tested difficult examples, not just one successful example.
- I have an evaluation method.
- I can identify the first wrong stage when the pipeline fails.
- I would not send an uncertain perception result directly into a physical action without an appropriate fallback/safety mechanism.

Final takeaway

A detector is not the robot. The transferable skill is the engineering structure: detect -> filter -> select -> locate -> decide -> act -> verify. At every arrow ask: What do I know? What am I assuming? What could fail? What evidence tells me the output is reliable enough?