Practical Object Detection with YOLO

Training Models on Real-World Datasets

Book 2 | Practical Computer Vision with Deep Learning

Dr. Ali Khan | **Dr. Somaiya Khan**

March 2026

Chapter 1

From Benchmark Experiments to Real-World Detection

Object detection research has historically relied on benchmark datasets such as COCO and PASCAL VOC to evaluate and compare algorithms. These benchmarks serve an important purpose: they provide controlled conditions, standardized annotations, and shared evaluation protocols that make algorithmic comparisons meaningful. A detector that achieves 50% mAP50–95 on COCO demonstrates stronger performance on that benchmark than one that achieves 40% under the same evaluation protocol.

However, benchmark performance is not the same as deployment performance. A model that performs well on COCO may struggle when applied to a real-world application, such as detecting vehicles from UAV imagery, monitoring infrastructure from aerial platforms, or identifying domain-specific objects in industrial environments. These failures do not necessarily indicate weaknesses in the model architecture. More often, they arise because the deployment environment differs substantially from the data on which the model was trained.

Understanding this mismatch, and learning how to reduce its impact, is one of the most important practical skills in applied object detection. This book focuses on that process.

1.1 The Gap Between Benchmarks and Practice

Large benchmark datasets are designed to represent a broad distribution of visual scenes. COCO, for example, contains 80 object categories covering people, animals, vehicles, household objects, and many other everyday classes captured primarily from natural ground-level viewpoints. This diversity is precisely what makes COCO a valuable benchmark.

Real detection problems, however, are rarely general-purpose. A UAV traffic monitoring system observes vehicles from a top-down perspective that differs substantially from the street-level imagery found in COCO. An infrastructure inspection system may encounter only a small number of object categories under highly repetitive backgrounds. A medical imaging application operates in a visual domain that shares almost no resemblance to natural photographs.

In each case, the deployment environment represents a specialized distribution that is only partially captured by benchmark datasets.

As the gap between the training environment and deployment environment increases, detection performance often deteriorates. Models may miss objects entirely, produce systematic misclassifications, or generate excessive false positives in scenes that differ substantially from those observed during training.

Table 1.1 summarizes some of the most common differences between benchmark datasets and real-world deployment environments.

Table 1.1: *Key differences between benchmark dataset conditions and typical real-world deployment environments.*

Dimension	Benchmark (e.g. COCO)	Real-World Application
Object categories	Broad (80 everyday classes)	Narrow and domain-specific
Viewpoint	Primarily ground-level	Often overhead, oblique, or fixed
Object scale	Mixed; typically moderate	May be consistently very small
Background	Diverse natural scenes	Often repetitive and domain-specific
Annotation quality	Expert-curated, consistent	Variable; annotation errors common
Class balance	Moderate imbalance	Often severe imbalance
Deployment constraint	None	Real-time, edge-device, or latency-limited

1.2 Domain Shift

The mismatch between training data and deployment data is commonly referred to as **domain shift**. Domain shift can significantly reduce detection performance even when the model architecture is appropriate and the training

Chapter 2

Defining the Detection Problem

Before a single image is collected or a bounding box is drawn, many of the most important decisions in an object detection project have already been made. These decisions determine what the model is expected to detect, how success will be measured, and what constraints must be satisfied during deployment. Every subsequent stage of the workflow, including dataset collection, annotation, training, evaluation, and deployment, depends on the quality of these early choices.

Poorly defined problems often produce datasets that do not reflect operational requirements, annotation guidelines that cannot be applied consistently, and models that perform well during evaluation but fail in real-world use. In contrast, a well-defined problem creates a clear framework that guides every technical decision throughout the project.

This chapter examines the key decisions involved in defining an object detection problem and explains why they matter in practice.

2.1 Identifying Target Objects

The first question in any detection project appears deceptively simple: what should the model detect?

In some applications, the answer is straightforward. A traffic monitoring system may simply need to detect vehicles. In other cases, the decision requires careful analysis. An aerial monitoring system may detect all vehicles as a single class, or it may distinguish between cars, trucks, and buses. Although both formulations address the same application, they lead to very different datasets, annotation requirements, training strategies, and performance expectations.

Several factors should be considered when defining object classes:

- **Visual distinguishability.** Can an annotator reliably distinguish between two proposed classes at the available image resolution? If not, the model is unlikely to distinguish them reliably either. In such cases, classes should be merged or higher-resolution imagery should be collected.
- **Operational relevance.** Does the application actually require class-level discrimination, or is object presence sufficient? In many aerial surveillance applications, detecting the presence of a vehicle may be sufficient for counting, monitoring, or traffic estimation. If class identity does not influence downstream decisions, a simpler single-class detector may outperform a multi-class detector trained on the same data.
- **Data availability.** Some object categories occur far less frequently than others. A class represented by only a small number of examples will be difficult to learn regardless of the model architecture.
- **Annotation feasibility.** Complex class definitions often require domain expertise and increase annotation cost. Simpler class definitions generally produce more consistent labels and higher-quality datasets.

Granularity is a commitment. Once a dataset has been annotated at a particular level of granularity, increasing that granularity later often requires re-annotating the entire dataset. Define object classes before annotation begins and validate them on a representative sample first.

2.2 Choosing Detection Granularity

Granularity refers to how finely objects are subdivided into classes. The choice of granularity involves a fundamental trade-off. Finer granularity provides more detailed information but requires more training data per class and creates a more difficult classification problem.

Table 2.1 illustrates how the same detection task can be formulated at different levels of granularity and the practical consequences of those choices.

Chapter 3

Dataset Collection Strategies

The quality of an object detection system is ultimately constrained by the quality of its training data. Model architectures continue to improve, but no architecture can compensate for a dataset that fails to represent the conditions encountered during deployment.

For this reason, dataset collection should not be viewed as a preliminary step before model training. It is a central part of the engineering process. Decisions made during collection determine what the model can learn, which failure modes it will encounter, and how well it will generalize beyond the training set.

This chapter examines the major sources of detection data, practical strategies for collecting representative imagery, and the challenges involved in building datasets for real-world applications.

3.1 Sources of Detection Data

Detection datasets can originate from several sources. Each source offers different advantages and limitations, making it suitable for different stages of a project.

Public benchmark datasets such as COCO, PASCAL VOC, and ImageNet-Det provide large annotated collections covering common object categories. These datasets are valuable for pretraining and benchmarking, but they rarely match the conditions of a specific deployment environment. They should be viewed as a starting point rather than a complete solution.

Domain-specific research datasets are collected for particular applications and operating environments. Examples include UCAS-AOD and UAVDT, both of which contain aerial imagery captured under conditions that differ substantially from conventional benchmark datasets. Such datasets often provide a strong foundation for domain adaptation and transfer learning.

Custom-collected imagery is gathered specifically for the target application. This approach requires the greatest effort but provides the highest level of control over image quality, environmental coverage, and deployment alignment. Most production systems rely on at least some custom data.

Synthetic data generated through rendering engines or simulation environments can provide virtually unlimited annotated images. Synthetic imagery is particularly useful when real-world data is expensive, difficult, or unsafe to collect. Its primary limitation is the difference between simulated and real-world visual characteristics.

Table 3.1 summarizes the major data sources used in object detection projects.

Table 3.1: *Comparison of common detection dataset sources across practical dimensions.*

Source	Domain Fit	Annotation	Volume	Best Use
Public benchmark	Low	Pre-made	High	Pretraining and baselines
Domain-specific dataset	Medium	Pre-made	Medium	Domain adaptation and fine-tuning
Custom collection	High	Manual	Variable	Production systems
Synthetic generation	Variable	Automatic	Unlimited	Rare classes and augmentation

Different sources offer different trade-offs between annotation cost, scalability, and deployment alignment. In practice, successful datasets often combine multiple sources rather than relying exclusively on one.

3.2 Collecting Images That Represent Deployment Conditions

The most common dataset collection mistake is collecting images under convenient conditions rather than deployment conditions. A model trained on images captured under ideal circumstances often performs poorly when deployed in the real world.

Effective collection strategies should deliberately capture the variability that the system is expected to encounter during operation.

Chapter 4

Annotation Workflows and Quality Control

Once images have been collected, the next step is to convert them into training data that a detection model can learn from. This process is known as annotation.

Although object detection annotation appears straightforward, it is one of the most important factors affecting model performance. A detector learns directly from its labels. If annotations are inconsistent, incomplete, or inaccurate, the resulting model will inherit those errors regardless of the architecture used.

For this reason, annotation should be viewed as a quality-critical engineering activity rather than a routine labeling task. Clear standards, consistent workflows, and systematic quality control procedures are essential for building reliable detection datasets.

This chapter examines annotation standards, YOLO annotation requirements, quality control procedures, and common annotation mistakes encountered in practical object detection projects.

4.1 Bounding Box Annotation Standards

Before annotation begins, clear written guidelines should define exactly how bounding boxes are drawn. Without explicit standards, different annotators often make different decisions, introducing inconsistencies that reduce dataset quality.

The following conventions should be documented before annotation starts:

- **Tight enclosure.** Bounding boxes should fit closely around the visible object boundary with minimal background included. Excessively loose boxes reduce localization accuracy during training.
- **Complete visible coverage.** The entire visible portion of the object should be enclosed. Bounding boxes that crop part of the object introduce incorrect localization signals.
- **Separate annotations for overlapping objects.** Each object should receive its own bounding box, even when objects overlap significantly.
- **Consistent occlusion handling.** Rules for partially visible objects should be defined before annotation begins and applied consistently throughout the dataset.
- **Truncated objects.** Objects near image boundaries should be annotated according to a predefined visibility threshold. A common rule is to annotate objects that remain at least 50% visible.

Run a calibration session before full annotation. Before beginning large-scale annotation, ask all annotators to label the same 20 to 30 images independently. Compare the results and discuss disagreements. This process often reveals ambiguities in the annotation guidelines before they propagate throughout the dataset.

Although annotation formats are relatively simple, understanding how bounding box coordinates are represented helps prevent formatting mistakes during dataset preparation.

Chapter 5

Dataset Preparation and Organization

Once images have been collected and annotated, the dataset must be organized in a format that the YOLO training pipeline can use efficiently. This involves creating the correct directory structure, defining a valid dataset configuration file, constructing appropriate training, validation, and test splits, and verifying the dataset before training begins.

Although these tasks may appear administrative, many training failures and misleading experimental results originate from mistakes made during dataset preparation. Careful organization ensures that models learn from the intended data and that reported performance accurately reflects real-world performance.

5.1 Directory Structure

The Ultralytics YOLO framework expects images and labels to be organized in parallel directory trees. Each image file must have a corresponding label file with the same filename and a `.txt` extension.

Images that do not contain target objects should still have an empty label file. Missing label files are interpreted differently from empty label files and may lead to unexpected behavior during training.

A standard directory structure is shown below:

Listing 5.1: *Standard YOLO dataset directory structure.*

```
dataset/
|-- images/
|   |-- train/
|   |-- val/
|   `-- test/
|
`-- labels/
|-- train/
|-- val/
`-- test/
```

The `images/` and `labels/` directories must exist under the same parent directory. The YOLO framework automatically locates labels by replacing the directory name `images` with `labels` in the file path.

Common path mistake. If annotations are stored in a directory named `annotations/` rather than `labels/`, the training pipeline will not locate them correctly. Always use the standard YOLO directory structure.

5.2 Creating the Dataset YAML Configuration

Every YOLO dataset requires a YAML configuration file describing dataset locations and class definitions.

Listing 5.2: *Example YOLO dataset configuration.*

```yaml
path: /path/to/dataset

train: images/train
val: images/val
test: images/test

nc: 2

names:
0: car
1: airplane
```

Several consistency rules must be satisfied:

- The value of `nc` must match the number of classes listed in `names`.
- Class indices in annotation files must correspond exactly to the order specified in `names`.

Chapter 6

Transfer Learning for Domain Adaptation

Most domain-specific object detection projects do not have enough data to train a deep neural network from scratch. Datasets such as UCAS-AOD contain only a few thousand object instances, which is far below the scale typically required for effective large-scale feature learning.

Fortunately, modern object detection models are rarely trained from random initialization. Instead, they begin from weights that have already been trained on large benchmark datasets such as COCO. This process, known as transfer learning, allows previously learned visual representations to be adapted to a new task with far less data and computational effort.

Transfer learning is one of the primary reasons why modern object detection systems can achieve strong performance on specialized datasets using only a few thousand training images. Understanding how transfer learning works and how to configure it correctly is therefore essential for applied detection projects.

This chapter examines how pretrained YOLO models are adapted to new domains, how transfer learning influences convergence behavior, and which training decisions have the greatest impact on performance.

6.1 Why Transfer Learning Works

A YOLOv11 model pretrained on COCO has already learned a rich hierarchy of visual features. Early layers detect edges, corners, and textures. Middle layers learn shapes, object parts, and structural patterns. Deeper layers learn increasingly semantic representations that help distinguish between object categories.

Many of these features remain useful even when the target dataset contains classes that do not exist in COCO. For example, aerial vehicles and aircraft in UCAS-AOD differ substantially from the ground-level imagery found in COCO, yet the underlying visual representations learned during pretraining still provide a valuable starting point.

During fine-tuning, the early layers often require only minor adjustments because their features are broadly applicable. The middle and deeper layers adapt to the new domain, while the detection head is largely relearned because its output depends directly on the classes defined in the target dataset.

As a result, transfer learning typically converges faster, requires less training data, and produces better performance than training from random initialization.

Table 6.1: *Typical differences between training from scratch and transfer learning for domain-specific object detection.*

Aspect	Training From Scratch	Transfer Learning
Training data requirement	Very large datasets typically required	Effective with a few thousand images
Convergence speed	Slow	Fast
Computational cost	High	Lower
Initial performance	Near random	Strong starting point
Generalization	Depends heavily on dataset size	Benefits from pretrained visual representations

6.2 Loading Pretrained Weights

The Ultralytics framework makes transfer learning straightforward. Loading a pretrained checkpoint requires only a few lines of code:

Listing 6.1: *Loading a pretrained YOLOv11 model.*

```
from ultralytics import YOLO

model = YOLO('yolo11n.pt')

model.info()
```