MSc in Computer science
@
Roskilde University

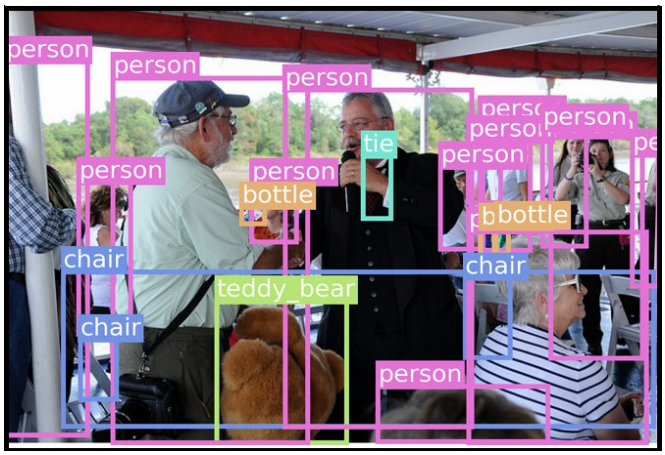
YOLO Variants vs. Pre-Trained Models: A Comparison in Object Detection.

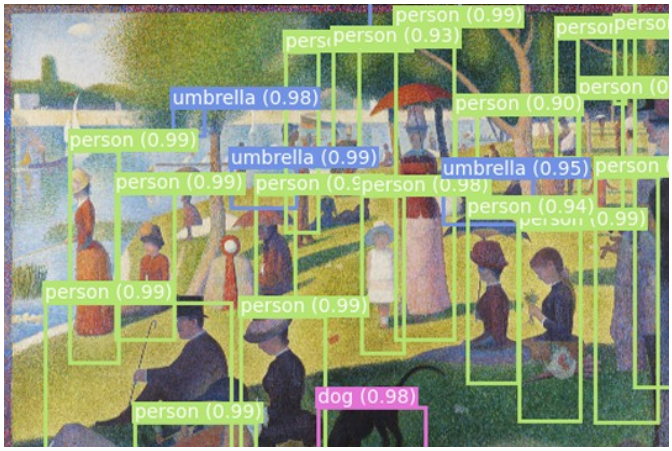
A Deep Learning Project

By

Abul Kasem Mohammed Omar Sharif
Md Sharif Uddin Khan

GitHub: https://github.com/sharifkea/Mini_project_deep_learning.git





Introduction

In our Deep Learning course, Chapter 12 of Deep Learning with Python covers object detection. It shows how to train a basic YOLO model on the COCO dataset and test it. The chapter also uses a pre-trained RetinaNet model for comparison by tested with the image "*A Sunday on La Grande Jatte*" (seurat.jpg). For our project, we built on this work. We trained a custom basic YOLO and a YOLOv8 model on a COCO subset. We compared their results. We also tested pre-trained Faster R-CNN and RetinaNet on the same seurat.jpg image. We kept the scope small: same data size, short training, and focus on detection counts.

What We Did in Code

We used four Jupyter notebooks for the project. In mini_train_custom.ipynb, we followed the book's basic YOLO setup. We loaded COCO data, filtered for images with ≤4 objects, trained for 4 epochs on a ResNet50 backbone, and saved the model as '*trained_yolo_model.keras*'. In mini_train_v8.ipynb, we trained YOLOv8 with the same data split (80/20 train/val) and epochs, saving as '*yolov8_retrained.pt*'. In mini_comp_cus_v8.ipynb, we loaded both models and ran predictions on a COCO test image (000000019489.jpg, with 21 ground-truth objects). In FasterRCNN_vs_RetinaNet.ipynb, we loaded pre-trained RetinaNet (Keras Hub) and Faster R-CNN (Torchvision) and tested on seurat.jpg.

We used a filtered COCO subset: about 94k images reduced to those with 1-4 bounding boxes. We shuffled and split the data. All code ran in the JAX backend. We added Matplotlib plots to visualize results.

Code Breakdown: Key Parts and Their Roles

All notebooks share core steps: load COCO JSON annotations, map image IDs to file paths, and scale bounding boxes to normalized [0,1] coordinates with the '*scale_box*' function (it pads images to a square). Metadata forms a list of dicts with boxes, labels, and paths.

- **Data Preparation (All Notebooks):** Loop through annotations to build per-image lists of scaled boxes and labels. Filter with '*len(set(tuple(box) for box in x["boxes"])) <=4*' for simple scenes. Shuffle and split 80/20. For YOLO formats, symlink images to '*yolo_dataset/images/labels/train|val*' and write .txt label files (class x_center y_center width height).
- **Training (mini_train_custom and mini_train_v8):** Custom YOLO uses a 6x6 grid (91 COCO classes), ResNet50 backbone, and custom loss (box regression + IoU confidence) on 448x448 crops. We save with '*model.save("trained_yolo_model.keras")*'. YOLOv8 loads via '*YOLO("yolov8n.pt"*', sets COCO classes, and trains with '*model.train(data="dataset.yaml", epochs=4*)'.
- **Inference and Visualization (mini_comp_cus_v8 and FasterRCNN_vs_RetinaNet):** Load models and predict. For custom YOLO, unpack outputs, threshold confidence >0.1, and draw boxes with 'draw_box' (Rectangle patches + HSV colors). YOLOv8 uses 'results = model(img)' and filters by confidence. Pre-trained models: RetinaNet with 'detector.predict(image)'; Faster R-CNN with tensor transforms and '*model(img_tensor)*'. Normalize boxes to rel_xywh. Use counters for class stats. We set PIL's MAX_IMAGE_PIXELS to None to load large images like seurat.jpg.

Visualization helpers ('*draw_image*', '*label_to_color*') appear everywhere. They scale images to [0,1] extents and add colored boxes with labels. The code stays short and clear.

Discussion of Results

We tested on two images. The COCO test image has 21 objects (13 people, 3 chairs, 3 bottles, 1 teddy bear, 1 tie). Our basic YOLO detected only 2 items with a confidence of (confidence=0.1). YOLOv8 did better: 7 at 0.5 and 11 at 0.1. YOLOv8 uses improved anchors and focal loss, which helps even with limited training.

For seurat.jpg (a painting with people, umbrellas, and a dog), pre-trained RetinaNet found 6 items (3 umbrellas, 3 people). It missed details in the art style. Faster R-CNN detected 26 (20 people, 5 umbrellas, 1 dog). As a two-stage model, it refines proposals well for subtle features.

Pre-trained models outperform our trained ones right away. YOLOv8 beats basic YOLO clearly. Lower thresholds show more detections but add false positives.

Model	Test Image	Detections (conf thresh)	Classes Hit
Basic YOLO	COCO (21 GT)	2 (0.1)	Limited detail
YOLOv8	COCO (21 GT)	7 (0.5), 11 (0.1)	People, chairs, etc.
RetinaNet (pre)	Seurat	6	Umbrella-3, person-3
Faster R-CNN (pre)	Seurat	26	Person-20, umbrella-5, dog-1

Insights from Our Results: Why Some Models Outperform Others

We noticed clear gaps in performance between our models, even though we trained the YOLO versions with the same data subset (images with ≤4 objects, 80/20 split) and just 4 epochs. Basic YOLO managed only 2 detections on the COCO test image, while YOLOv8 hit 11 (at conf=0.1). On seurat.jpg, pre-trained RetinaNet found 6 items, but Faster R-CNN spotted 26. Here is our take on why these differences happen.

Trained Models: Basic YOLO vs. YOLOv8
Basic YOLO follows the book's simple grid design (6x6 cells, basic box regression loss). It predicts one box per cell, which works for easy cases but struggles with overlaps or small objects in our 448x448 crops. With limited epochs, it underfits—too few parameters to learn COCO's variety.

YOLOv8, even starting from a nano preset and retrained briefly, pulls ahead thanks to its modern tweaks. It drops anchors for direct regression, uses focal loss to handle class imbalance, and has a CSP-like backbone for better features. These let it generalize faster, catching more people and chairs in the test image. Short training favors YOLOv8 because its architecture is more robust out-of-the-box.

Pre-Trained Models: RetinaNet vs. Faster R-CNN
RetinaNet is a solid one-stage detector with focal loss for hard examples, but it scans the whole image at once. On seurat.jpg's pointillist style (fuzzy edges, crowds), it misses subtle umbrellas and the dog—only 6 confident hits.

Faster R-CNN shines as a two-stage model. It first proposes regions (via RPN), then refines them with a separate classifier. This boosts precision on complex scenes like the painting, nailing 20 people and

extras. Pre-training on full COCO gives it an edge we couldn't match in 4 epochs—it's tuned for real-world nuances.

In short, better architectures (YOLOv8's efficiency, Faster R-CNN's refinement) and pre-training win over basics, no matter the equal setup. It shows why we iterate designs: raw training time isn't everything. This insight pushes us to try fine-tuning next.

Conclusion

Our project extends the book's object detection work. It shows progress in models like YOLOv8 and Faster R-CNN. Training from scratch teaches key ideas, but pre-trained options give strong results fast. With more epochs or data tweaks, our models could improve. Future steps might include fine-tuning on art images. This keeps the focus on practical comparisons.

References

1. Chollet, F. (2021). *Deep Learning with Python* (2nd ed.). Manning Publications. (Our starting point for the basic YOLO setup and RetinaNet testing—Chapter 12.)
2. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (The original YOLO paper; we built our custom version on this.)
3. Lin, T.-Y., et al. (2017). Feature Pyramid Networks for Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (Backbone for RetinaNet and Faster R-CNN insights.)
4. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in Neural Information Processing Systems (NeurIPS)*. (Explains the two-stage magic we saw in results.)
5. Jocher, G., et al. (2023). Ultralytics YOLOv8. GitHub repository. https://github.com/ultralytics/ultralytics. (For our YOLOv8 training and comparisons.)