IMPROVING YOLOV8 FOR FAST FEW-SHOT OBJECT DETECTION BY DINOV2 DISTILLATION

Guillaume Fourret[*] [†1,2] *Marc Chaumont*[1,3] *Christophe Fiorio*[1] *Gérard Subsol*[1]

[1]ICAR research team, LIRMM, Univ Montpellier, CNRS, Montpellier, France
[2]Drone Geofencing, Nîmes, France
[3]University of Nîmes, France

ABSTRACT

Recent neural networks for object detection achieve excellent performance when trained on large databases but still struggle to learn new objects with few examples, leading to the development of Few-Shot Object Detection (FSOD). State-of-the-art FSOD methods often use computationally heavy architectures like Faster R-CNN or pre-trained Vision Transformers (ViTs) such as DINOV2, limiting their usage for real-time inference in resource-constrained environments. We propose a novel approach that transfers rich features from a pre-trained ViT (DINOV2) to the lightweight YOLOv8 architecture via knowledge distillation. This bridges the gap between FSOD performance and efficiency, enabling YOLOv8 to better handle few-shot scenarios while retaining computational efficiency. Experiments on the MSCOCO benchmark adapted for FSOD show that our method enhances the performance of lightweight detectors, highlighting the benefits of combining ViT feature learning with efficient detectors for real-world FSOD.

Index Terms— DINOv2, YOLOv8, Real-Time, Few-Shot Object Detection, Distillation

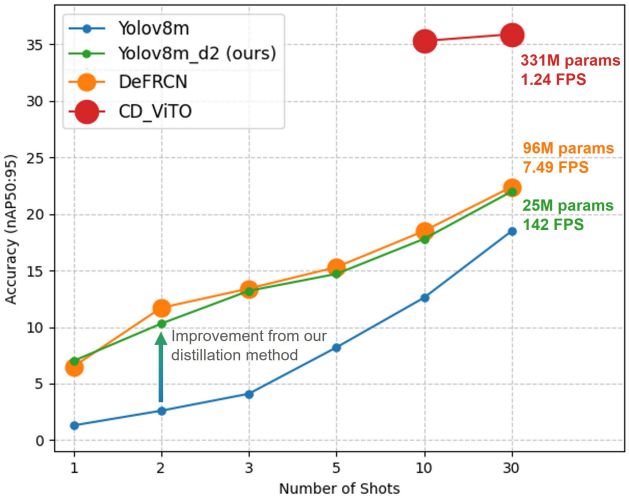

Fig. 1. Comparison of our proposed distillation method, YOLOv8m_d2, with the baseline YOLOv8m and other state-of-the-art methods, highlighting the trade-off between accuracy on novel classes at multiple shots, parameter efficiency, and FPS achieved by our approach.

1. INTRODUCTION

Recent deep learning models have improved image processing tasks such as 2D classification, segmentation, and object detection (defined by both localization and classification). These models need large labeled databases, which are hard to create when images are rare and labeling is costly and labor-intensive.

The field of Few-Shot Object Detection (FSOD) [1], which consists for a detector to learn a new object with few examples, has therefore become increasingly popular. Two main training paradigms have emerged in FSOD, meta-learning [2] and transfer-learning [3], both based on a first pre-training stage followed by a fine-tuning one.

However, most state-of-the-art FSOD methods rely on computationally intensive architectures, such as Faster R-CNN [4] or large pre-trained ViTs like DINOV2 [5], [6], making them unsuitable for real-time applications on devices with limited computational resources and/or memory size, particularly in embedded systems.

Recently, more advanced one-stage object detectors, particularly the YOLO family [7], have gained popularity for their superior efficiency and performance in practical applications. However, YOLO models typically require extensive labeled data and are not specifically tailored for FSOD tasks.

In this article, we present in Sec. 2 the transfer-learning paradigm in FSOD along with key methods in the field, notably the ones using Self-Supervised Learning (SSL) pre-trained transformers, and the principle of distillation [8]. Next, in Sec. 3, we describe our approach to performing knowledge distillation from DINOv2 to YOLOv8 [9], taking

[*]Corresponding author: guillaume.fourret@lirmm.fr

[†]This work is partially funded by the ANRT (Association Nationale de la Recherche et de la Technologie). This work was performed using HPC resources from GENCI-IDRIS (Grant 2024-AD011015472)

in account the difference of architecture between ViT and CNN. Finally, in Sec. 4, we assess the effectiveness of the distillation process and evaluate the proposed approach on the FSOD benchmark MSCOCO [10], demonstrating performance improvements with our distilled YOLOv8 model as shown in Fig. 1.

2. RELATED WORK

2.1. General FSOD consideration

Few-Shot Object Detection (FSOD) methods follow two stages [1]. First, during pre-training, the detector is trained on a large labeled dataset to learn detecting base classes, C_{base}. Then, in fine-tuning, the model is adapted to both the base classes and novel classes, C_{novel}, using K-shot samples (i.e., K labeled bounding boxes). These novel classes are entirely distinct from the base ones ($C_{base} \bigcap C_{novel} = \emptyset$), resulting in an N-way K-shot setup, where N is the total number of classes.

Initially, FSOD was tackled with two primary approaches: meta-learning and transfer learning. Meta-learning methods, such as [2], use episodic training to simulate N-way K-shot tasks by sub-sampling data during pre-training. This helps the model quickly adapt to novel objects using few examples, often relying on siamese networks for support-query pair comparisons [11]. However, transfer learning [3] has emerged as the dominant paradigm for FSOD, surpassing meta-learning in performance. It usually employs classical supervised learning to train a detector on C_{base} during pre-training, then fine-tunes to transfer features to C_{novel} with limited examples.

2.2. Transfer learning methods in FSOD and SSL

Transfer learning methods initially relied on Faster R-CNN [4], with TFA [3] demonstrating that freezing the backbone during fine-tuning outperformed meta-learning approaches. Later, DeFRCN [12], one of the most effective Faster R-CNN-based method, addressed the challenge of balancing classification and localization in few-shot scenarios using a gradient decoupling strategy between Region Proposal Network (RPN), RCNN-head and backbone.

Recently, transfer-learning approaches in FSOD have significantly benefited from transformer architectures and Self-Supervised Learning (SSL), which enable the extraction of high-quality features. ImTED [13] improves standard detection tasks by using a ViT encoder pre-trained with SSL as the backbone for Faster R-CNN. It further enhances FSOD by replacing the conventional detector head with a Masked Autoencoder (MAE) [14] decoder, allowing the entire model to undergo MAE-based pre-training. This approach minimizes the number of randomly initialized parameters during fine-tuning, leading to better results in FSOD scenarios. Similarly, DETReg [15] pre-trains the entire detection architecture using SSL within the DETR [16] framework. It introduces two SSL pretext tasks: generating pseudo-labels for detection using Selective Search [17] and predicting object embeddings derived from a pre-trained SwAV [18] model.

DINOv2 [5], [6] has then emerged as one of the most effective ViTs for feature extraction, thanks to its large-scale SSL pre-training on the private LVD-142M dataset. DE-ViT [19] utilizes DINOv2 as a frozen backbone to pre-calculate class and background prototypes from support images, refining RPN-generated bounding boxes with a learnable propagation network. CD-ViTO [20] improves DE-ViT and achieves state-of-the-art performance by converting pre-computed class prototypes into learnable features, emphasizing high-quality shots via an multi-layer perceptron, and applying domain adaptation to improve robustness to domain shifts. Finally, FM-FSOD [21] uses cross-attention between class prototypes and query features extracted by DINOv2, generating bounding boxes via a transformer decoder inspired by DETR. A large language model (LLM) integrates prototypes, proposals, and class names for final classification.

While effective, these methods rely on large frozen models, limiting their use in resource-constrained real-time settings where lightweight detectors like YOLO excel but are very underrepresented in FSOD. This prompts the question: *How can we exploit DINOv2's high-quality features for FSOD while retaining YOLOv8's speed and efficiency?*

2.3. The interest of distillation methods

Knowledge distillation [8] involves transferring knowledge from a large, complex teacher model to a smaller, more efficient student model. This technique has been applied to object detection [22], where smaller Faster R-CNN models were trained using the teacher's classification, regression outputs, and intermediate layer features as soft labels.

A recent and effective variant of knowledge distillation is self-distillation, as demonstrated by the object detector D-FINE [23], which uses it to constrains the nature of information propagated through the network. DINOv2[5], [6] also uses self-distillation in a self-supervised learning setup, where both the teacher and student share the same architecture and learn simultaneously, with the teacher directing the learning process.

In our work, two key questions arise: *How can we effectively transfer knowledge from DINOv2's large ViT architecture to YOLOv8's lightweight CNN design? Furthermore, is distillation beneficial in a Few-Shot Object Detection setting?*

3. DINOV2 DISTILLATION IN YOLOV8

We explored three distinct distillation strategies to transfer knowledge from DINOv2 to YOLOv8. A key objective of these methods is to ensure that all distillation weights are removable at inference, retaining YOLOv8's original speed and memory efficiency while benefiting from improved training.

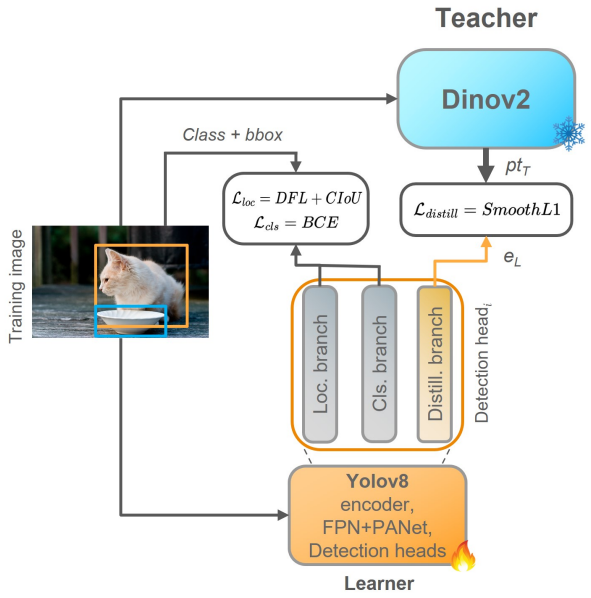

Fig. 2. Schematic overview of our first distillation approach, featuring the addition of a distillation branch alongside the original two branches in the i-th detection head.

Our first approach, named **YOLOv8m_d1** and summarized in Fig. 2, operates at the level of the detection heads. YOLOv8 originally features three detection heads at different spatial resolutions, achieved through a Feature Pyramid Network and PANet, with two parallel branches dedicated to localization and classification tasks (in gray on the figure). In this approach, we introduce an additional distillation branch (in orange) within each detection head. These branches maintain the same resolution as the others and employs a 1×1 convolution to produce an embedding of size $d = 1024$ for each pixel, matching the embedding size of DINOv2l (with $d = 1024$ corresponding to the Large version of DINOv2; the exact value depends on the specific DINOv2 variant used for distillation). During training, we process the images simultaneously through YOLOv8 and DINOv2, extracting the patch tokens from the final multi-head attention layer of DINOv2. The distillation is performed by computing a SmoothL1 loss between the normalized output embeddings of YOLOv8's distillation branch e_L and DINOv2's patch tokens pt_T, resized to match the spatial resolution:

$$\mathcal{L}_{\text{SmoothL1}}(e_L, pt_T) = \begin{cases} 0.5 \cdot (e_L - pt_T)^2, & \text{if } |e_L - pt_T| < 1, \\ |e_L - pt_T| - 0.5, & \text{otherwise.} \end{cases}$$

Our two other methods are more detailed in Fig. 3. The second one, **YOLOv8m_d2** (green arrows), eliminates the need for a separate distillation branch by integrating the distillation mechanism directly into the original classification branch of YOLOv8. Specifically, we extend the last convolution of this branch by adding $d = 1024$ channels, enabling it to simultaneously process both the original classification outputs (green block in the figure) and the embedding predictions (red block). As before, the SmoothL1 loss is applied to the embedding predictions.

Finally, our third method **YOLOv8m_d3** (red arrows) builds on the second approach by incorporating the distillation signal into the localization branch. In ViT architectures, much of the localization information is embedded within attention maps, which capture the correlations between patches. However, since YOLOv8 lacks an attention mechanism, directly predicting attention maps would be suboptimal, as it cannot inherently model correlations beyond its receptive fields. To address this, we extend the localization branch to predict embeddings (red block), similar to the classification branch. These embeddings are then fed into a new multi-head self-attention layer (noted "Self-Attention$_i$" in Fig. 3), from which attention maps (blue block) are extracted. These attention maps, alongside those from DINOv2, are used for the distillation process calculated with a SmoothL1 loss.

Additionally, in all our methods, DINOv2 remains frozen throughout the training process to preserve its strong generalization capabilities. Distillation is applied during both pre-training and fine-tuning. In pre-training, it enhances YOLOv8's internal representations and aligns its latent space with DINOv2, while in fine-tuning, it provides a stable reference to guide the model.

Note that our method focuses on YOLOv8 due to its competitive accuracy compared to newer versions and its well-established reliability. However, we emphasize that our approach can be seamlessly integrated into the latest YOLO11, as both versions share the same detection head architecture for even faster inference speed.

4. EXPERIMENTS

We evaluated our methods on the standard MSCOCO benchmark [10], which consists of 123,287 images annotated across 80 classes. All images are used for training, except for a reserved subset of 5,000 images designated for validation. To adapt MSCOCO for FSOD [3], the dataset is divided into 60 base classes (C_{base}) for the pre-training stage. During the K-shot fine-tuning stage, labels for the remaining 20 classes (C_{novel}) are introduced alongside the base classes. Performance metrics include mAP50 and mAP50:95, reported separately for base and novel classes as bAP50, bAP50:95, nAP50, and nAP50:95. For implementation, we utilized the medium version of YOLOv8 (YOLOv8m) by Ultralytics and the large version of DINOv2 (DINOv2l) with the addition of registers from [6] to produce cleaner attention maps, useful for YOLOv8m_d3.

4.1. Pre-training stage

We first performed the pre-training stage in a standard supervised learning setup on the base classes C_{base}, using the original YOLOv8m as the baseline and applying all DINOv2

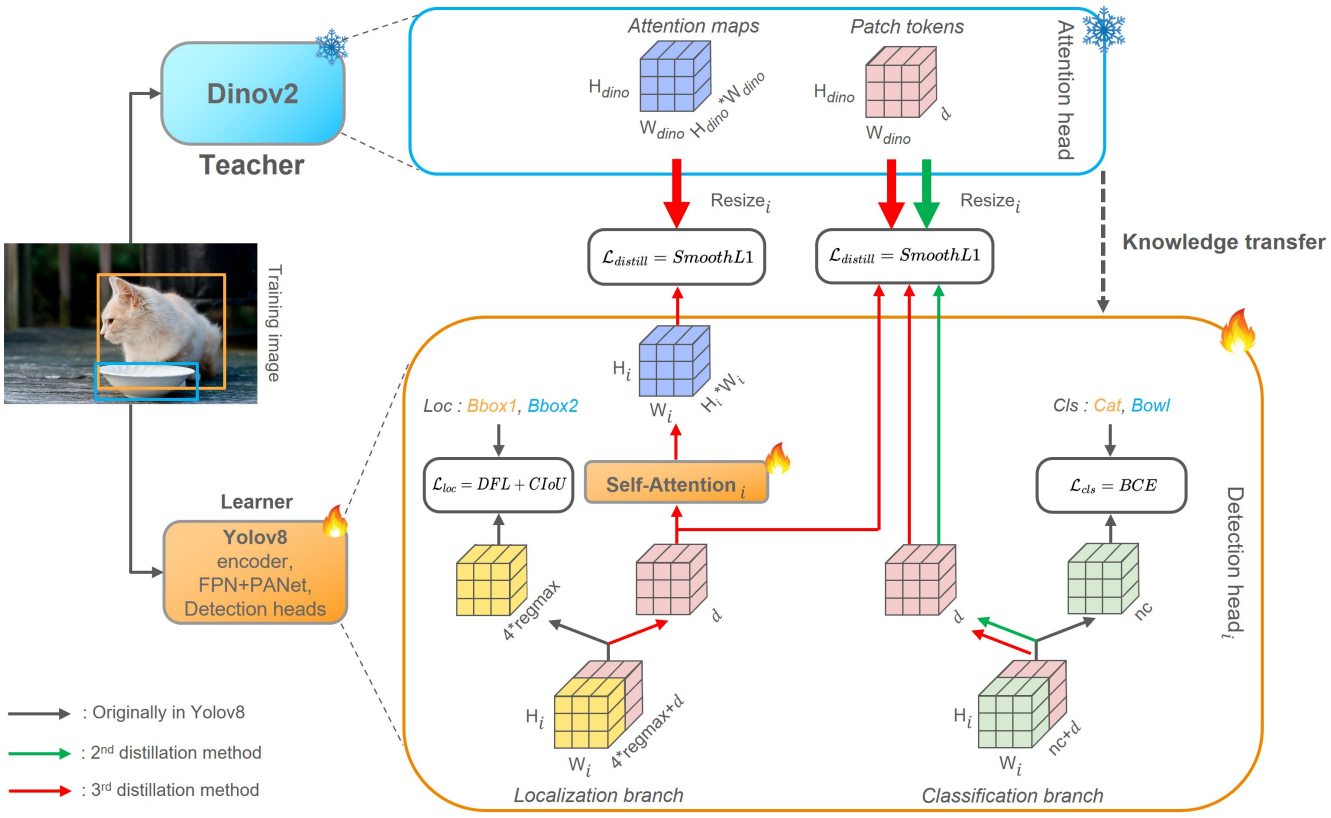

Fig. 3. Diagram illustrating our second approach (YOLOv8m_d2, green arrow) and third approach (YOLOv8m_d3, red arrow) for distilling DINOv2 into YOLOv8 for Few-Shot Object Detection. The figure highlights the i-th detection head among the three in YOLOv8. "Regmax" and "nc" are dimensions initially introduced in YOLOv8 for computing the Distribution Focal Loss (DFL, localization) and Binary Cross-Entropy (BCE, classification). Distillation is performed during both pre-training and fine-tuning stages, with DINOv2 remaining frozen throughout the training process.

Table 1. Mean average performance on the 60 base classes C_{base} of MSCOCO after pre-training.

Pre-training metrics	bAP50	bAP50:95
YOLOv8m vanilla [9]	61.25	45.62
YOLOv8m_d1	62.40	46.61
YOLOv8m_d2	62.47	46.70
YOLOv8m_d3	62.44	46.15

distillation strategies. The training utilized an SGD optimizer with a batch size of 512 distributed across 8 Nvidia A100 GPUs, employing a cosine learning rate schedule and an initial learning rate of $\mathrm{lr} = 10^{-2}$ for 450 epochs. The performance metrics for each method are summarized in Tab. 1. All distillation strategies outperformed the baseline training, achieving up to a +1.2 bAP50 improvement in the best case.

4.2. K-shot fine-tuning

The fine-tuning process was conducted using an SGD optimizer with a batch size of 16 on a single Nvidia A100 GPU. The initial learning rate was set to $\mathrm{lr} = 10^{-3}$, and training was performed for 250 epochs. Following standard practices in FSOD on MSCOCO, we evaluated the models with $K = 1, 2, 3, 5, 10, 30$ shots. The results, presented in Tab. 2, demonstrate that our distilled versions achieve superior metrics for novel class learning (nAP50, nAP50:95) compared to the baseline YOLOv8m (vanilla). We can also remark that the first approach, which introduces a separate branch for distillation, performs worse than the other two methods that integrate distillation directly into the original branches. This observation aligns with findings from [13], [15], which suggest that to achieve improved results, a greater portion of the model's parameters should benefit from pre-training and, in our case, have direct access to the distillation signal. Moreover, we observe that YOLOv8_d2 achieves the best performance in the extremely low-shot regime (e.g., 1–5 shots), whereas YOLOv8_d3 demonstrates better scalability as the number of shots increases, particularly at 10 and 30 shots.

The key advantage of our methods is that the distilled YOLOv8 retains the same inference speed and number of parameters as the original model while delivering improved

Table 2. Results of the K-shot fine-tuning on the first MSCOCO seed from [3], comparing the YOLOv8m baseline, our distilled models, DeFRCN (results from their GitHub [2]), and CD-ViTO (note that their results for 10 and 30 shots are averaged over multiple seeds and included for approximate comparison).

	1-shot				**2-shot**			
	bAP50	**bAP50:95**	**nAP50**	**nAP:5095**	**bAP50**	**bAP50:95**	**nAP50**	**nAP:5095**
YOLOv8m vanilla [9]	46.3	33.0	1.7	1.3	44.9	32.2	4.0	2.6
YOLOv8m_d1	49.9	35.7	2.9	2.5	46.8	33.7	3.6	2.6
YOLOv8m_d2	50.1	35.9	9.9	7.0	47.6	34.2	14.4	10.3
YOLOv8m_d3	45.2	32.4	6.2	4.9	43.6	31.2	10.2	7.0
DeFRCN [12]	48.7	31.8	10.9	6.5	49.7	32.5	20.6	11.7
CD-ViTO [20]	-	-	-	-	-	-	-	-
	3-shot				**5-shot**			
	bAP50	**bAP50:95**	**nAP50**	**nAP:5095**	**bAP50**	**bAP50:95**	**nAP50**	**nAP:5095**
YOLOv8m vanilla [9]	43.1	31.2	6.5	4.1	42.9	30.8	12.7	8.2
YOLOv8m_d1	42.3	30.6	14.3	8.1	43.7	31.3	18.8	10.2
YOLOv8m_d2	48.1	34.5	18.1	13.2	50.1	35.7	19.8	14.7
YOLOv8m_d3	40.5	28.8	13.6	8.3	42.0	29.8	20.6	13.8
DeFRCN [12]	49.8	32.5	24.2	13.4	50.6	33.1	28.4	15.3
CD-ViTO [20]	-	-	-	-	-	-	-	-
	10-shot				**30-shot**			
	bAP50	**bAP50:95**	**nAP50**	**nAP:5095**	**bAP50**	**bAP50:95**	**nAP50**	**nAP:5095**
YOLOv8m vanilla [9]	45.3	32.0	22.0	12.6	43.4	30.8	30.5	18.5
YOLOv8m_d1	45.5	32.4	23.1	15.7	46.8	33.3	32.7	21.5
YOLOv8m_d2	52.6	37.3	24.2	17.8	46.5	32.9	33.2	22.0
YOLOv8m_d3	44.1	31.4	29.2	18.6	44.4	31.3	37.9	25.1
DeFRCN [12]	53.1	34.5	34.5	18.5	52.8	34.6	39.9	22.4
CD-ViTO [20]	-	-	54.9	35.3	-	-	54.5	35.9

Table 3. Comparison of model sizes (number of parameters) and inference speeds (Frames Per Second, FPS) measured on a single Nvidia RTX A6000 GPU using PyTorch (without TensorRT optimization).

	Number of Parameters ↓	**FPS** ↑
YOLOv8m [9]	25,902,640	142
DeFRCN [12]	96,754,958	7.49
CD-ViTO [20]	331,149,640	1.24

performance compared to the vanilla version. Table 3 presents the number of parameters and FPS for YOLOv8m, as well as for DeFRCN (with 4 times more parameters and 19 times lower FPS) and CD-ViTO (with 12 times more parameters and 114 times lower FPS). These results underscore YOLOv8's suitability for applications with limited computational and memory resources.

[2] https://github.com/er-muyue/DeFRCN

5. CONCLUSION

In this study, we focused on the YOLOv8 architecture as a prominent example of lightweight detectors, which are underrepresented in the field of Few-Shot Object Detection (FSOD). We developed knowledge distillation schemes leveraging DINOv2 as the teacher model to provide a strong supervisory signal and to harness its high-quality features, despite the architectural differences between the two models. Our experiments on the MSCOCO dataset demonstrated that these distillation methods effectively enhance YOLOv8's performance in FSOD scenarios without adding extra parameters or computational overhead during inference. Future work could investigate the effectiveness of these approaches in Cross-Domain FSOD [20] scenarios and real-world applications, where substantial differences exist between the domains and object categories used in pre-training and fine-tuning.

References

[1] M. Köhler, M. Eisenbach, and H.-M. Gross, "Few-shot object detection: A comprehensive survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[2] X. Wu, D. Sahoo, and S. Hoi, "Meta-rcnn: Meta learning for few-shot object detection," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 1679–1687.

[3] X. Wang, T. Huang, J. Gonzalez, *et al.*, "Frustratingly simple few-shot object detection," *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, H. D. III and A. Singh, Eds., pp. 9919–9928, Apr. 2020. [Online]. Available: https://proceedings.mlr.press/v119/wang20j.html.

[4] S. Ren, K. He, R. Girshick, *et al.*, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[5] M. Oquab, T. Darcet, T. Moutakanni, *et al.*, "Dinov2: Learning robust visual features without supervision," *arXiv:2304.07193*, 2023.

[6] T. Darcet, M. Oquab, J. Mairal, *et al.*, "Vision transformers need registers," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=2dnO3LLiJ1.

[7] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023.

[8] G. Hinton, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[9] G. Jocher, J. Qiu, and A. Chaurasia, *Ultralytics YOLO*, version 8.0.0, Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

[10] Michael, B. Serge, H. James, *et al.*, "Microsoft coco: Common objects in context," *Computer Vision – ECCV 2014*, Tomas, S. Bernt, T. T. F. David, *et al.*, Eds., pp. 740–755, 2014.

[11] Q. Fan, W. Zhuo, C.-K. Tang, *et al.*, "Few-shot object detection with attention-rpn and multi-relation detector," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4013–4022, 2020.

[12] L. Qiao, Y. Zhao, Z. Li, *et al.*, "Defrcn: Decoupled faster r-cnn for few-shot object detection," *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8681–8690, 2021.

[13] F. Liu, X. Zhang, Z. Peng, *et al.*, "Integrally migrating pre-trained transformer encoder-decoders for visual object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 6825–6834.

[14] K. He, X. Chen, S. Xie, *et al.*, "Masked autoencoders are scalable vision learners," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16 000–16 009, 2022.

[15] A. Bar, X. Wang, V. Kantorov, *et al.*, "Detreg: Unsupervised pretraining with region priors for object detection," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14 605–14 615, 2022.

[16] N. Carion, F. Massa, G. Synnaeve, *et al.*, "End-to-end object detection with transformers," in *European conference on computer vision*, Springer, 2020, pp. 213–229.

[17] J. R. Uijlings, K. E. Van De Sande, T. Gevers, *et al.*, "Selective search for object recognition," *International journal of computer vision*, vol. 104, pp. 154–171, 2013.

[18] M. Caron, I. Misra, J. Mairal, *et al.*, "Unsupervised learning of visual features by contrasting cluster assignments," *Advances in neural information processing systems*, vol. 33, pp. 9912–9924, 2020.

[19] X. Zhang, Y. Liu, Y. Wang, *et al.*, "Detect everything with few examples," in *8th Annual Conference on Robot Learning*, 2024. [Online]. Available: https://openreview.net/forum?id=HlxRd529nG.

[20] Y. Fu, Y. Wang, Y. Pan, *et al.*, "Cross-domain few-shot object detection via enhanced open-set object detector," in *European Conference on Computer Vision*, Springer, 2025, pp. 247–264.

[21] G. Han and S.-N. Lim, "Few-shot object detection with foundation models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 28 608–28 618.

[22] G. Chen, W. Choi, X. Yu, *et al.*, "Learning efficient object detection models with knowledge distillation," *Advances in neural information processing systems*, vol. 30, 2017.

[23] Y. Peng, H. Li, P. Wu, *et al.*, *D-fine: Redefine regression task in detrs as fine-grained distribution refinement*, 2024. arXiv: 2410.13842 [cs.CV].