Adaptation de Yolov8 pour la détection d'objets avec peu d'exemples

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

[1] Équipe ICAR, LIRMM, Univ. Montpellier, CNRS, France
[2] Univ. Nîmes, France
[3] Drone Geofencing, Nîmes, France

{guillaume.fourret, fiorio, gerard.subsol, marc.chaumont}@lirmm.fr

Résumé

Les réseaux récents pour la détection d'objets obtiennent d'excellentes performances quand ils sont entraînés sur de grandes bases de données, mais ont toujours des difficultés pour apprendre un nouvel objet avec peu d'exemples. Les méthodes de ce domaine utilisant plutôt des architectures lourdes, nous avons décidé d'adapter les modules présentés dans DeFRCN dans l'architecture plus récente et rapide de Yolov8. Nous montrons ici l'impact de cette modification sur le benchmark MSCOCO, et finalement parlons des biais de cette méthode.

Mots-clés

Yolov8, Détection d'objets en peu d'exemples, Apprentissage par transfert

Abstract

Recent networks for object detection obtain excellent performance when trained on large databases but still have difficulties learning a new object with few examples. The methods of this domain using rather heavy architectures, we have decided to adapt the modules presented in DeFRCN into the newer and faster architecture of Yolov8. We show here the impact of this modification on the MSCOCO bechmark, and finally discuss about the biases existing in this method.

Keywords

Yolov8, Few-Shot Object-Detection, Transfer Learning

1 Introduction

Les progrès récents en architectures et méthodes d'entraînement des réseaux neuronaux ont grandement amélioré les performances en vision par ordinateur, notamment en classification, segmentation et détection d'objets (définie par la localisation et classification). La plupart des modèles sont entraînés de manière supervisée sur de vastes ensembles de données annotées, souvent difficiles et coûteux à obtenir.
Les méthodes d'apprentissage avec peu d'exemples [5] ("Few-Shot Object Detection", FSOD) ont émergé pour répondre à ce problème. Deux paradigmes de ce domaine sont le méta-apprentissage [10], qui simule des entraînements avec peu d'exemples et l'apprentissage par transfert [12], qui vise à acquérir de bonnes capacités de représentation sur des ensembles de données volumineux pour les transférer à de nouveaux objets avec peu d'exemples.
Un défi notable dans ce dernier est la capacité d'un détecteur à séparer la localisation et la classification, comme présenté dans DeFRCN [3] avec leur couche de découplage de gradient (GDL). Pour améliorer la classification, un second module, le PCB, utilise des vecteurs supports (prototypes) pré-calculés. Cependant, cette méthode est spécifique à l'architecture en deux étapes du Faster R-CNN [7]. Des détecteurs plus récents en une seule étape, tels que ceux de la famille YOLO, sont apparus offrant des performances et une vitesse d'inférence supérieure. Dans cet article, nous montrons comment nous avons intégré ces deux modules dans l'architecture de Yolov8 [2] pour améliorer ses performances avec peu d'exemples d'entraînement.

2 État de l'art

2.1 Méthodes et paradigmes du FSOD

Toutes les approches suivent un schéma d'entraînement en 2 étapes. Une première base de données est utilisée pour le pré-entraînement sur les classes de base. Une deuxième base de données sert ensuite pour l'apprentissage des nouvelles classes et est constituée de K exemples ("K-shots") de *toutes* les classes.
Le méta-apprentissage [10] simule des tâches FSOD en sous-échantillonnant K-shots des classes de base lors du pré-entraînement. La plupart des méthodes utilisent une branche en parallèle d'un Faster R-CNN pour créer des vecteurs supports/prototypes représentant chaque classe. Ces techniques diffèrent principalement par l'endroit où ces vecteurs sont agrégés dans le Faster R-CNN (avant [10] ou [6] après le "Region Proposal Network" RPN) et par l'opération d'agrégation utilisée (produit simple, attention...).
L'apprentissage par transfert a émergé récemment comme un paradigme plus performant. Dans [12], un simple apprentissage (en gelant l'encodeur) d'un Faster R-CNN pré-entraîné a surpassé les méthodes de méta-apprentissage. De plus, ces auteurs ont établi les méthodes actuelles d'évaluation pour les méthodes FSOD, en particulier sur MSCOCO [4]. Le Faster R-CNN a été réutilisé dans DeFRCN [3] et

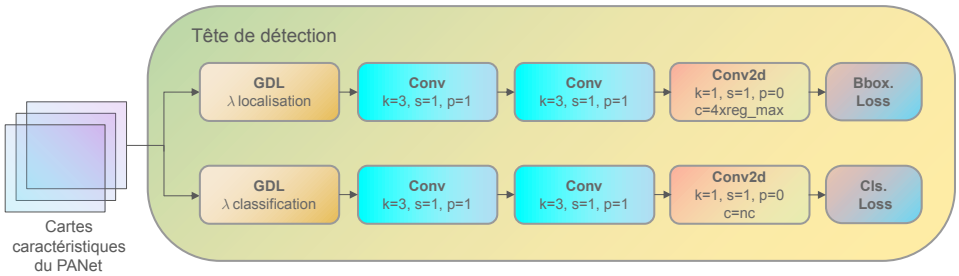

FIGURE 1 – Schéma d'architecture d'une tête de détection de Yolov8 avec les couches GDL.

dans DCFS [1] pour sa capacité à séparer la classification et la localisation. L'apprentissage de ces deux tâches exige que le réseau apprenne des caractéristiques spécifiques aux classes pour la classification, ainsi que d'autres invariantes pour la localisation. En FSOD, ce problème est amplifié en l'absence d'un nombre suffisant d'exemples pour trouver un compromis. Le module GDL de DeFRCN résout ce problème en ajustant le gradient provenant du RPN et de la tête de classification R-CNN. De plus, leur module PCB compare des prototypes pré-calculés aux prédictions du réseau pendant l'inférence pour ajuster les scores de classification.

2.2 Yolov8

Yolov8 est la dernière version des algorithmes YOLO et introduit un changement de paradigme. En effet, Yolov8 est un modèle n'utilisant pas d'ancres pré-calculées pour ses détections, ce qui le rend facilement adaptable à d'autres tâches. Le but de ces modèles sans ancres est d'effectuer des prédictions denses (i.e : pour chaque pixel de leurs espaces latents). D'une manière générale, Yolov8 est inspiré de l'architecture présentée dans [11]. Pour obtenir des représentations à plusieurs niveaux, les auteurs utilisent un encodeur connecté à un "Feature Pyramid Network" [9] et, pour Yolov8, un "Path Aggregation Network" [8]. Puis, Yolov8 utilise pour chacun de ces 3 niveaux de résolution une tête de détection composée de deux branches pour la localisation et la classification. Pour les pertes, la localisation utilise une variante de la Focal Loss et de l'IoU, tandis que la classification utilise une entropie croisée.
Yolov8 donne certes de très bons résultats dans un cadre classique, mais n'est pas adapté au problème du FSOD.

3 Intégration du GDL et PCB dans Yolov8

3.1 Gradient Decoupling Layer

Depuis Yolov1, la prédiction de la classification et de la localisation se fait à partir des mêmes poids, ce qui rend un découplage impossible. Avec son changement d'architecture, Yolov8 utilise des branches distinctes en parallèle. La branche de localisation agit alors comme un détecteur d'objets "générique" qui propose des boîtes englobantes, tandis que la branche de classification sort des "heatmaps" pour chacune des classes. Cette nouvelle architecture rend possible l'intégration du module GDL en ajoutant ces couches de découplage au début de chacune des branches, comme illustré dans la figure 1, la rétropropagation dans Yolov8 devenant alors :

$$\theta_{PANet} \leftarrow \theta_{PANet} - \gamma(\lambda_{loc}\frac{\partial \mathcal{L}_{loc}}{\partial \theta_{PANet}} + \lambda_{cls}\frac{\partial \mathcal{L}_{cls}}{\partial \theta_{PANet}}) \tag{1}$$

Où θ_{PANet} représente les paramètres du backbone et du FPN+PANet de Yolov8, $\mathcal{L}_{loc}$, $\mathcal{L}_{cls}$ et λ_{loc}, λ_{cls} la perte et les valeurs de découplage de gradients sur les branches de localisation et de classification, et γ le learning rate.

3.2 Prototypical Calibration Block

Nous calculons d'abord les prototypes en donnant les K-shots de chaque classe à un extracteur de caractéristiques pré-entraîné afin d'obtenir des vecteurs. Ensuite, le PCB calcule le prototype p_c pour chaque classe en moyennant ces vecteurs, formant la banque de prototypes $P = \{p_c\}$.
Puis dans Yolov8, comme illustré dans la figure 2, nous prenons chaque boîte englobante proposée par la branche de localisation en entrée d'un "RoI Align" afin d'extraire de la feature map de l'image en inférence des représentations de taille fixe f_{bbox}. Enfin, pour chaque boîte englobante, on calcule la similarité cosinus entre sa représentation et les prototypes de chaque classe pour obtenir son score de classification d'après le module PCB :

$$PCB_{bbox} = \frac{f_{bbox} \cdot p_c}{\|f_{bbox}\|\|p_c\|}, p_c \in P \tag{2}$$

Pour finir, nous fusionnons les scores de classification de toutes les boîtes englobantes du PCB et de Yolov8 C_{yolo} avec une somme pondérée par α en profondeur sur les heatmaps pour obtenir le score de classification finale C_{final} :

$$C_{final} = \alpha \cdot C_{yolo} + (1 - \alpha) \cdot PCB \tag{3}$$

4 Expérimentations

Nous avons évalué l'impact de ces modules dans Yolov8 sur MSCOCO adapté au FSOD, comme décrit dans [12]. Sur les 80 classes, 60 sont les classes de base, tandis que les 20 dernières sont les nouvelles. Pour l'entraînement, toutes les images de MSCOCO ont été utilisées, à l'exception de 5000 réservées pour la validation. Les métriques utilisées sont bAP50,bAP,nAP50,nAP qui correspondent au mAP50 et mAP pour les classes de base et les nouvelles classes.
Module GDL. Nous avons tout d'abord effectué plusieurs pré-entraînements avec le modèle Yolov8m en modifiant

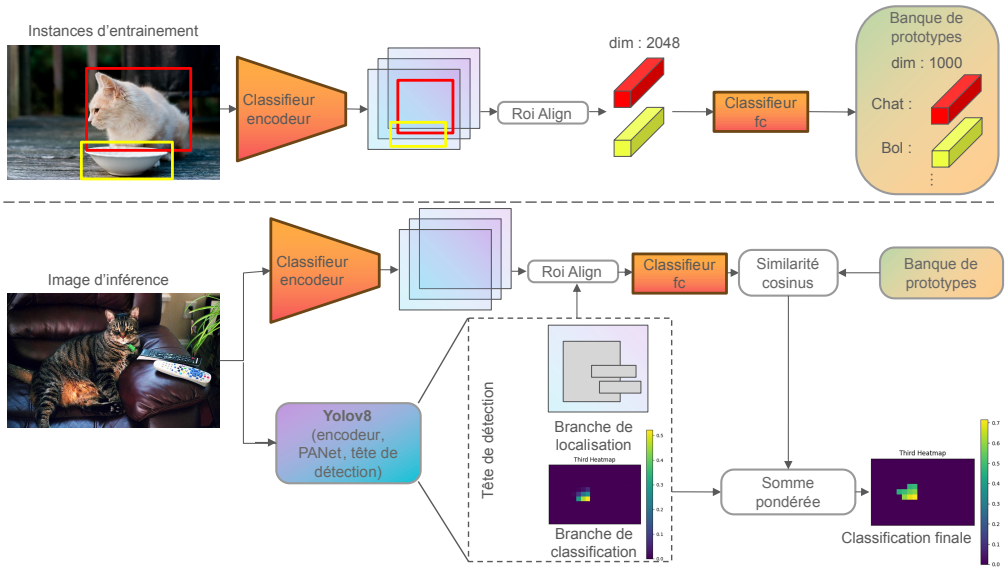

FIGURE 2 – Illustration du module PCB pour une des trois têtes de détection et seulement la heatmap pour la classe *Chat*. "*Classifieur fc*" dénote la dernière couche complètement connectée du classifieur pré-entraîné sur ImageNet.

	1-shot				*2-shot*			
	bAP50	bAP	nAP50	nAP	bAP50	bAP	nAP50	nAP
Yolov8m_vanilla	13.9	9.1	3.9	2.3	12.9	8.9	6.3	3.7
Yolov8m_GDL	54.7	39.0	10.4	6.9	49.0	34.7	12.0	7.4
DeFRCN	48.9	32.0	8.8	5.1	50.4	32.9	16.8	9.6
	3-shot				*5-shot*			
	bAP50	bAP	nAP50	nAP	bAP50	bAP	nAP50	nAP
Yolov8m_vanilla	19.1	13.4	9.3	5.6	19.3	13.6	11.7	7.4
Yolov8m_GDL	56.0	40.7	15.1	10.2	53.2	38.2	21.3	14.0
DeFRCN	50.6	33.1	21.9	12.3	51.5	33.6	26.1	14.2
	10-shot				*30-shot*			
	bAP50	bAP	nAP50	nAP	bAP50	bAP	nAP50	nAP
Yolov8m_vanilla	19.3	13.6	15.5	10.0	23.6	16.9	23.2	15.7
Yolov8m_GDL	50.0	35.1	26.2	16.9	52.1	36.5	31.8	21.1
DeFRCN	53.3	34.6	32.2	17.3	53.6	34.9	38.3	21.4

TABLE 1 – Comparaison entre Yolov8 vanilla, Yolov8 avec module GDL, et DeFRCN avec GDL uniquement.

les valeurs de découplage. Cependant, dans ce cas classique avec beaucoup de données, nous n'avons pas observé d'amélioration comme dans DeFRCN pour le Faster R-CNN, les meilleurs modèles étant ceux rétro-propageant également la localisation. Tous ces pré-entraînements obtiennent cependant de meilleurs scores que DeFRCN étant donné l'architecture plus récente de Yolov8 (+2.1 bAP50 sans localisation, +4.3 bAP50 avec).

Nous avons effectué ensuite la phase d'apprentissage K-shots en testant pour chacun des pré-entraînements de nouvelles valeurs de découplage. Le meilleur résultat que nous ayons obtenu fut en utilisant pour le pré-entraînement $\lambda_{loc} = 0.25$ et $\lambda_{cls} = 0.75$, et pour le finetuning $\lambda_{loc} = 0.1$ et $\lambda_{cls} = 0.1$. Les résultats du Yolov8 résultant, présentés dans le tableau 1, montrent une amélioration par rapport à Yolov8 classique et des scores presque équivalent à DeFRCN en nAP pour le 5,10, et 30-shots, tout en ayant 2 fois moins de paramètres (25M vs 51M).

Nous avons également testé d'autres tailles de Yolov8. Le modèle s donne des performances inférieures (-3.4 nAP50), et le modèle x n'apporte pas d'améliorations notables.

Module PCB. Comme dans DeFRCN, nous avons utilisé le modèle Resnet101 pré-entrainé sur ImageNet comme extracteur de caractéristiques. Nous avons effectué des tests avec plusieurs valeurs de α. De plus, nous appliquons ce module uniquement lorsque Yolov8 prédit une nouvelle classe afin de ne pas potentiellement détériorer les classes de base qui sont déjà bien apprises par le réseau. Enfin, nous appliquons le module uniquement lorsque la confiance de Yolov8 se situe entre une borne inférieure c_{min} et une borne supérieure c_{max}. Le module PCB intégré à Yolov8 nous apporte un léger gain, détaillé dans le tableau 2, en utilisant les paramètres $\alpha = 0.5$, $c_{min} = 0.05$ et $c_{max} = 1$.

5 Discussions

L'utilisation des modules GDL et PCB peut donc apporter un gain pour Yolov8 dans le contexte du FSOD. Cependant,

1-shot		*2-shot*		*3-shot*	
nAP50	nAP	nAP50	nAP	nAP50	nAP
+0.1	+0.1	+0.7	+0.4	+0.8	+0.6
5-shot		*10-shot*		*30-shot*	
nAP50	nAP	nAP50	nAP	nAP50	nAP
+1.7	+1.0	+1.7	+1.0	+1.6	+1.0

TABLE 2 – Gain du module PCB sur les nouvelles classes.

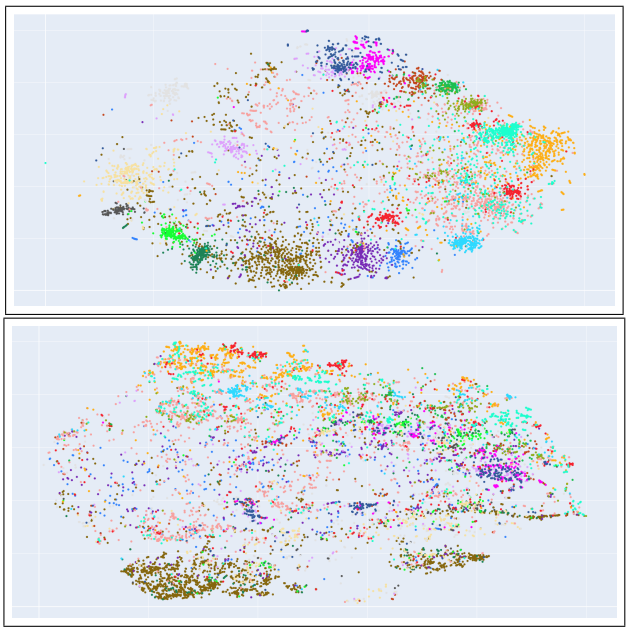

FIGURE 3 – Visualisation t-SNE des vecteurs des 20 nouvelles classes (1 couleur = 1 classe) obtenus par *a*) Resnet101 entraîné sur ImageNet (module PCB) *b*) Yolov8 entraîné sur les 60 classes de base seulement.

nous voulons remettre en perspective nos résultats en analysant certains biais que nous pensons qu'il existe avec cette méthode (et d'autres dans la littérature).

5.1 Utilisation d'un extracteur de caractéristiques pré-entrainé

Le module PCB, comme beaucoup de méthodes utilisant des prototypes pré-calculés, utilise un extracteur de caractéristiques pré-entraîné sur ImageNet. Nous aimerions souligner ici que l'utilisation de ces méthodes semble biaisée dans le cas de l'apprentissage few-shot. Bien qu'aucune des classes de ImageNet n'ait exactement le même nom que celles des nouvelles classes de MSCOCO, on peut trouver des objets très similaires sémantiquement. Par exemple pour les animaux, les classes *Cat*, *Dog*, sont de nouvelles classes à apprendre, or certaines classes présentent dans ImageNet partagent beaucoup de traits avec celles-ci (*Siamese Cat*, *Persian Cat*, *Shetland Sheepdog* ...). Dans la figure 3a, on peut constater que les nouvelles classes sont "compactes" alors qu'elles sont censées n'avoir jamais été vues, tandis que dans la figure 3b, on voit des classes "éparpillées". Cela tendrait à montrer que le module PCB a déjà "appris" les nouvelles classes.

5.2 Suivi des performances sur la validation

Pendant l'apprentissage des nouvelles classes sur MSCOCO, un suivi des performances du détecteur est calculé sur l'ensemble de validation à chaque époque afin de pouvoir obtenir les meilleurs poids. Cela nécessite donc beaucoup d'exemples de ces nouvelles classes dans l'ensemble de validation ce qui n'est pas vraiment crédible dans un cas d'application réel du FSOD où les seuls exemples sont ceux de l'entraînement. Pour essayer de quantifier cet impact, nous avons comparé pour un même Yolov8 les différences avec et sans suivi (l'entraînement s'arrêtant au bout de 180 époques). Sans ce suivi peu réaliste, nous obtenons une différence notable de -9.7 bAP50 et -2.4 nAP50.

Remerciements

Nous remercions l'Association Nationale de la Recherche et de la Technologie ainsi que Drone Geofencing pour le financement de la thèse CIFRE.

Références

[1] Bin-Bin Gao et al. Decoupling classifier for boosting few-shot object detection and instance segmentation. *NeurIPS*, 35 :18640–18652, 2022.

[2] Glenn Jocher et al. YOLO by Ultralytics, January 2023. https ://github.com/ultralytics/ultralytics.

[3] Limeng Qiao et al. Defrcn : Decoupled faster r-cnn for few-shot object detection. *ICCV*, pages 8681–8690, 2021.

[4] Michael Maire et al. Microsoft coco : Common objects in context. pages 740–755. Springer International Publishing, 2014.

[5] Mona Köhler et al. Few-shot object detection : a comprehensive survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2023.

[6] Qi Fan et al. Few-shot object detection with attention-rpn and multi-relation detector. *ICCV*, pages 4013–4022, 2020.

[7] Shaoqing Ren et al. Faster R-CNN : Towards real-time object detection with Region Proposal Networks. *NeurIPS*, Vol. 28, 2015.

[8] Shu Liu et al. Path aggregation network for instance segmentation. In *CVPR*, June 2018.

[9] Tsung-Yi Lin et al. Feature pyramid networks for object detection. *CVPR*, pages 2117–2125, 2017.

[10] Xiongwei Wu et al. Meta-rcnn : Meta learning for few-shot object detection. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 1679–1687, 2020.

[11] Zhi Tian et al. Fcos : Fully convolutional one-stage object detection. *ICCV*, pages 9627–9636, 2019.

[12] Xin Wang, Thomas E. Huang, Trevor Darrell, Joseph E Gonzalez, and Fisher Yu. Frustratingly simple few-shot object detection. *ICML*, 119 :9919–9928, July 2020.