

Projet 07: Développez une preuve de concept

L'objectif de ce projet est d'améliorer la solution déployée au projet précédent, limitée par le fait qu'elle ne peut pas "reconnaître" différentes races de chien sur une même image. Nous allons donc considérer une nouvelle fois le jeu de données *Stanford dogs dataset*. Pour améliorer la solution dans ce sens, nous allons utiliser les modèles Deep Learning de détection d'objets basés sur l'architecture *YOLO*, connu pour sa précision et sa grande rapidité. Les modèles de la famille *YOLO* prennent une image (ou vidéo) en entrée et produisent en sortie les *boîtes d'encrage* pour la localisation (délimitation par des rectangles d'encadrement) ainsi que la probabilité pour la classification des objets détectés sur l'image (ou vidéo). Un exemple d'inférence est montré sur la figure 1.

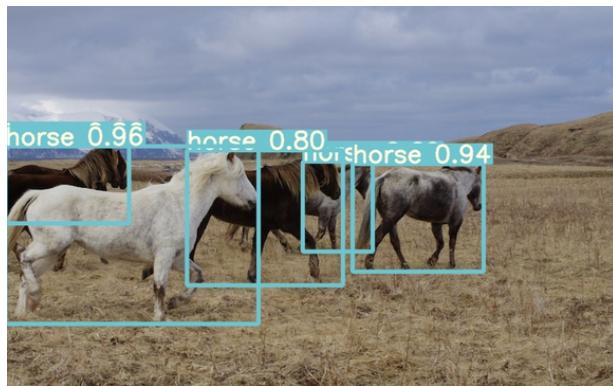


Figure 1: Détection des différents chevaux sur une même image.

Contents

1	Introduction	2
2	État de l'art	3
2.1	La famille des modèles à deux passes	3
2.2	La famille YOLO	4
3	Transfert d'apprentissage	5
3.1	Configuration	6
3.2	Comparaison des performances	6
4	Conclusion	8
5	References	10

1 Introduction

Dans le contexte de la vision par ordinateur, la détection d'un ou plusieurs objets dans une image est, de nos jours, particulièrement bien traitée. Cette dernière consiste à reconnaître des objets dans un environnement et à les classer par catégorie. La localisation se visualise grâce à un rectangle d'ancrage encadrant la cible. La détection d'objet sur une image regroupe donc les tâches de classification ainsi que la localisation de cet objet. La segmentation sémantique des images consistant en la classification de chaque pixel en fonction de la classe de l'objet auquel il appartient est la troisième classe de problématiques en vision par ordinateur. Sur la figure 2(a), on peut voir les différents types de résultats auquel on s'attend selon la problématique d'intérêt.

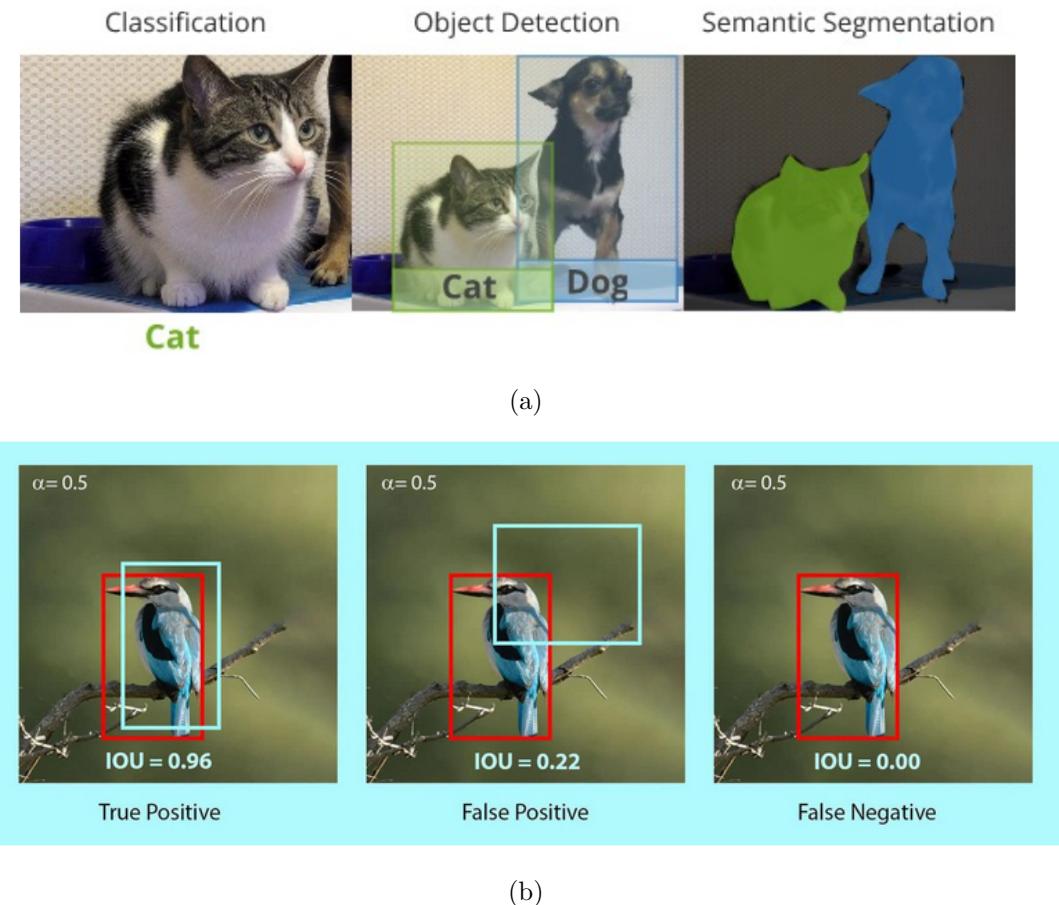


Figure 2: (a) Les trois principales classes de problématiques en vision par ordinateur. (b) Métrique pour la détermination de la boîte d'encrage basée sur le seuil IoU (ici $\alpha = 0.5$).

Localiser un objet dans une image peut s'exprimer sous la forme d'une tâche de régression : pour prédire le rectangle encadrant l'objet, une approche classique consiste à prédire les coordonnées du centre du rectangle ainsi que sa hauteur et sa largeur. Pour mesurer les performances de la localisation, on utilise la métrique mAP (*mean Average-Precision*, ou la moyenne des précisions moyennes) basé sur un seuil dit "*IoU*" (*Intersection over Union*) : seuil basé sur le rapport entre l'intersection de la zone prédite et réelle avec l'union de ces deux zones (voir la figure 2(b)):

- La prédiction est un "Vrai-Positif" : lorsque $IoU \geq$ seuil

- La prédiction est un "Faux-Positif" : lorsque $\text{IoU} \leq \text{seuil}$
- La prédiction est un "Faux-Négatif" : lorsque aucun rectangle n'est prédict alors que l'image en contient

Il est à noter qu'en détection d'objets on ne considère pas les "Vrai-Négatif" car ils correspondent à toutes les parties de l'image n'ayant pas de boîtes d'encrages. Un IoU de 1 signifie que les coordonnées des boîtes prédictes et réelles sont identiques, et un IoU de 0 signifie qu'il n'existe aucun chevauchement entre elles. Pour chaque classe $i \in [1, 120]$, la *Average-Precision* correspond à l'aire sous la courbe *ROC*. La *mAP* est alors donnée par la moyenne des selon toutes les classes du problème :

$$mAP = \frac{1}{N} \sum_{i=1}^{N=120} AP_i \quad (1)$$

YOLO a été proposé par [Redmon et al., 2016] pour traiter les problèmes de rencontrés par les modèles existant de reconnaissance d'objets. Une des principales limitations des modèles de pointe (précédant YOLO) était qu'ils n'était pas assez rapide pour être utilisé en temps réel. Dans un premier temps, nous allons présenter l'état de l'art des approches pour la détection d'objets en Deep Learning. Nous allons décrire succinctement les évolutions qui ont contribué à l'amélioration progressive de la famille des modèles à travers les différentes versions. Ensuite, nous allons présenter les résultats du *Transfer Learning* où l'on a utilisé les deux dernières versions : **YOLOv5** et **YOLOv7** publiés en mai 2020 et juillet 2022, respectivement. Quelques pistes pour améliorer les performances du transfert d'apprentissage seront discutés en conclusion.

2 État de l'art

Pour détecter plusieurs objets dans une même image, les premières approches procédaient en deux étapes : (1) en extrayant des zones d'intérêts sur toute l'image et (2) en appliquant ensuite l'algorithme de classification. Cette approche était lente et imposait de relire plusieurs fois l'image source. L'idée fondamentale derrière **YOLO** est de ne faire qu'une seule passe (ou lecture) de l'image (**YOLO** = *You Only Look Once*) et de prédire simultanément l'encadrement et la classe de l'objet. Il se positionne dès le départ sur une plus grande rapidité d'exécution que les modèles en deux passes, tout en conservant une précision proche de ceux-ci.

2.1 La famille des modèles à deux passes

Tout a commencé avec l'article [Girshick et al., 2014] qui proposa le modèle appelé *R-CNN* prenant en entrée des "régions d'intérêts" (extraits via l'algorithme *Selective Search* de regroupement de pixels pour une tâche dite de *region proposals*) et un réseau de neurones convolutifs (CNN) standard pour les classer. Il existe également l'autre famille d'algorithmes de *region proposals* basés sur le glissement d'une "fenêtre" sur l'image en input. Ce premier modèle a rapidement évolué vers *Fast R-CNN* ([Girshick, 2015]) où une technique appelée *Region of Interest Pooling* permettait de réduire les calculs coûteux de l'algorithme *Selective Search* et rendait le modèle beaucoup plus rapide. Enfin est venu *Faster R-CNN* ([Ren et al., 2015]), l'une des méthodes de détection d'objets les plus populaires avant l'apparition des architectures **YOLO**. Le point principal de ce dernier est de faire du réseau de propositions des régions une partie intégrante du réseau neuronal, s'affranchissant ainsi de l'algorithme de segmentation *Selective Search*. Les modèles

en deux étapes requièrent plus de temps à l’inférence mais ont permis pendant un moment d’obtenir de meilleures prédictions que les modèles à une étape (dont **YOLO** fait partie). Voici en résumé les principales étapes de cet algorithme :

1. L’image en entrée est passée dans un CNN (*backbone* ou *base features extractor*) pour obtenir une carte des caractéristiques des objets présents sur l’image.
2. En utilisant des cadres, on génère des propositions de régions sur cette carte caractéristiques (cadres de délimitation qui contiennent les objets pertinents de l’image).
3. La carte caractéristique extraite par le CNN en (1) et les cadres de délimitation des objets pertinents obtenus grâce à la mise en commun des régions d’intérêt déterminées dans l’étape (2) sont utilisés pour générer une nouvelle carte caractéristique.
4. Les régions regroupées passent ensuite par des couches entièrement connectées pour la prédiction des coordonnées des zones des objets et les classes de sortie.

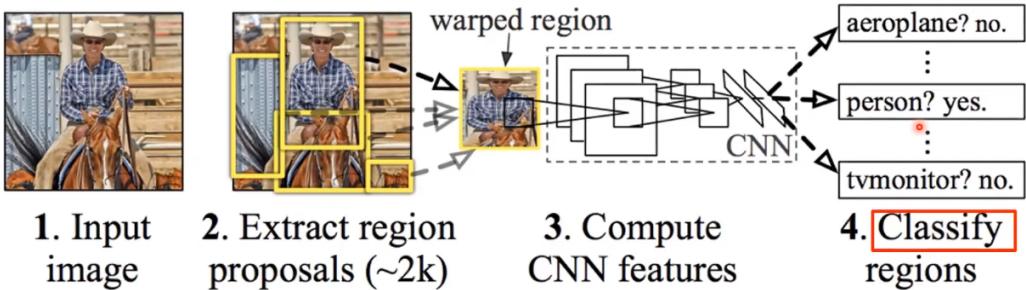
Bien que ces approches aient résolu les problèmes de limitation des données et de modélisation dans la détection d’objets, elles ne sont pas en adaptée pour la détection en temps réel. Sur la figure 3(a) sont illustrées les étapes précédemment décrites.

2.2 La famille YOLO

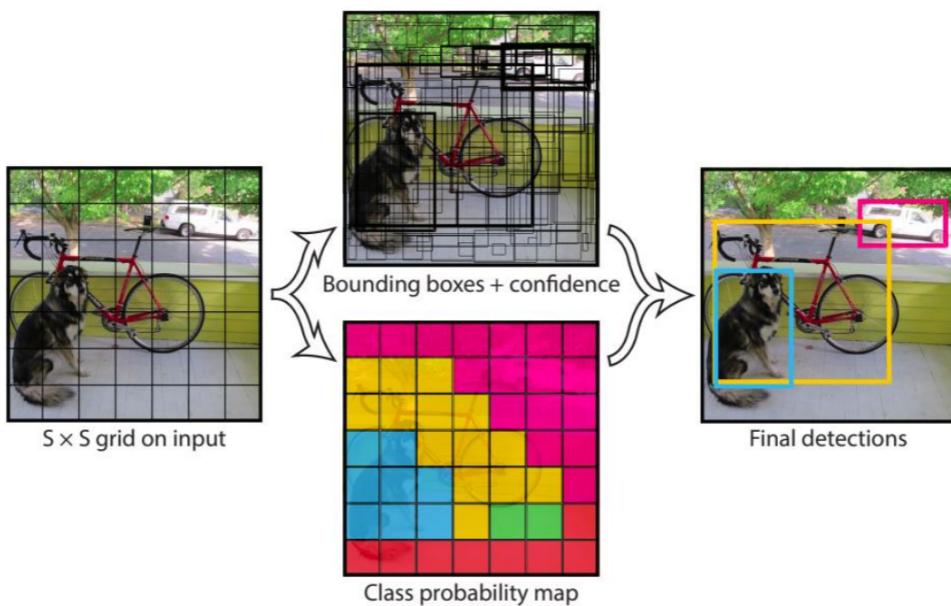
L’algorithme **YOLO**, dont les trois versions originelles ont été développées par la même équipe de chercheurs, a donc gagné en popularité en raison de ses performances supérieures par rapport aux techniques de détection d’objets susmentionnées. Contrairement à ce que l’on pourrait penser au vu de ses performances, les récentes versions n’ont pas une architecture complètement nouvelle ou révolutionnaire par rapport aux méthodes de l’état de l’art. Il faut plutôt les voir comme concervant le concept de ses prédecesseurs optimisé par un ”sac” (ou *bag*) de nouvelles technologies : en ré-implémentant un nombre important d’avancées présents dans la recherche scientifique de ces dernières années permettant de gagner en performances sans impacter la rapidité à l’inférence.

Le modèle fonctionne en divisant d’abord l’image d’entrée en une grille ($S \times S$) de cellules, où chaque cellule de la grille est chargée de prévoir les cadrages et leurs scores de confiance. Si le centre de la boîte englobante de l’objet se trouve dans une grille, alors cette grille est responsable de la détection de cet objet. Sur l’illustration de la figure 3(b), une image est divisée en une grille de 7×7 . La carte des probabilités de classe (par cellule) et les cadres de délimitation (plusieurs cadres par cellules) avec confiance sont ensuite combinés (l’IoU est ensuite calculé pour regrouper les cadres qui détectent le même objet) en un ensemble final de cadres de délimitation et d’étiquettes de classe.

La cinquième version de **YOLO** n’a pas été accompagné par une publication, et on trouvera dans [Wang et al., 2022] les améliorations apportés dans la septième version. Les principales améliorations entre les versions successives concernent principalement l’architecture du réseau ”*backbone*”, une plus grande résolution numérique de l’image en entrée ou encore la méthode de calcul des différentes fonctions perdes.



(a) Extrait de [Girshick et al., 2014].



(b) Extrait de [Redmon et al., 2016]

Figure 3: (a) Les quatre principales étapes de classification avec la famille des modèles R-CNN. (b) Principe de détection simultanée de la famille des modèles **YOLO**.

3 Transfert d'apprentissage

Le principe du transfert d'apprentissage (*transfer learning*) est non seulement d'accélérer considérablement l'entraînement, mais aussi de permettre d'obtenir de bonnes performances avec des jeux de données d'entraînement relativement réduit - ici nous allons utiliser le dataset au complet avec l'ensemble des 120 races. Les couches de sorties des modèles d'origines ont été remplacées car le nombre de sorties approprié doit être adapté à notre tâche de classification (ici 120). De manière comparable, les couches cachées supérieures peuvent s'avérer être moins utiles que les couches inférieures, car les caractéristiques de haut-niveau intéressantes pour notre tâche sont différentes des cibles utilisées lors du pré-entraînement. Pour simplifier la tâche dans ce projet, nous allons conserver toutes les couches cachées et remplacer uniquement la couche de sortie.

3.1 Configuration

L’arborescence des répertoires contenant les jeux d’entraînement et jeux de validations est montrée sur la figure 4(a) : les fichiers images et annotations sont séparés dans des sous-dossiers pour chacun des répertoires **train** et **val**. À chaque image est associé un fichier texte d’annotations contenant les informations sur le rectangle d’encadrement ainsi que la race. Un rectangle est annoté par une série de 5 chiffres (voir la figure 4(a)):

- numéro de la classe
- position selon Ox du centre de la boîte
- position selon Oy du centre de la boîte
- largeur de la boîte
- hauteur de la boîte

Si une image contient plusieurs chiens, on a alors une ligne pour chaque boîte dans le fichier annotation correspondant.

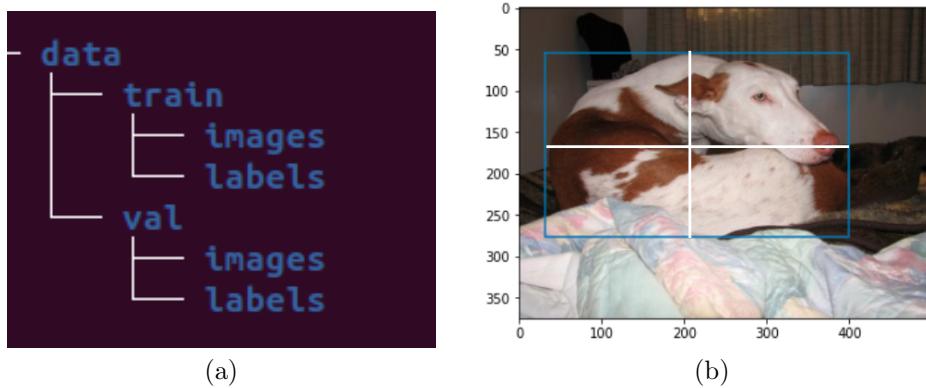


Figure 4: (a) Arborescence des répertoires. (b) Annotations pour un rectangle d’encadrement.

3.2 Comparaison des performances

Pour ce projet, le modèle a été entraîné sur l’ensemble du dataset. Les poids pré-entraînés pour la version **5** et **7** doivent être téléchargés via github aux adresses <https://github.com/ultralytics/yolov5.pt> et https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7_training.pt, respectivement.

Les matrices de confusions ainsi que les courbes ROC obtenues à la fin de l’entraînement des deux versions sont montrées sur la figure 5. À la fin de l’entraînement, on arrive globalement à une précision moyenne (basé sur un seuil IoU à 0.5) autour de 0.8 selon les deux modèles. Les résultats obtenus avec la version **7** étant légèrement meilleures que ceux obtenus avec la version **5**. L’entraînement n’a pas spécialement été chronophage : les résultats satisfaisant montrés dans cette section ont été obtenus avec 70 epochs pour 6h30 environ de calcul sur la machine GPU P100 de la plateforme Kaggle.

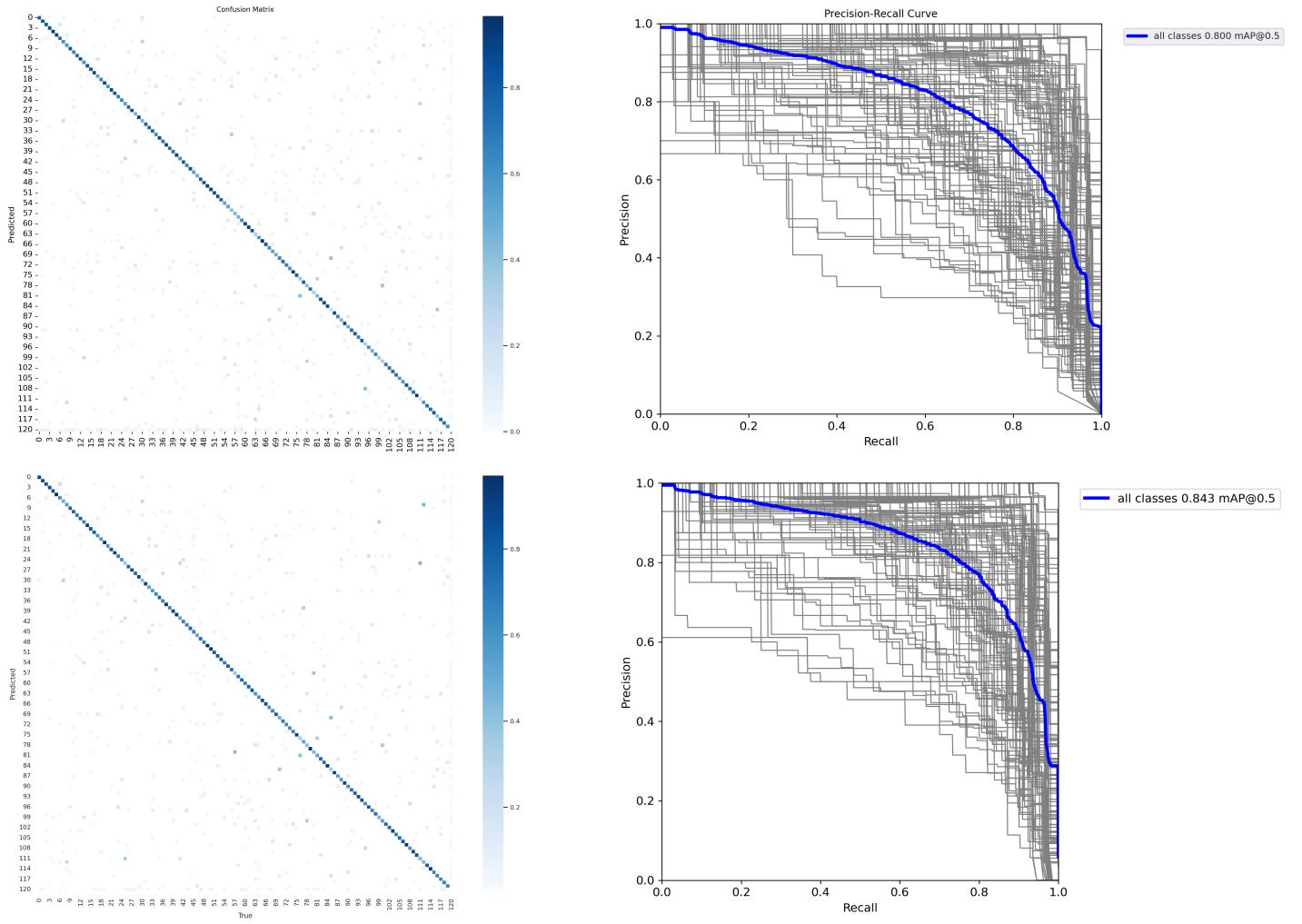


Figure 5: Matrice de confusion (à gauche) et courbe *Precision-Recall* (à droite) obtenus avec la version 5 (Haut) et la version 7 (Bas) sur l’ensemble des 120 races de chien du dataset.

Dû au découpage de l’image input, il est reconnu que **YOLO** a des difficultés pour reconnaître de petits objets sur une image. Dans notre cas, nous ne sommes pas confronté à ce problème car la dimension du chien sur une image sera toujours supérieures à la dimension d’une cellules de la grille. On peut voir sur la figure 6 un résultat de l’inférence avec les nouveaux poids ré-entraîné du modèle **YOLOv7**. Lorsque les chiens (ou les ”objets” de manière plus générale) sont proches les uns des autres l’algorithme de localisation peut avoir des difficultés à prédire correctement la position des rectangles d’encadrements sur l’image.

Sur la figure 7 est montré l’évolution des fonctions de perte lors de la phase d’apprentissage. Il y a trois types de fonctions pertes que les modèles **YOLO** tente d’optimiser pour améliorer les performances :

- **Box loss** : fonction perte *Mean Squared Error* pour la prédiction des coordonnées du rectangle d’encadrement.
- **Objecness loss** : mesure la probabilité qu’un objet soit bien présent (optimisation via la fonction *Binary Cross Entropy*).
- **Classification loss** : fonction perte *Cross Entropy* usuelle pour la classification multi-classe.

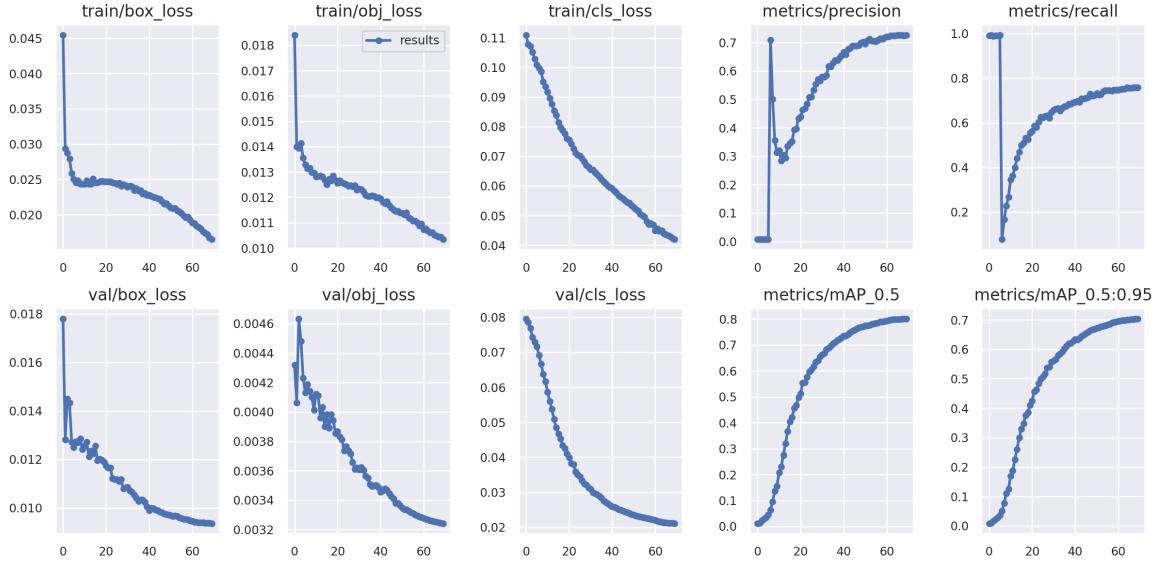


Figure 6: Inférence sur deux nouvelles images en utilisant le modèle **YOLOv7** ré-entraîné.

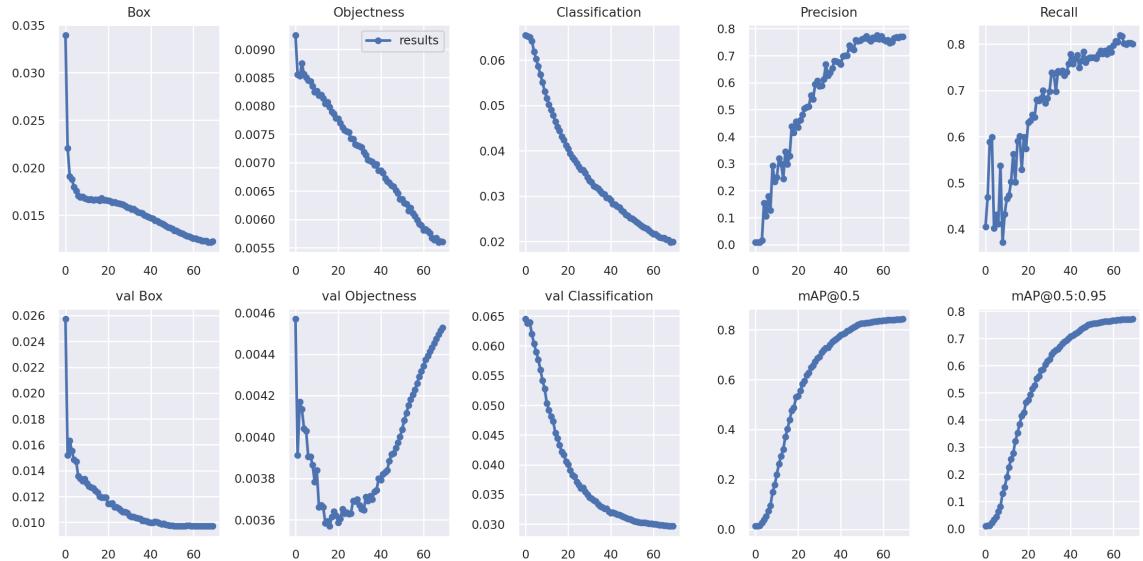
Les améliorations apportées dans la septième version permettent d’obtenir une meilleure optimisation des fonctions pertes lors des phases d’entraînement et de validation. Les performances sur la classification ainsi que la localisation sont également meilleures avec la septième version. Comme on peut le voir, la phase d’apprentissage est arrêtée au début de la saturation de la courbe d’apprentissage pouvant entraîné du sur-apprentissage.

4 Conclusion

Les résultats obtenus ici sont très satisfaisant au vu de la simplification apportée à l’approche de transfert d’apprentissage. Pour améliorer les performances, nous pouvons retirer certaines des couches supérieures des modèles d’origine et figer de nouveau les couches cachées restantes, et répéter la procédure jusqu’à trouver le bon nombre de couches à réutiliser.



(a) Évolution des fonctions pertes avec la version 5.



(b) Évolution des fonctions pertes avec la version 7.

Figure 7: Résultat des fonctions pertes et des métriques lors de la phase d'entraînement et de validation du modèle **YOLOv5** (a) et **YOLOv7** (b) sur l'ensemble des 120 races de chien du dataset.

5 References

- [Girshick, 2015] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.
- [Girshick et al., 2014] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.
- [Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.
- [Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.
- [Wang et al., 2022] Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*.