

Détection d'objet appliquée au jeu "Où est Charlie?"

AUTEURS

Klein Timothée
Ragot Cyrian
Arrix-Pouget Baptiste
Rosseel Yannis

2025-05-25

Table des matières

1	Introduction	1
2	Constitution de la base de données	1
2.1	Acquisition et annotation des données	1
2.2	Partitionnement des données	3
2.3	Pronostics	3
2.4	Chargement des données	4
3	Modèle et entraînement	5
3.1	Description du modèle et hyperparamètres	5
3.2	Différentes approches	5
3.2.1	Entraînement sur les images complètes	5
3.2.2	Entraînement sur les imagettes	5
3.2.3	Entraînement sur les imagettes avec ré-annotations	7
4	Analyse détaillée des résultats	11
4.1	Analyse quantitative	11
4.1.1	Analyse de la loss	12
4.1.2	Analyse du rappel	12
4.2	Analyse qualitative	13
5	Conclusion	18
5.1	Bilan	18
5.2	Difficultés rencontrées	18
5.3	Perspectives et améliorations	18
Liste des Figures		I
Références		II

1 Introduction

Dans le cadre du projet d'apprentissage profond, nous avons choisi le sujet de détection d'objets dans les livres "Où est Charlie ?". L'objectif est de pouvoir détecter dans une image les personnages Charlie, Wenda, Odlaw et Wizard qui sont représentés sur la figure 1.

Dans tout le projet, nous utilisons le modèle Yolo implémenté par ultralytics, toutes références au modèle Yolo fait donc référence à cette implémentation. [1]



FIGURE 1 – Les personnages des livres

2 Constitution de la base de données

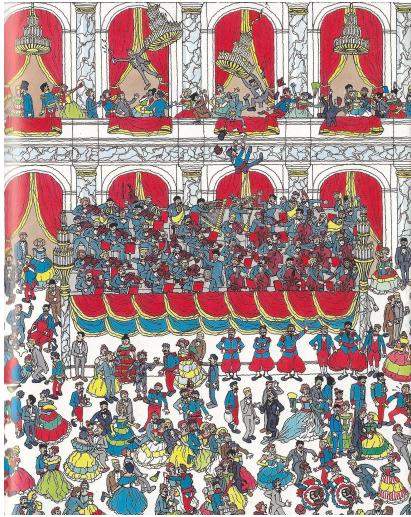
2.1 Acquisition et annotation des données

Pour construire notre base de données, nous avons dans un premier temps cherché sur le web différentes images des livres. [2] [3] [4] [5] [6] Cependant, beaucoup d'entre elles étaient incomplètes (recadrées) et nous en avons trouvé peu. Finalement, nous avons pu nous procurer un livre de poche contenant les six premières éditions de la série, que nous avons scannées et recadrées pour compléter notre base de données. [7] Les images que nous avons pu rassembler se trouvent dans le projet GitHub lié à ce papier. [8] Quelques exemples de la base de données sont donnés en figure 2.

Certaines images telles que celle de la figure 3 n'ont pas été gardées, car elles contiennent beaucoup de faux Charlies, Odlaw, Wenda ou Wizard et donc pourraient fortement tromper notre modèle.

Dans un second temps, nous avons annoté les images à l'aide de l'outil collaboratif et open-source Label Studio que nous avons hébergé sur un VPS. [9] Cet outil a été très pratique, car très intuitif et permettant de collaborer en temps réel sur une même base de données et d'exporter les annotations sous différents formats. Puisque nous travaillons avec le modèle YOLO, nous avons exporté les annotations en format YOLO directement.[1]

Aussi, on peut noter que nous avons choisi de ne pas classer le personnage Woof, le chien de Charlie, car il est souvent très caché et on ne voit que sa queue dépasser.



(a) Scannée, 2nd page d'une illustration entière



(b) Trouvée en ligne, non entière

FIGURE 2 – Images de la base de données.

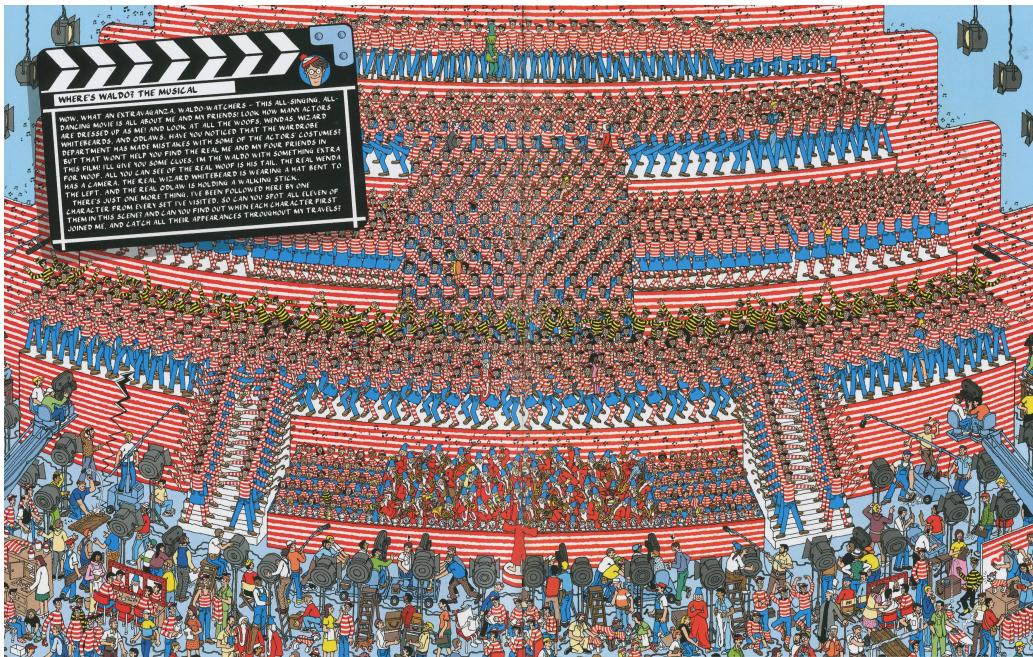


FIGURE 3 – Image retirée.

2.2 Partitionnement des données

Le partitionnement des données a quelques aspects spécifiques, car nos données proviennent de plusieurs sources. En effet, certains fichiers contiennent une illustration entière (c'est-à-dire une double-page) des livres "Où est Charlie?", tandis que d'autres fichiers contiennent seulement une page ou une version recadrée de l'illustration (ce sont les images trouvées sur internet uniquement). Voir figure 2. Ceci est important à noter, car sur une illustration entière, on retrouve toujours les quatre objets à classer, mais sur des images recadrées ou sur une seule page, il n'y a jamais toutes les classes présentes dans les fichiers. Aussi, pour éviter d'inclure un biais dans nos données, nous sommes contraints de séparer les données par fichiers. Il ne faudrait pas qu'une information soit redondante dans les jeux de données.

Ainsi, pour séparer équitablement les images dans un dataset comprenant des données d'entraînement, de validation et de test, nous avons écrit un script `split_dataset.py`. [8] Ce script réparti les données avec une répartition 80% / 10% / 10% respectivement dans les trois jeux d'entraînement, de validation et de test en veillant à ajouter des proportions égales d'images avec une illustration entière et d'images recadrées incomplètes. Les paramètres du script peuvent être modifiés dans le fichier.

Ce script nous permet aussi de renommer les noms fichiers images et des fichiers labels avec des noms normalisés. Le format des noms est comme suit : <numéro livre>-<numéro page>-<type> où le type vaut 0 lorsque l'image est une illustration entière, vaut 1 ou 2 si c'est la première ou deuxième page d'une illustration, et il vaut 3 si c'est une version incomplète recadrée d'une illustration.

Le script range le dataset selon la convention demandée par Yolo [1], voir figure 4.

```
dataset/
|-- test/
|   |-- images/
|   '-- labels/
|-- train/
|   |-- images/
|   '-- labels/
 '-- valid/
     |-- images/
     '-- labels/
```

FIGURE 4 – Structure du dossier pour l'entraînement du modèle YOLO

2.3 Pronostics

Bien que le projet soit amusant et intéressant, notre problème reste très compliqué pour plusieurs raisons. En effet, notre base de données est très petite pour obtenir de bons résultats (100 images dont

70 sont des illustrations complètes). Nous aurons du mal à obtenir un modèle qui ne sous-apprend pas tout en ayant de bons résultats.

De plus, puisque les images proviennent de plusieurs sources, nous disposons d'images ayant des résolutions différentes, ceci pourra influencer nos résultats.

Aussi, les illustrations sont faites pour tromper l'œil humain, elles peuvent facilement tromper notre modèle également : en effet, dans les illustrations, il y a souvent des personnages qui ressemblent à Charlie avec des couleurs similaires ou des vêtements rayés. De plus, les 4 personnages sont souvent très bien cachés derrière d'autres objets et on ne peut voir qu'une partie de leurs corps, la plupart du temps on ne voit que leurs têtes et elles sont même parfois partiellement cachées, en exemple voir figure 5. Enfin, nous pouvons noter que les personnages se ressemblent : Wenda et Charlie ont des rayures rouges et blanches, des lunettes et un pantalon bleu, et Odlaw porte aussi des lunettes et des vêtements à rayures.



(a) Charlie avec tête cachée



(b) Odlaw sans corps

FIGURE 5 – Exemples de personnages cachés.

2.4 Chargement des données

Le dossier dataset du projet GitHub, qui a été construit avec le script de partitionnement des données, peut être importé directement dans Google Colab, ainsi qu'un fichier YAML `data.yaml` qui spécifie les noms des classes, le nombre de classes et l'emplacement des données d'entraînement, de validation et de test. Ce fichier YAML est écrit selon les conventions de Yolo. [1] [8]

3 Modèle et entraînement

3.1 Description du modèle et hyperparamètres

Notre premier choix a été YOLOv11 comme modèle à entraîner pour notre problème. C'est un très bon modèle pour la détection d'objets et nous n'avons pas eu besoin de le changer au cours du projet. En particulier, nous avons utilisé la bibliothèque ultralytics [1].

Quant aux hyperparamètres que nous avons utilisé, nous avons utilisé ceux par défaut, car ils convenaient déjà à notre problème :

- Pas de flip upside-down, car nos personnages ont toujours la tête en haut,
- Un flip left-right avec une probabilité 0,5, car il y a quelques images où les personnages sont dans l'autre sens, cela permet au modèle de savoir les reconnaître,
- Pas de rotation, car nos personnages ne sont jamais penchés,
- Pour le reste des paramètres, nous n'avons pas considéré que nous devions les changer car ce sont des paramètres de couleur ou qui changent le placement, ce qui nous est utile.

Nous avons à chaque fois entraîné notre modèle avec 50 epochs.

3.2 Différentes approches

L'entraînement du modèle peut être réalisé dans Google Colab en important le fichier `Charlie.ipynb` présent dans notre projet. [8] Avec ce fichier, il est possible de reproduire les différents entraînements que nous allons présenter dans la suite.

3.2.1 Entrainement sur les images complètes

Naïvement, dans un premier temps, nous avons entraîné notre modèle directement sur les images complètes. Pour ce premier entraînement, nos annotations avaient été faites selon la règle suivante : chaque boîte englobante comprend la plus grande zone de chaque personnage visible. Nous verrons plus tard que nous avons choisi de modifier notre manière d'annoter lors d'un autre entraînement. Cet entraînement a été fait en fixant le paramètre `imgsz` de YOLO à 300 pixels dans un premier temps, et nous avons observé des résultats extrêmement médiocres. Puisque nos images sont grandes et avec de très petits détails, dans un deuxième temps, nous avons tenté de fixer la taille des images à 1000 pixels. Cette fois, les résultats se sont légèrement améliorés, mais étaient toujours loin d'être satisfaisants. Nous avons donc changé notre manière de faire en utilisant des imagettes à la place d'images entières.

3.2.2 Entrainement sur les imagettes

Comme les personnages sont très petits à l'échelle d'une image, l'entraînement sur les images entières n'était pas satisfaisant, nous avons donc décidé d'entraîner le modèle sur des imagettes de taille 300*300 autour des personnages.

Notre script `imagettes.py` récupère les images entières et leurs labels et crée une imagette pour chaque personnage de chaque image. L'algorithme fonctionne comme suit : l'imagette est centrée sur la boîte englobante du personnage labélisé, mais si le personnage est proche d'un bord de l'image entière alors l'imagette est décalée pour ne pas sortir de l'image entière, et si l'imagette d'un personnage coupe la boîte englobante d'un autre personnage, l'imagette est décalée de sorte à englober ce second personnage (comportement par défaut). La figure 6 montre ces différents cas de figures. Une aide pour utiliser ce script est fourni dans le README du projet. [8]

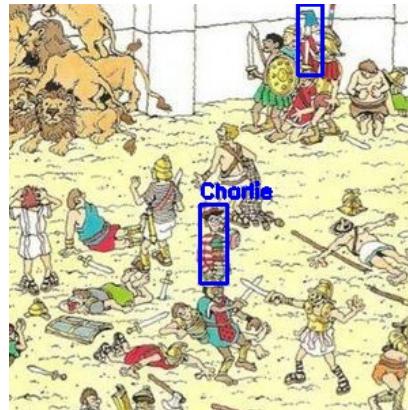
Avec cet algorithme, les personnages sont très souvent positionnés au centre de l'imagette mais cela ce pose pas de réel problème car l'augmentation de données interne à YOLO fait des collages des images qui lui sont fournis.



(a) imagette classique



(b) imagette décalée à cause d'un bord



(c) imagette décalée à cause d'un personnage

FIGURE 6 – Exemples d'imagettes

Après avoir entraîné notre modèle sur les imagettes (dont les résultats étaient plus satisfaisants), il a fallu faire les prédictions sur l'image entière.

Ainsi notre script `full_image_predict.py` découpe les images fournies dans un dossier en ima-

gettes (de la taille de celles de notre entraînement) de sorte que chaque imagette recouvre de moitié, ou plus si des bords sont atteints, ses voisines horizontales et verticales. Puis la prédiction sur chaque imagette est effectuée. Et enfin les résultats sont regroupés en ne gardant que les boîtes englobantes dépassant un certain seuil de confiance (donné en entrée) et si deux boîtes englobantes se coupent, il garde celle avec la plus haute confiance afin qu'un personnage ne soit pas détecté plusieurs fois à cause des superpositions (cela ne devrait pas éliminer de bonnes boîtes englobantes car en général les personnages sont éloignés les uns des autres).

L'affichage est fait sur l'image complète avec des boîtes de couleur avec `visualize_annotations.py` qui a été fait sur la base du projet [10].

Une aide pour utiliser ces scripts est fourni dans le README du projet. [8]

3.2.3 Entraînement sur les imagettes avec ré-annotations

Dans cette partie nous présentons pourquoi et comment nous avons ré-annoté les personnages afin d'obtenir de meilleurs résultats.

Le script `add_annotation.py` permet d'ajouter des annotations à celles déjà existantes, on choisit un numéro entre 0 et 3 pour choisir la classe (Charlie-0, Odlaw-1, Wenda-2, Wizard-3), et on peut dessiner un rectangle sur l'image en cliquant puis en relâchant appuyé jusqu'à mettre l'autre coin du rectangle. On passe à l'image suivante avec 's' et on stoppe avec 'q'.

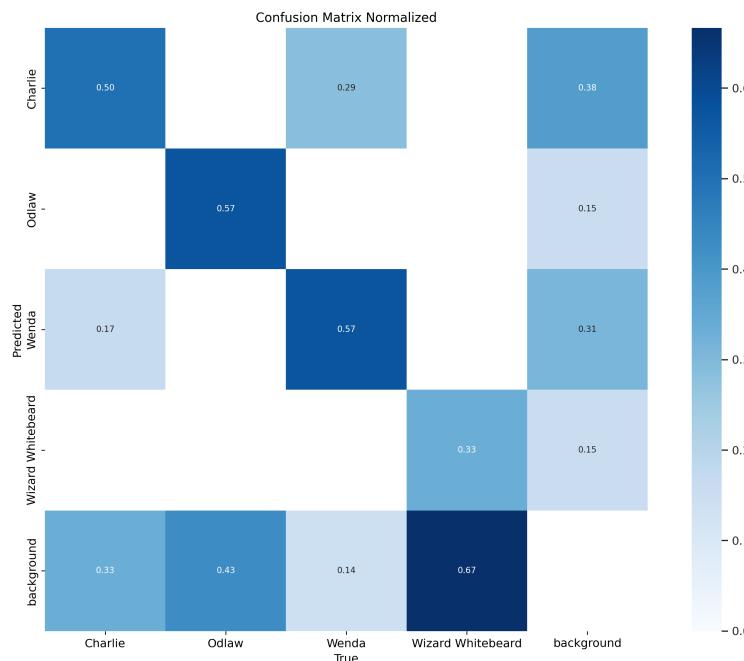


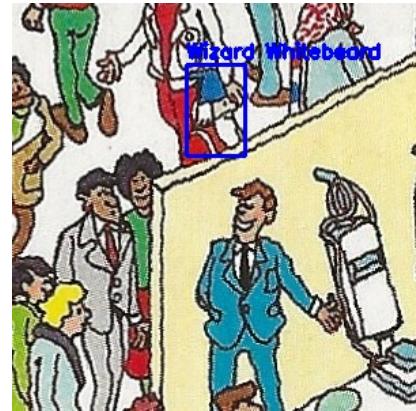
FIGURE 7 – Matrice de confusion avec annotation de Wizard avec son bâton

En analysant la matrice de confusion de la figure 7 obtenue lors de l'entraînement précédent, on remarque que les scores pour Wizard sont les moins bons avec seulement 33% de déetections correctes, 67% de déetections manquantes et 15% de confusion avec le background.

Notre première idée pour améliorer ces scores était d'enlever le bâton du Wizard puisqu'il est souvent assez loin du personnage et rajoute du fond d'image dans l'annotation, ce qui peut rendre Wizard plus difficile à reconnaître. La figure 8 montre un exemple d'annotation modifiée.



(a) Annotation avec bâton



(b) Annotation sans bâton

FIGURE 8 – Ré-annotation du Wizard

3.2 Différentes approches

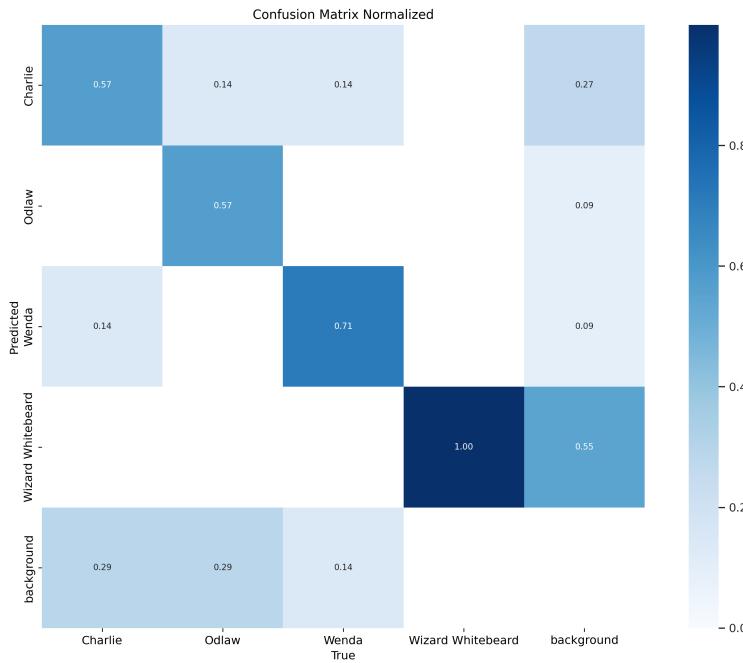


FIGURE 9 – Matrice de confusion avec annotation de Wizard sans son bâton

Avec ces nouvelles annotations, on obtient la matrice de confusion de la figure 9. On remarque une bonne amélioration des détections du Wizard, tous les Wizard sont détectés, mais il y a 55% de détections en trop.

En voulant améliorer davantage les résultats de toutes les classes, nous avons eu une autre idée : au lieu de faire une annotation avec le corps en entier, nous avons pensé à annoter uniquement la tête des personnages car leurs corps ne sont que rarement visibles, ainsi la taille des objets à détecter varie moins, comme on peut le voir sur la figure 11, et on conserve les détails les plus importants, ce qui peut aider à la reconnaissance. La figure 10 montre un exemple d'annotation modifiée.

3.2 Différentes approches

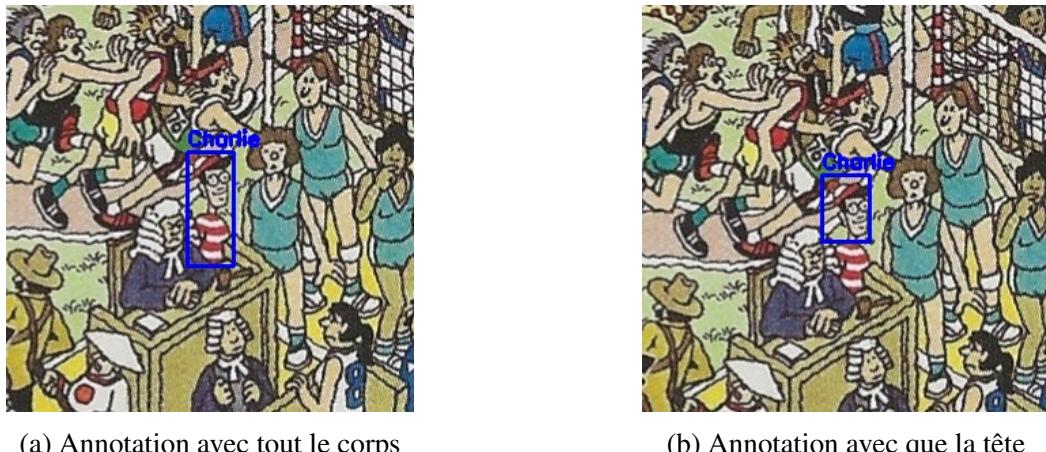
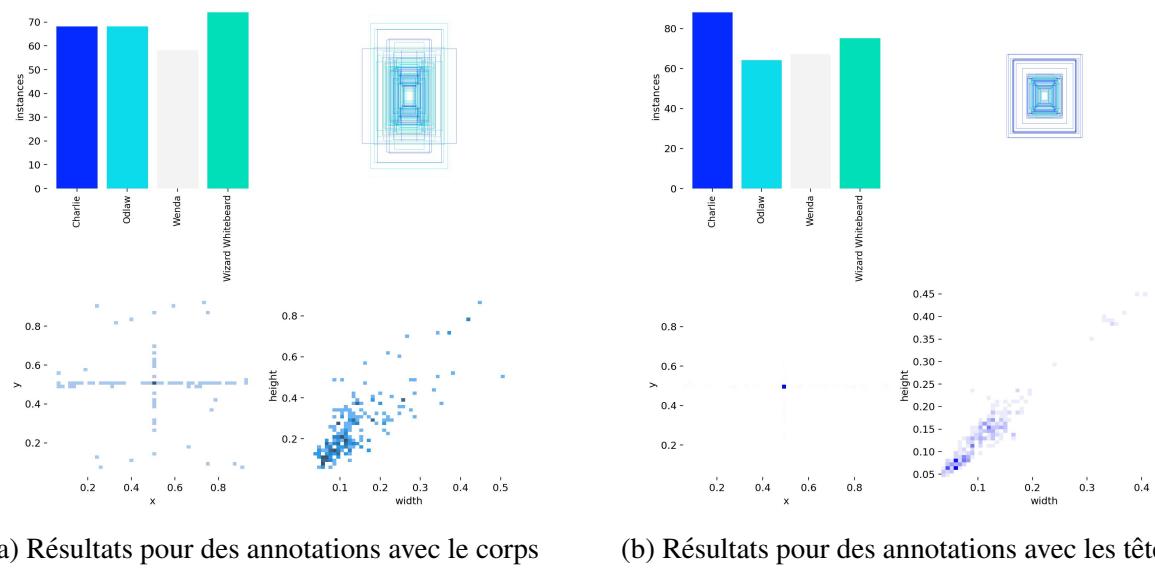


FIGURE 10 – Ré-annotation avec uniquement la tête



(a) Résultats pour des annotations avec le corps (b) Résultats pour des annotations avec les têtes

FIGURE 11 – Résultats des labels trouvés avant et après ré-annotation des têtes

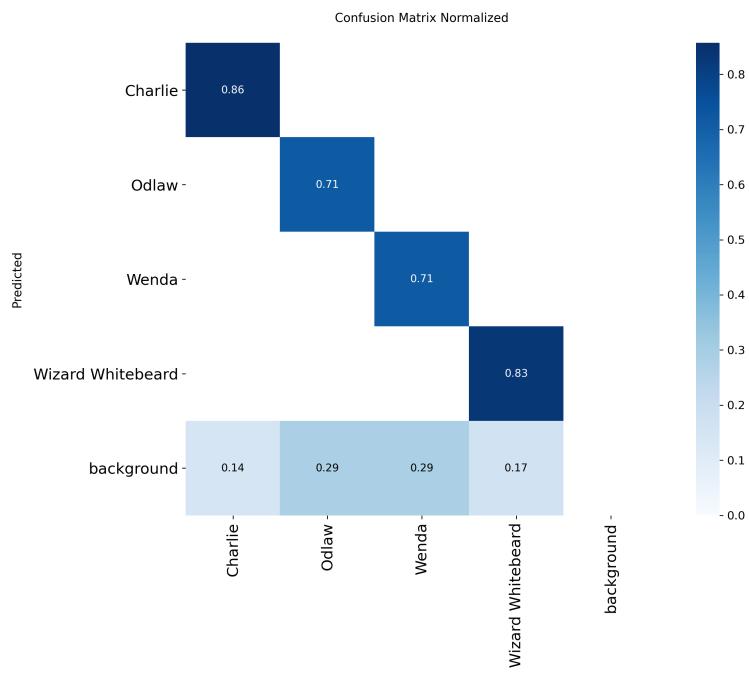


FIGURE 12 – Matrice de confusion pour des annotations avec les têtes seulement

Finalement, on obtient la matrice de confusion de la figure 12. Désormais, même si tous les personnages ne sont pas trouvés, on en trouve plus qu'avant de manière générale et il n'y a plus de confusion entre les classes, ainsi lorsque qu'un personnage est détecté, on est sûr qu'il s'agit du bon.

4 Analyse détaillée des résultats

L'analyse qui suit concerne le dernier modèle entraîné le plus performant qui a été présenté dans la sous-sous-section 3.2.3

4.1 Analyse quantitative

La matrice de confusion figure 12 est en fait biaisée, car elle a été calculée sur les imagettes du dossier validation, donc uniquement des parties restreintes des images complètes. La première chose qui saute aux yeux est qu'il n'y a aucune prédiction du modèle dans du "background" lorsqu'on prédit sur les imagettes. En revanche, en faisant la prédiction sur des images complètes (voir sous-section 4.2), il arrive que le modèle ne détecte pas un personnage et le considère comme "background", même si cela est rare.

Les analyses des métriques obtenues que nous allons faire vont donc être faites dans ce contexte.

4.1.1 Analyse de la loss

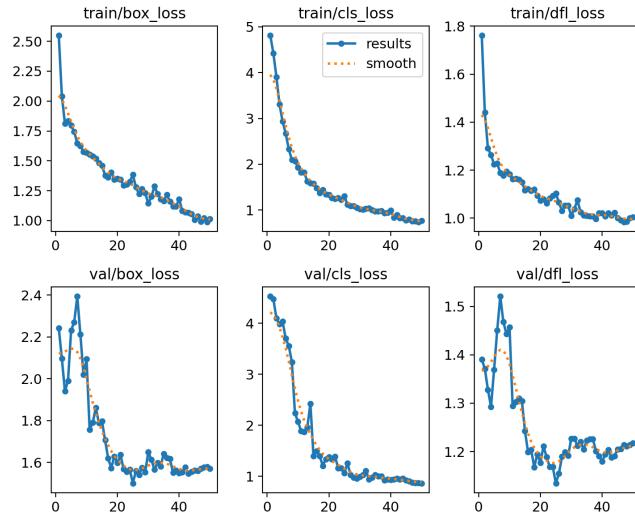


FIGURE 13 – Courbes des loss

La première chose que l'on peut remarquer est que notre modèle sous-apprend. En effet, sur les trois courbes de validation, la loss stagne après une vingtaine d'epochs. De plus, la courbe de train continue de descendre, cela veut dire que le modèle peut encore apprendre plus. Nous nous y étions attendus, car nous savions dès le début que nous avions trop peu d'images dans notre base de données.

4.1.2 Analyse du rappel

Dans cette partie, nous n'allons analyser que la courbe de rappel. C'est ce qui nous intéresse le plus pour ce problème : le "cas d'utilisation" de ce modèle est de trouver les personnages du livre. Cela signifie que si le modèle se trompe et donne des prédictions là où il n'y a pas de personnage, ce n'est pas très grave, tant que les vrais personnages sont trouvés.

De plus, la Précision n'aurait pas eu de sens ici (comme la matrice de confusion) car elle n'aurait pas reflété le fait que le modèle se trompe sur des images complètes. Le rappel ici a du sens, car il y a une imagette par label, donc si on ne trouve pas un personnage sur une imagette, on peut être confiant que l'on ne l'aurait pas trouvé sur l'image complète.

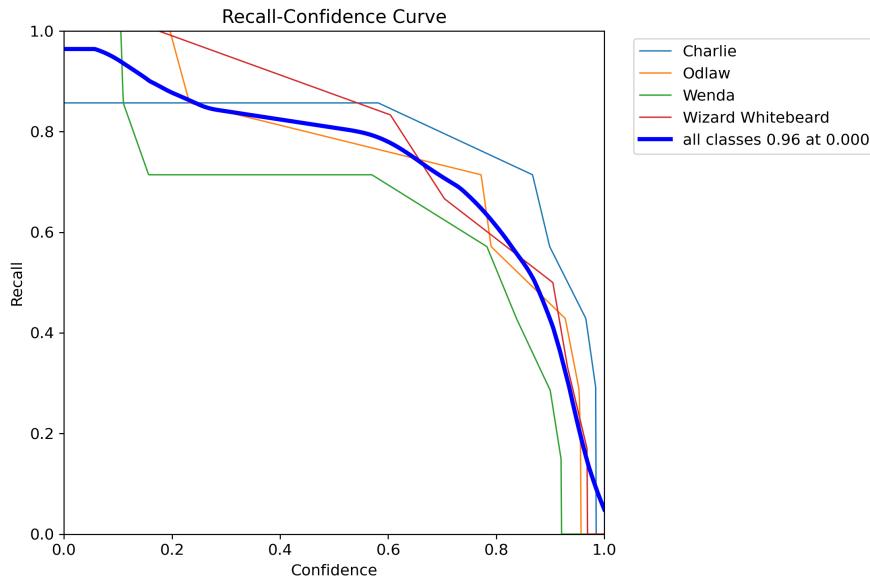


FIGURE 14 – Courbes du rappel

Cette courbe n'est pas entièrement satisfaisante, en particulier pour Wenda; mais elle reste très convenable pour les données que nous avions. Elle nous permet aussi de choisir une confiance pour utiliser le fichier python `full_image_predict.py` qui en a besoin pour filtrer les boîtes. Le plateau du rappel moyen de toutes les classes (en bleu) entre 0.2 et 0.6 paraissant être un bon endroit, nous avons choisi une valeur de 0.5 par défaut, pour éviter d'avoir trop de prédictions par image que l'utilisateur devrait filtrer (nous en avons quand même beaucoup sur certaines images).

4.2 Analyse qualitative

Voyons déjà à quel point le modèle fait de fausses prédictions sur quelques images de l'ensemble de test :

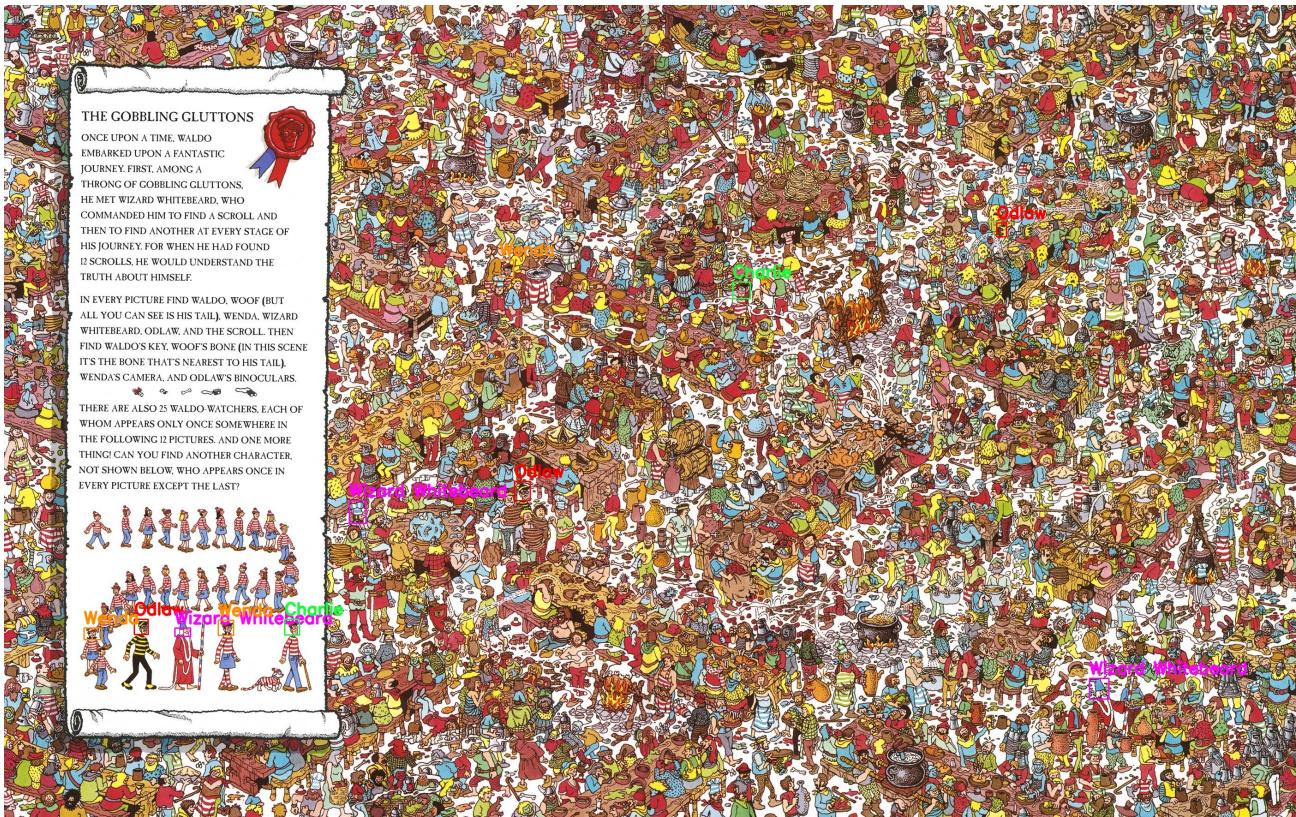


FIGURE 15 – Prédictions sur l'image 3-1-0

Sur la figure 15 les prédictions du modèle sont plutôt satisfaisantes, tous les personnages ont été trouvés, même si quelques personnages du fond ont été détectés à tort (un Odlaw, une Wenda et un Sorcier qui étaient en fait du fond).

De même, sur la figure 16 (demi-page) tous les personnages ont été trouvés (une Wenda et un Charlie) avec une erreur de prédiction du fond en Charlie. C'est encourageant, car les motifs rouges et blancs de cette image auraient pu mener à plus d'erreurs de détections. Cela a probablement été évité, car on ne considère plus que la tête des personnages, et une des raisons secondaires qui avait aussi motivé le passage a des annotations uniquement "tête" était que le modèle "corps" confondait parfois des rayures avec les vêtements des personnages cherchés.

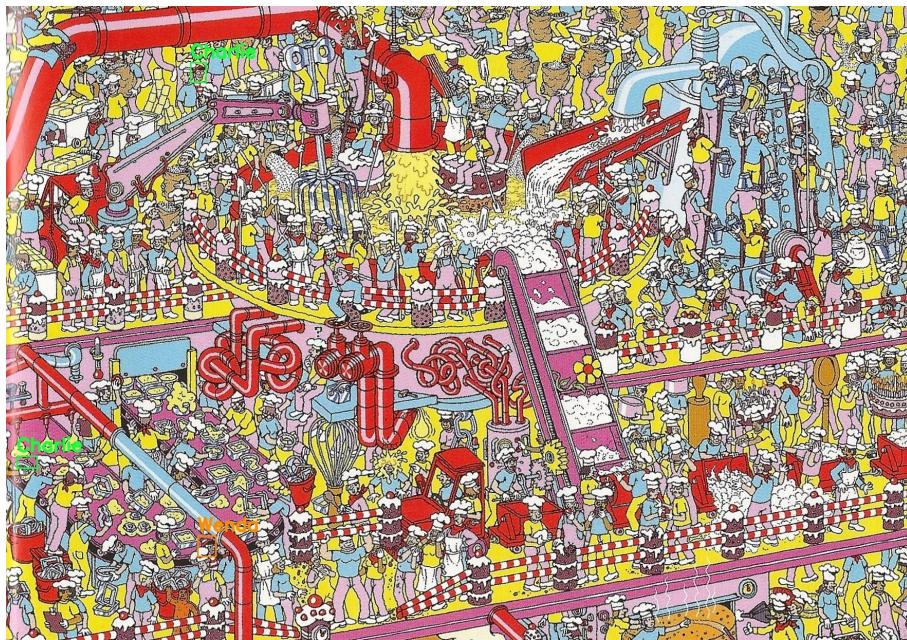


FIGURE 16 – Prédiction sur l'image 5-6-2 (image coupée)



FIGURE 17 – Prédiction sur l'image 6-12-2 (image coupée)

Dans nos données de test, nous n'avons qu'un seul cas où le modèle se trompe de classe, figure 17 (Charlie au lieu de Wenda). C'est assez rassurant, étant donné nos inquiétudes initiales sur la ressemblance entre Charlie et Wenda. De plus étant donné notre problème, nous sommes toujours contents qu'il ait détecté un personnage, même avec une mauvaise classe.

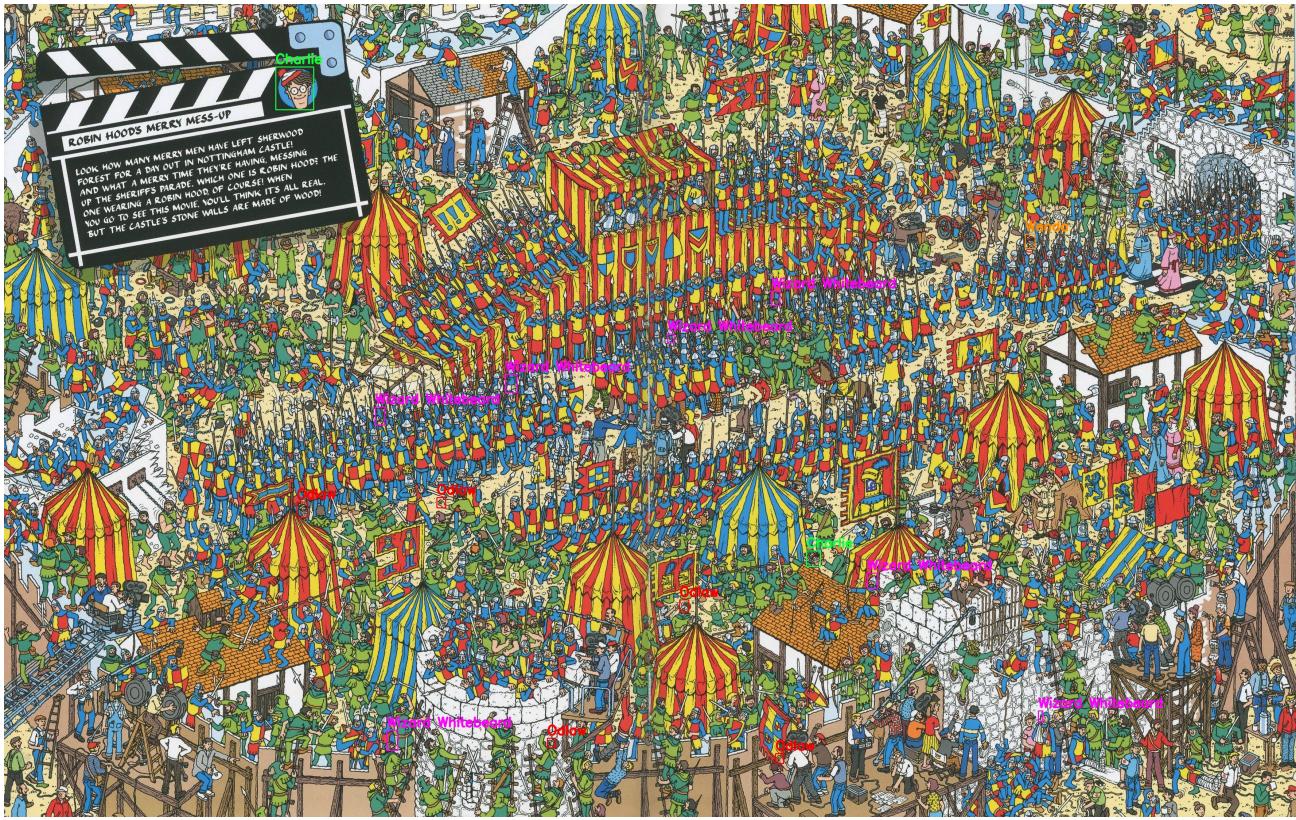


FIGURE 18 – Prédictions sur l'image 4-10-0

Dans ce cas-ci, le modèle a eu plus de mal avec le fond, avec 6 Sorciers et 2 Odlaws détectés à tort, et ce dû aux uniformes des soldats qui sont de la même couleur que le chapeau du sorcier. Cependant, encore une fois, l'intégralité des personnages a été trouvée.

Pour la figure 19 également le modèle a rencontré des difficultés, comme les marins partagent des similitudes avec Wenda (chapeau, couleurs) ils ont été détectés comme tels à 5 reprises. De plus Wenda elle-même n'a pas été trouvée par le modèle (en bas à gauche de l'image, derrière les épines dorsales du monstre), le reste des personnages ayant bien été détecté.

Nous avons aussi remarqué que notre modèle se débrouille mal avec des images de basse résolution et ne trouve pas ou très peu les personnages (comme figure 20 avec uniquement Wenda et Wizard de trouvés avec le seuil de 0.5). Sur l'image 1-5-0 (de l'ensemble de validation), c'est même pire ; avec un seuil de 0.5, aucun personnage valide n'est trouvé.



FIGURE 19 – Prédictions sur l'image 6-3-0 (image coupée)

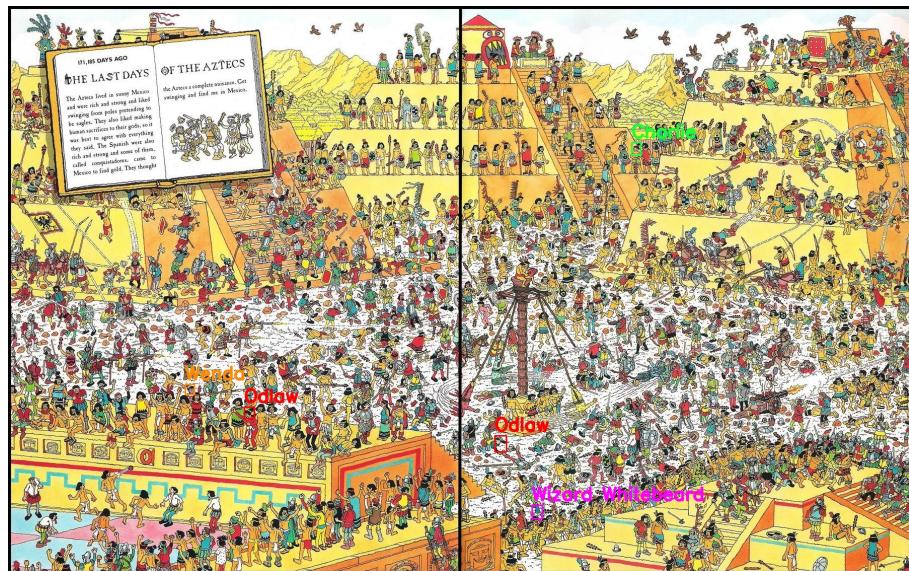


FIGURE 20 – Prédictions sur l'image 2-7-0 (image coupée)

Cependant, malgré quelques erreurs de détection due à une mauvaise connaissance de certains

fonds, les résultats sont encourageants, car la majorité du temps tous les personnages sont détectés. Le modèle actuel peut servir de pré-traitement pour un utilisateur qui n'a qu'à faire le tri entre les quelques bonnes et mauvaises détections.

5 Conclusion

5.1 Bilan

Notre projet a quelques aspects originaux, en effet, nous avons voulu résoudre le jeu "Où est Charlie" donc la manière de juger nos modèles est différente afin de parvenir à un produit final qui répond à notre problématique. Par exemple, nous avons été moins concernés par l'exactitude des boîte englobante autour des personnages, car un utilisateur souhaiterait seulement trouver les personnages, mais n'a pas besoin d'avoir un contour parfait de l'objet. On peut même aller plus loin et dire que l'utilisateur n'a pas non plus absolument besoin de connaître les classes trouvées et le pourcentage de confiance pour chaque objet trouvé, il peut seulement parcourir les objets trouvés pour rapidement localiser les personnages. Notre outil est donc une aide au jeu, et donc n'a pas besoin d'être entièrement précis. En revanche, comme expliqué dans la sous-sous-section 4.1.2, c'est le rappel qui nous a le plus intéressé.

Nos diverses tentatives nous ont permis d'améliorer convenablement les résultats de notre modèle. La première version sur les images complètes n'était vraiment pas concluante, mais sur des imagettes, nous avons commencé à obtenir des résultats convaincants.

5.2 Difficultés rencontrées

La principale difficulté a été de constituer notre base de données, car les images des livres ne sont pas publiques. Nous avons dû croiser plusieurs sources et nous procurer un livre rassemblant plusieurs éditions. Malgré nos efforts, nous ne pouvions pas obtenir suffisamment de données pour entraîner un tel modèle parfaitement. Aussi, pour cette raison, nous avons été contraints de veiller à la qualité de notre base de données et de nos annotations, c'est pourquoi nous avons ré-annoté plusieurs fois les images et tenté différentes solutions.

5.3 Perspectives et améliorations

Malgré nos efforts, notre base de données n'était pas encore au point. En effet, les résultats sur des images de basse résolution comme sur la figure 20 montrent que nous pourrions avoir de meilleurs résultats avec des images de meilleure qualité. Il aurait aussi fallu obtenir toutes les images d'une unique et même source afin d'avoir des images de la même qualité et surtout d'une meilleure qualité. Par exemple, si nous avions toute la série des livres et un très bon scanner.

Cependant, on peut se dire que l'utilisateur lambda de notre modèle utiliserait cette aide au jeu avec une photo prise de son téléphone par exemple. Dans ce cas, il vaudrait mieux avoir des images d'une

qualité comparable à une photo prise au smartphone, afin d'obtenir un modèle qui répond vraiment mieux au besoin.

Aussi, nous avons choisi des imagettes de taille 300*300, mais les images entières ne sont pas toutes de la même taille et ces tailles varient beaucoup d'une image à l'autre. Ainsi, nos imagettes n'ont pas la même résolution. Une idée pour pallier ce problème serait de créer les imagettes à l'échelle, c'est-à-dire utiliser des tailles d'imagettes proportionnelles aux tailles des images.

Une autre piste d'amélioration serait de copier-coller les têtes des personnages sur d'autres imagettes ne contenant pas d'objets. Ceci permettrait d'avoir des fonds d'image différents et donc plus de contexte, car YOLO n'utilise pas les images qui ne contiennent pas d'objet à détecter lors de l'entraînement. En effet, lorsque l'on travaille avec nos imagettes, une grande partie du "background" des images entières n'est pas utilisé pour l'entraînement alors que c'est un contexte qui peut être utile à YOLO.

Ensuite, puisque nous nous sommes orientés dans la conception d'un modèle qui puisse être réellement utilisé comme une aide au jeu, nous avons délaissé certaines capacités du modèle YOLO, comme expliqué plus haut. Par exemple, nos annotations des têtes seulement ne permettent pas d'avoir un modèle qui détecte exactement le contour du corps d'un personnage. Si nous voulions absolument créer un tel modèle, nous pourrions envisager d'utiliser le modèle actuel, puis imaginer un autre modèle ou algorithme qui serait chargé de prolonger les boîtes englobantes afin de détecter exactement les personnages.

Enfin, pour pouvoir mieux comparer les résultats finaux des différentes méthodes et des différents entraînements, nous aurions dû lancer l'algorithme de détection pour des images entières (`full_images_predict.py`) sur les données de test puis recalculer les différentes métriques d'évaluation sur ces prédictions des images entières. En effet, les métriques que nous avons montrées ont été faites sur des imagettes et donc n'évaluent pas correctement la finalité de notre problème, à savoir la prédiction sur une image entière. Nous n'avons malheureusement plus le temps pour tenter cette approche d'évaluation.

Table des figures

1	Les personnages des livres	1
2	Images de la base de données.	2
3	Image retirée.	2
4	Structure du dossier pour l'entraînement du modèle YOLO	3
5	Exemples de personnages cachés.	4
6	Exemples d'imagettes	6
7	Matrice de confusion avec annotation de Wizard avec son bâton	7
8	Ré-annotation du Wizard	8
9	Matrice de confusion avec annotation de Wizard sans son bâton	9
10	Ré-annotation avec uniquement la tête	10
11	Résultats des labels trouvés avant et après ré-annotation des têtes	10
12	Matrice de confusion pour des annotations avec les têtes seulement	11
13	Courbes des loss	12
14	Courbes du rappel	13
15	Prédictions sur l'image 3-1-0	14
16	Prédictions sur l'image 5-6-2 (image coupée)	15
17	Prédictions sur l'image 6-12-2 (image coupée)	15
18	Prédictions sur l'image 4-10-0	16
19	Prédictions sur l'image 6-3-0 (image coupée)	17
20	Prédictions sur l'image 2-7-0 (image coupée)	17

Références

- [1] G. Jocher and J. Qiu, “Ultralytics yolo11,” 2024.
- [2] vc1492a, “Hey waldo.” <https://github.com/vc1492a/Hey-Waldo>, 2020. Accédé : 2025-05-17.
- [3] tadejmagajna, “Hereiswally.” <https://github.com/tadejmagajna/HereIsWally>, 2020. Accédé : 2025-05-17.
- [4] B. Kenstler, “Where’s waldo : Terminator edition.” <https://hackernoon.com/wheres-waldo-terminator-edition-8b3bd0805741>, 2017. Accédé : 2025-05-17.
- [5] K. Brandt, “wally dset v2 computer vision project.” <https://universe.roboflow.com/kurtis-brandt/wally-dset-v2>, 2021. Accédé : 2025-05-17.
- [6] “r/random acts of amazon.” https://www.reddit.com/r/Random_Acts_Of_Amazon/search/?q=waldo. Accédé : 2025-05-17.
- [7] M. Handford, *Où est Charlie ? Tout Charlie dans une édition à mettre dans toutes les poches*. Gründ, 1st ed., 2010.
- [8] “Charlie-db.” <https://github.com/pekatour/Charlie-DB>, 2025. Accédé : 2025-05-17.
- [9] Label Studio, “Label studio.” <https://labelstud.io/>, 2024. Accédé : 2025-05-17.
- [10] J. Malipeddi, “Visualize-yolo-annotations.” <https://github.com/jiteshm17/Visualize-Yolo-annotations>, 2021. Accédé : 2025-05-17.