



Rapport projet Deep Learning : Détection d'objets

Détection et classification des classes de personnages du jeu Dofus

**EL HARRANI Abdelali
PEYROUTON Clément**

MASTÈRE VALDOM 2025

1 - Introduction	2
2 - Collecte et prétraitement des images	3
3 - Entraînement du modèle	8
4 - Résultats	8
5 - Conclusion	16

1 - Introduction

Le répertoire *github* associé à ce rapport est disponible [ici](#).

La détection d'objets est un domaine clé de l'intelligence artificielle et de la vision par ordinateur, avec des applications variées allant de la surveillance à l'analyse d'images médicales. Dans le cadre de ce projet, nous avons choisi d'explorer cette technologie en appliquant un modèle de détection d'objets à un univers qui nous est familier et qui nous passionne : le jeu Dofus. Dofus est un MMORPG en ligne développé par Ankama, se déroulant dans un univers médiéval-fantastique. Le jeu propose 19 classes de personnages, chacune possédant ses propres caractéristiques et capacités. Les joueurs évoluent dans un monde ouvert, accomplissent des quêtes, affrontent des monstres et interagissent avec d'autres joueurs à travers un système de combat au tour par tour. Une capture d'écran de l'interface du jeu ainsi qu'une image des 19 classes du jeu sont présentées ci-dessous (*figure 1 et 2*).



Figure 1: Interface graphique du jeu



Figure 2 : Les 19 classes de dofus

Cette initiative avait un double objectif : approfondir notre savoir faire en apprentissage profond de manière ludique et développer un modèle capable d'identifier automatiquement les différentes classes de personnages présentent dans des captures d'écran du jeu.

Pour cela, nous avons entraîné des modèles YOLOv11 sur un dataset d'images issues du jeu. YOLO (*You Only Look Once*) est une série de modèles de détection d'objets, reconnue pour sa rapidité et son efficacité dans le domaine de la vision par ordinateur. YOLOv11 est l'avant dernière version de ce modèle polyvalent qui est capable d'accomplir diverses tâches, telles que la détection, la segmentation, l'estimation de pose ou la classification, tout en optimisant les performances et en s'adaptant à divers modes d'utilisation.

La première étape a consisté en la collecte et le prétraitement des images afin de constituer une base de données exploitable. Nous avons ensuite procédé à l'annotation des images, une étape essentielle pour l'apprentissage supervisé, en utilisant des outils spécialisés facilitant le travail collaboratif.

Une fois notre dataset préparé et segmenté en ensembles d'entraînement, de validation et de test, nous avons entraîné les modèles. Enfin, nous avons analysé les résultats obtenus pour évaluer l'efficacité des modèles et identifier d'éventuelles pistes d'amélioration.

Ce rapport détaille l'ensemble du processus, depuis la collecte des données jusqu'à l'interprétation des résultats, en mettant en avant les choix méthodologiques effectués et les défis rencontrés.

2 - Collecte et prétraitement des images

A) Collecte des images

Afin de constituer notre jeu de données, nous avons réalisé des captures d'écran directement en jeu en utilisant le mode créature (*figure 3*). Il s'agit d'un mode d'affichage simplifié des personnages qui met en évidence leur classe et n'affiche pas leurs équipements. Cela a pour objectif de réduire les erreurs du modèle lié aux ambiguïtés induites par l'équipement ou la personnalisation des personnages, rendant la classe difficilement distinguable même pour les annotateurs.

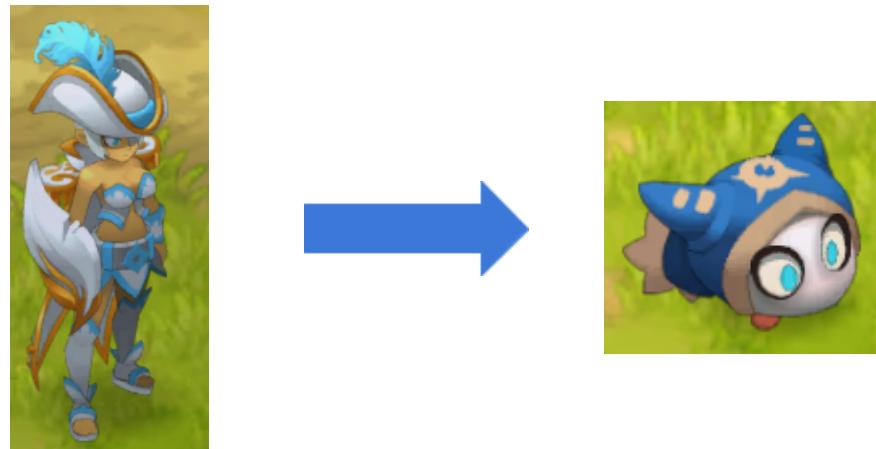


Figure 3: Comparaison entre les modes d'affichages classique et créature

Une fois les captures d'écran réalisés nous avons réalisé un script Python (*ImagesPreprocessing.ipynb*) permettant de :

1. **Supprimer les bordures des images** où se trouve l'interface du jeu afin de ne conserver que la zone jouable pouvant contenir des personnages (cf figure 1).
2. **Découper chaque image en 4 parties égales**, afin d'obtenir des personnages plus grands et donc d'améliorer la détection en augmentant la taille relative des objets dans l'image.
3. **Tri des images** dans le but de supprimer celles ne comportant aucun objet.
4. **Renommer les images** selon une nomenclature définie (*Image1*, *Image2*, *Image3*, ...), facilitant l'organisation et la gestion du dataset.

Deux exemples d'images qui constituent le dataset sont présentés ci-dessous (*figures 4 et 5*).



Figure 4 : Exemple n°1 d'image du dataset



Figure 5 : Exemple n°2 d'image du dataset

Finalement, après préparation et tri des images collectées nous avons obtenu un dataset de taille 603. Les scripts et le dataset sont stockés et disponibles sur notre dépôt GitHub <https://github.com/Clemspey/DeepLearningProject>.

B) Annotations

Dans notre cas, l'annotation des images a été organisée de la façon suivante :

- 1) Division du dataset en 2 (1 partie par annotateur)
- 2) Annotation du dataset par l'annotateur
- 3) Révision des annotations par le second annotateur
- 4) Regroupement des images annotées en un dataset et divisions en 3 sous-ensembles (entraînement, validation et test).

Afin d'annoter nos images, nous avons dans un premier temps choisi d'utiliser la plateforme **Make Sense** (<https://www.makesense.ai>) car cette dernière permet de réaliser des annotations Yolo sans téléchargement requis. Or, il nous a nous été impossible de charger des fichiers d'annotations réalisés par un autre annotateur ce qui bloquait la possibilité de vérification des labels. Nous avons donc migré sur la plateforme **Roboflow** (<https://roboflow.com>) car elle présente plusieurs caractéristiques correspondant à nos besoins :

- Pas de téléchargement requis.
- Possibilité de répartir la charge de travail entre les différents annotateurs associés au projet.
- L'ensemble des images et des annotations sont stockés en ligne, ce qui rend leur accès facile pour l'ensemble des annotateurs.
- Possibilité d'adapter la couleur des boîtes englobantes en fonction de la classe, ce qui rend la review des annotations plus rapide et simple.

- Présence d'informations descriptives liées au dataset (*répartition des classes sur les images, nombre de classe moyenne par image, ...*)
- Possibilité de générer automatiquement les sous-ensembles d'entraînement, de validation et de test selon une proportion de répartition définie par l'utilisateur.

Voici deux exemples d'images annotées (*figures 7 et 8*):



Figure 7 : exemple d'image annotée n°1



Figure 8 : exemple d'image annotée n°2

C) Informations sur le dataset final

Étant donné que le nombre de classes est relativement élevé par rapport à la taille totale du dataset, certaines classes sont naturellement sur-représentées tandis que d'autres sont sous-représentées. Après avoir tenté d'entraîner le modèle avec une centaine d'images et analysé les premiers résultats, nous avons choisi de réduire la dispersion des classes afin d'obtenir une distribution plus équilibrée. L'objectif était de permettre au modèle d'apprendre plus efficacement les classes initialement sous-représentées. Pour ce faire, nous avons délibérément augmenté le nombre d'images associées aux classes sous-représentées. Bien que la distribution finale ne soit pas homogène, l'écart de représentation des classes sous-représentées a été considérablement réduit permettant une meilleure détection des classes de la part du modèle (*figure 9*).

Distribution des classes au sein du dataset

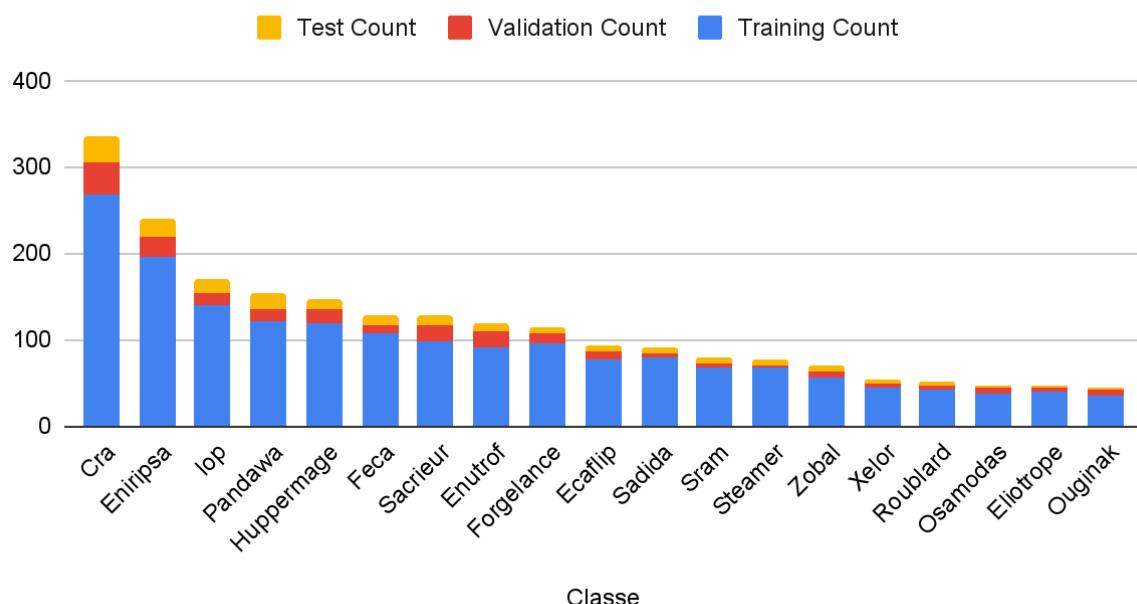


Figure 9 : Graphique de la distribution des classes

De manière générale, notre dataset se distingue par plusieurs atouts majeurs. Tout d'abord, chaque classe y est suffisamment représentée, garantissant un apprentissage équilibré du modèle. De plus, les personnages apparaissent dans une grande variété de décors, sous différents angles (de face, de dos, de côté, ...) et peuvent parfois être partiellement masqués par un objet ou le bord de la carte. Cette diversité permet au modèle d'apprendre à identifier les caractéristiques essentielles de chaque classe, indépendamment de l'environnement ou de la position des personnages.

Une fois toutes les images annotées, nous avons choisi d'affecter 80% du dataset en tant que dataset d'entraînement, 10% en tant que dataset de validation et 10% en tant que dataset de test. Au total, 483 images servent à l'entraînement du modèle, 61 permettent la validation et 59 sont destinées à la phase de test. Un fichier texte correspondant aux labels est associé à chaque image. Il rassemble le numéro de chaque classe représentée ainsi que les coordonnées des boîtes englobantes correspondantes (coordonnées du centre, hauteur et largeur).

3 - Entraînement du modèle

Pour entraîner notre modèle de détection d'objets, nous avons choisi d'utiliser les modèles pré-entraînés **YOLOv11 Nano** et **YOLOv11 Small** en raison de leurs performances équilibrées entre précision et rapidité. YOLOv11 Nano, étant une version extrêmement légère, est particulièrement adapté aux environnements à faibles ressources tout en offrant des résultats satisfaisants pour des tâches en temps réel. De son côté, YOLOv11 Small constitue un compromis entre compacité et précision, permettant d'obtenir des détections plus fines tout en restant efficace en termes de coût de calcul. Le gain observé en termes de mAPval pour la détection sur MS COCO est important, passant de 39,5 à 47 avec YOLOv11 Small par rapport à la version nano. Toutefois, ce gain devient de moins en moins significatif à mesure que la taille du modèle augmente : le passage de YOLOv11 Small à YOLOv11 Mid, par exemple, fait augmenter la mAPval de seulement 47 à 51,5. Cela met en évidence que l'amélioration en précision tend à se stabiliser à mesure que l'on utilise des modèles de plus grande taille (source : Ultralytics). Le fait d'avoir sélectionné ces deux modèles va nous permettre de confronter leurs performances appliquées à notre problème par la suite.

Après avoir rempli le fichier dataset.yaml conformément aux pré-requis de YOLO, nous avons lancé l'entraînement des modèles sur 50 époques, avec un batch size de 50 et une taille d'image de 512. Un paramètre de patience de 10 a également été configuré, ce qui permet d'interrompre l'entraînement si les métriques de validation ne montrent plus d'amélioration. Ce mécanisme est conçu pour prévenir le sur-apprentissage en arrêtant l'entraînement avant que le modèle ne commence à "apprendre par cœur" les données d'entraînement. Les modèles résultants ont été nommés yolo11n_dofus et yolo11s_dofus.

4 - Résultats

L'ensemble des résultats sont stockés dans les dossiers **yolo11n_dofus** (*Yolov11 nano*) et **yolo11s_dofus** (*Yolov11 small*) du dépôt.

A) Analyse de l'apprentissage des modèles

Avec la version nano, nous avons obtenu une précision de 92%, un rappel de 90%, une mAP50 de 93% et une mAP50-95 de 75% (*figure 10*). Concernant la version small, la précision est cette fois-ci de 97%, le rappel de 95%, une mAP50 de 96% et une mAP50-95 de 80% (*figure 11*). Nos deux modèles sont donc très performants car les scores de rappel, de précisions et les mAP50 sont proches de 1, mais aussi car les mAP50-95 proche ou égale à 0,8.

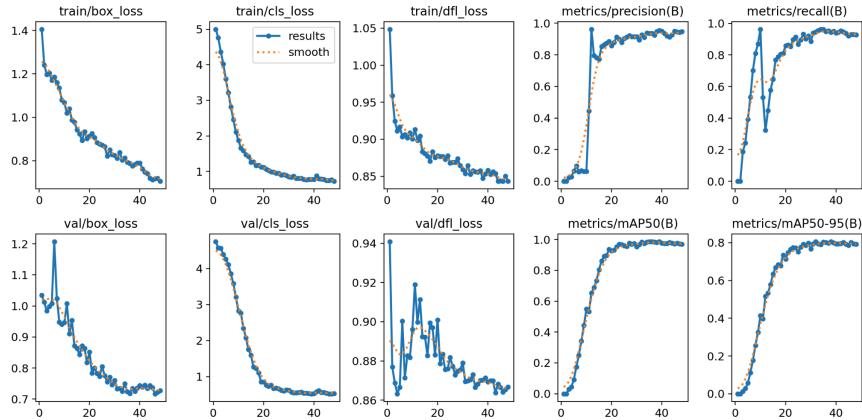


Figure 10 : Résultats de l'entraînement du modèle Nano

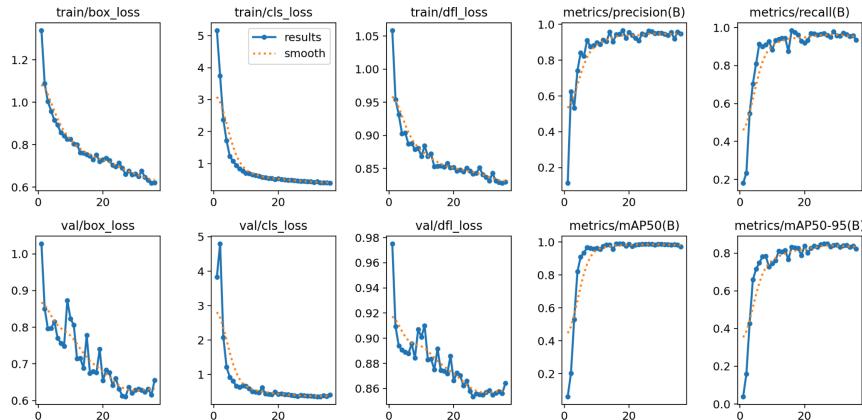


Figure 11 : Résultats de l'entraînement du modèle Small

Cela montre donc aussi que la version small a une meilleure détection d'objet mais aussi une meilleure prédiction des classes que la version nano, car ces scores sont supérieurs. De plus, comme on peut le voir au niveau des courbes des fonction de pertes, le nombre plus important de paramètres de la version small permet une convergence plus rapide et stable. Malgré le nombre de paramètres plus important, la version Small n'a pas été plus coûteuse à entraîner que la version Nano, puisque l'entraînement s'est terminé au bout de 36 époques (env. 6mn) pour la Small contre 48 (env. 6m30) pour la Nano.

B) Analyse des matrices de confusions

Suite aux entraînements des deux modèles, nous avons évalué avec notre ensemble de test nos modèles pour obtenir différentes métriques. L'une de ces métriques sont les matrices de confusion normalisées (*figure 12*) qui nous renseigne sur la proportion de classification correcte et incorrecte.

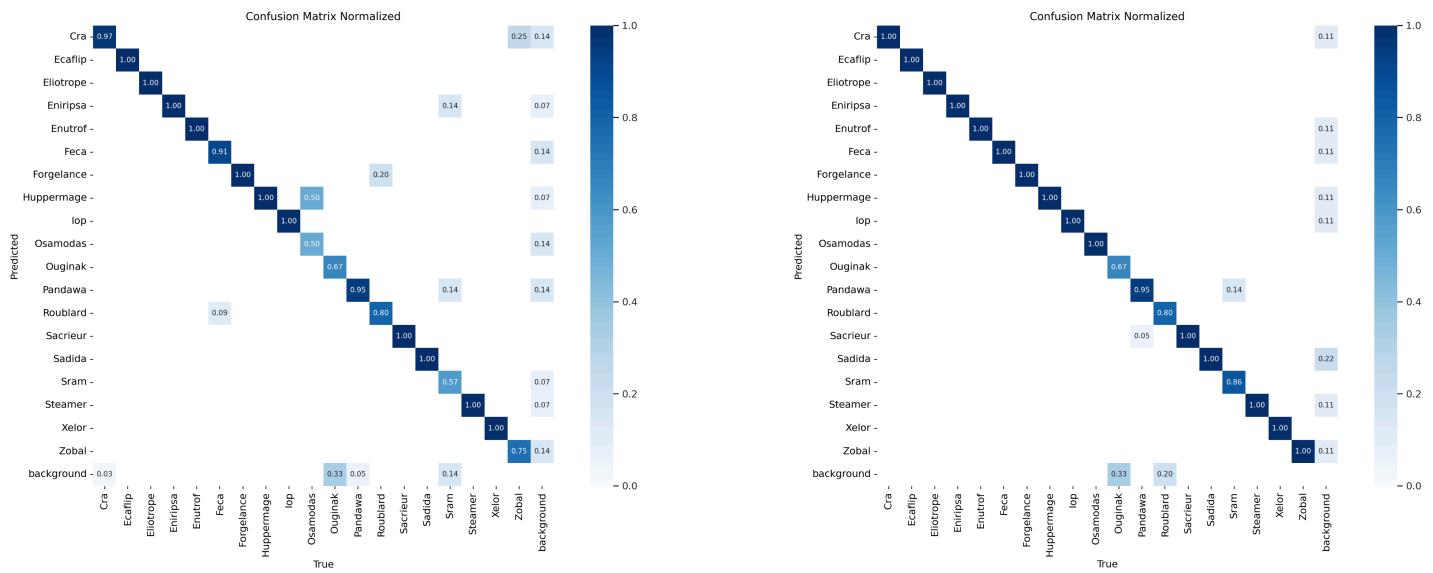


Figure 12 : Matrices de confusions des modèles nano (à droite) et small (à gauche)

Lorsque l'on regarde nos matrices de confusion, on peut observer des “diagonales fortes” sur nos matrices de confusion, c'est-à-dire des valeurs proches de 1 au niveau des diagonales correspondent aux prédictions correctes.

Plus en détails, les résultats du modèle Nano montrent une capacité de généralisation très précise sur la majorité des classes, avec 14 classes sur 19 présentant un taux de classification correct supérieur à 90 %. Cependant, certaines classes restent sujettes à des confusions notables, notamment Osamodas, Ouginak, Roublard, Sram et Zobal.

Par exemple, 50 % des Osamodas ont été prédits comme Huppermage, très probablement en raison de similitudes visuelles entre ces deux classes. De même, certaines classes comme Ouginak et Zobal affichent des erreurs de classification plus fréquentes, indiquant une difficulté du modèle à bien les distinguer. On remarque que les erreurs de classification surviennent exclusivement pour des classes sous-représentées (*cf figure 9*). On note également une tendance du modèle à confondre certaines classes avec le background, ce qui suggère que certains personnages peuvent être insuffisamment distincts de leur environnement ou partiellement masqués dans certaines images.

Avec le modèle Small, les bons résultats observés précédemment sont maintenus, et certaines erreurs ont été partiellement corrigées. Toutefois, des confusions significatives persistent, notamment pour l'Ouginak, qui est la classe la moins représentée.

Les performances des classes Pandawa, Roublard et Sram montrent encore une marge d'amélioration, avec des erreurs de classification récurrentes. Enfin, le modèle continue à confondre certaines classes avec le background.

C) Analyse des courbes de performance

Comme expliqué dans la partie précédente, nous avons évalué différentes métriques sur notre ensemble de test pour nos deux modèles. Outre les matrices de confusion, nous avons aussi obtenues les courbes F1, de précision, de rappel et précision rappel (*figure 13 & 14*).

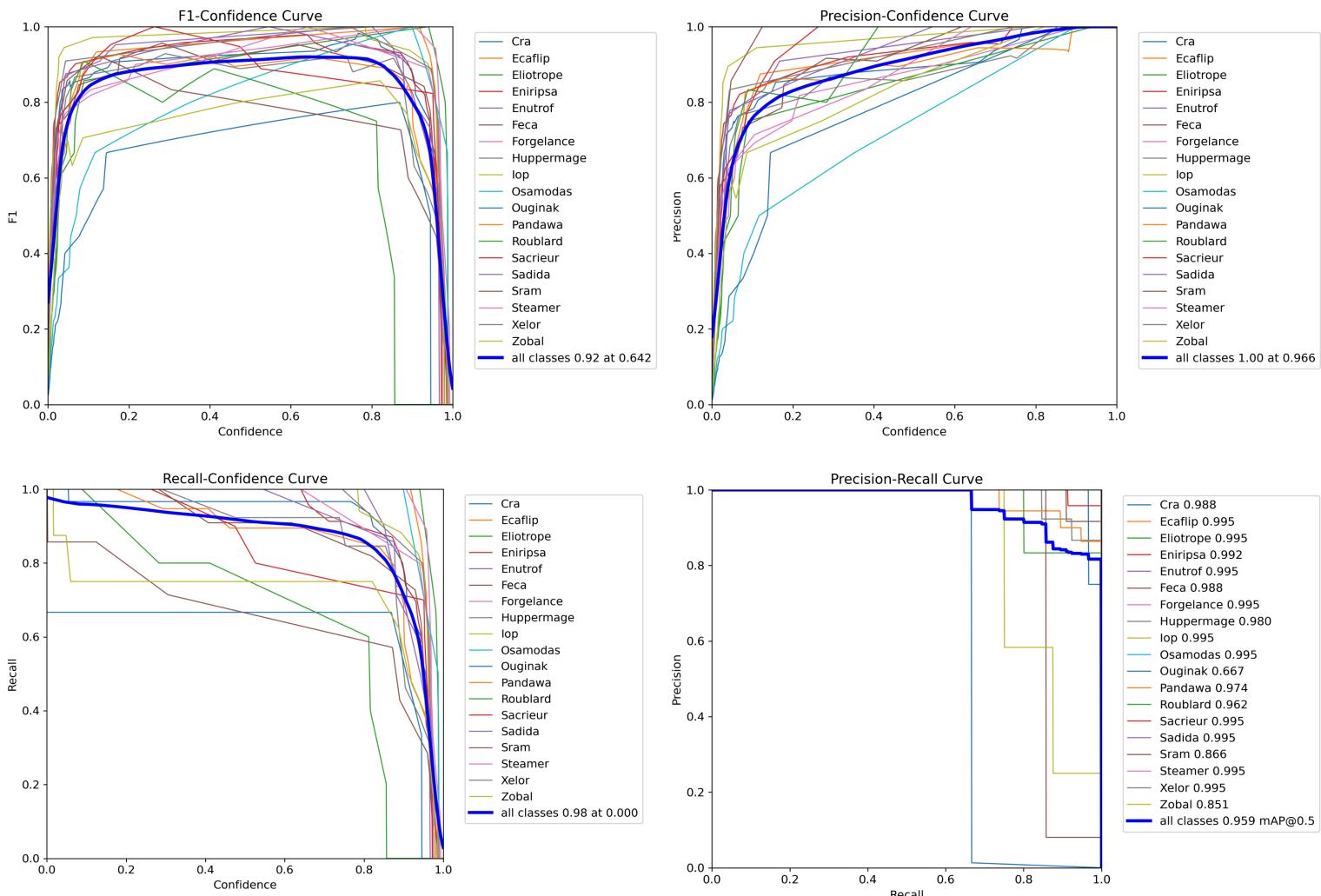


Figure 13 : Courbes F1, Précision, Recall et Précision Recall du modèle Nano

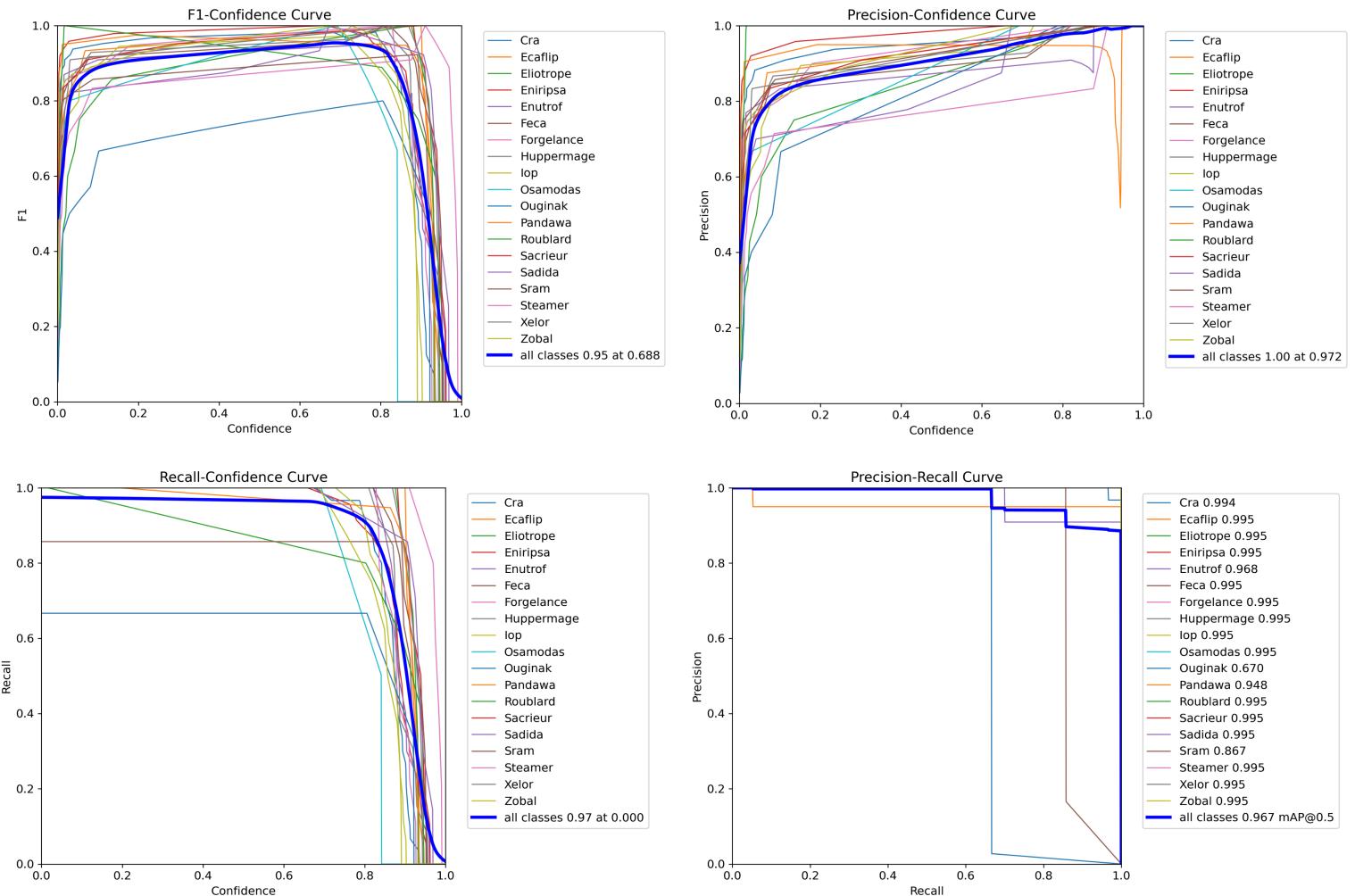


Figure 14 : Courbes F1, Précision, Recall et Précision Recall du modèle Small

Les courbes F1-Confidence, Précision-Confidence et Rappel-Confidence permettent d'évaluer l'impact du seuil de confiance sur les performances des modèles. On observe que pour les deux modèles, la précision augmente avec le seuil de confiance, atteignant des valeurs très élevées au-delà de 0.8. Cela signifie que lorsque le modèle est très sûr de ses prédictions, celles-ci sont généralement correctes. Toutefois, le rappel diminue fortement à mesure que le seuil de confiance augmente, indiquant que certains objets ne sont plus détectés à des niveaux de confiance élevés. Le compromis optimal se situe autour d'un seuil qui maximise la F1-score, observable autour de 0.6 à 0.7 pour les deux modèles.

Les courbes Précision-Rappel montrent des performances globalement solides, avec une précision supérieure à 0.95 pour la plupart des classes, bien que l'Ouginak présente un rappel plus faible, ce qui corrobore les observations faites avec la matrice de confusion. Le modèle Small apporte une amélioration générale, avec un mAP@0.5 passant de 0.959 (Nano) à 0.967 (Small) traduisant une meilleure capacité de détection

et de classification sur l'ensemble des classes. Cependant, certaines confusions persistent, notamment pour les classes les moins représentées.

Globalement, ces résultats confirment que l'augmentation de la capacité du modèle (de Nano à Small) améliore la robustesse et réduit certaines erreurs, bien que des confusions subsistent sur les classes les plus difficiles à distinguer. On notera que du point de vue de la mAP, le gain semble néanmoins très faible car le modèle nano parvenait déjà à de bonnes performances sur un grand nombre de classes.

D) Analyse qualitative

En plus des différentes métriques, nous avons aussi réalisé des inférences sur nos ensembles de validation et de test afin d'apprécier de façon qualitative les prédictions réalisées par nos modèles. L'ensemble des résultats des inférences pour les deux modèles est disponible sur notre GitHub, dans les dossiers correspondant aux modèles.

Dans l'ensemble, comme démontré de façon quantitative, les résultats obtenus sont très bons pour les deux modèles. Cependant, l'appréciation qualitative nous permet de mieux comprendre les erreurs commises par nos modèles. Globalement, on peut distinguer trois grands types d'erreurs :

- **Le modèle ne détecte pas le personnage** (*figure 15*), généralement parce que le contraste entre lui et le décor est faible.
- **Le modèle détecte un élément du background comme un personnage** (*figure 16*), car ce dernier possède une ou plusieurs caractéristiques communes (forme, couleur, etc.).
- **Le modèle inverse deux classes de personnages** (*figure 17*).

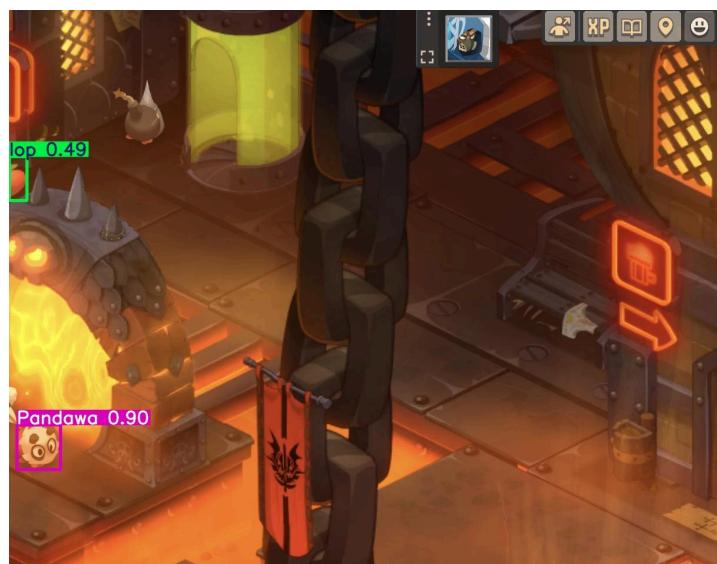


Figure 15 : Problème de détection d'un personnage (en haut à gauche)



Figure 16 : Problème de détection du background comme un personnage (box bleue)



Figure 16 : Problème de prédiction de la classe (box verte)

Généralement, on s'aperçoit que les problèmes **1 et 3** se produisent surtout sur des classes peu représentées dans notre dataset, comme on pouvait également l'observer sur la matrice de confusion. Cela s'explique notamment par le fait que la faible représentation de ces classes n'a pas permis au modèle d'apprendre aussi bien leurs caractéristiques clés que celles des classes plus représentées. La principale solution serait donc d'enrichir notre dataset avec de nouvelles images contenant ces classes, puis de réentraîner notre modèle pour lui permettre de mieux les distinguer.

Concernant le **problème 2**, comme observé dans la figure 16, il peut se produire même pour la classe la plus représentée dans notre dataset, ce qui montre que ce n'est pas dû à un mauvais apprentissage. On remarque que les probabilités renvoyées lors de ce type d'erreur sont généralement faibles ($> 0,5$). Une solution pourrait donc être

de définir un seuil de probabilité de détection plus élevé (ex. : 0,7) afin d'éviter les faux positifs. Cependant, cela entraînerait l'apparition de quelques faux négatifs dans les cas où la détection est correcte malgré une probabilité relativement faible. Or, en parcourant les inférences, on constate que cette modification supprimerait plus de faux positifs qu'elle ne créerait de faux négatifs. Cela serait donc une solution intéressante à mettre en place.

De plus, nous avons aussi essayé de réaliser des inférences avec le modèle Small entraîné (car plus performant) avec des images issues des anciennes versions de Dofus (*figure 18 et 19*).



Figure 18 : Prédiction sur des capture d'écran de Dofus 2.0 (2009-2024)



Figure 19 : Prédiction sur des capture d'écran de Dofus 1.29 (2004-2009)

Comme on peut le voir, le modèle est capable de détecter des objets sur des images issues des deux versions du jeu.

Concernant Dofus 2.0, plus récent, les détections et prédictions du modèle sont correctes, bien que les probabilités soient plus faibles que sur notre dataset de test. Cela s'explique par le fait que les visuels des personnages sont relativement proches entre les deux versions, permettant au modèle de réaliser des prédictions correctes.

Concernant Dofus 1.29, les détections et prédictions du modèle sont moins performantes. Cela s'explique par deux facteurs :

1. **La qualité de l'image** : Il a été impossible de trouver une image en haute résolution avec un nombre limité de personnages sur internet.
2. **Les différences visuelles entre les versions** : Certaines classes ont subi des changements visuels majeurs entre les versions, empêchant le modèle de les détecter correctement.

Néanmoins, le modèle reste capable de détecter et de prédire correctement de nombreux objets malgré les différences entre les versions, ce qui démontre sa robustesse et ses bonnes performances.

Enfin, l'analyse des inférences a également révélé une différence significative entre les versions Nano et Small. Comme mentionné précédemment, la version Small offre de meilleures performances en détection et classification. Cependant, cette amélioration a un coût : une inférence avec Small est environ **2,5 fois plus lente** (250 ms/image contre 100 ms/image) et le modèle est également **plus volumineux** (19,2 Mo contre 5,5 Mo).

Ainsi, malgré des performances légèrement inférieures, la version Nano peut s'avérer plus adaptée dans certaines situations, notamment lorsque le **temps d'inférence doit être optimisé**, que les ressources matérielles sont limitées (ex. : utilisation sur un Raspberry Pi), ou encore lorsque l'amélioration de précision offerte par la version Small n'est pas indispensable.

5 - Conclusion

Ce projet a constitué une excellente opportunité pour approfondir nos compétences en détection et classification d'images. L'utilisation de YOLOv11 s'est révélée pertinente, démontrant que cette technologie allie puissance et simplicité d'utilisation, même avec un dataset de taille modeste. Nos résultats montrent qu'avec un volume de données limité, il est déjà possible d'obtenir des performances satisfaisantes, ce qui souligne l'efficacité du modèle dans ce type de tâche.

Toutefois, plusieurs pistes d'amélioration pourraient être explorées pour aller encore plus loin. L'augmentation de la taille du dataset et une meilleure représentativité des classes sous-représentées permettraient sans doute d'améliorer la robustesse du modèle. De même, tester des architectures plus larges de YOLOv11 ou même la toute nouvelle version YOLOv12 pourrait offrir des gains en précision et en rappel. Bien que notre modèle n'ait pas de réelle utilité, il illustre le potentiel considérable de ces technologies pour de nombreuses applications industrielles. Ce travail met en lumière les vastes possibilités offertes par la détection et la classification d'images assistées par intelligence artificielle dans divers domaines.