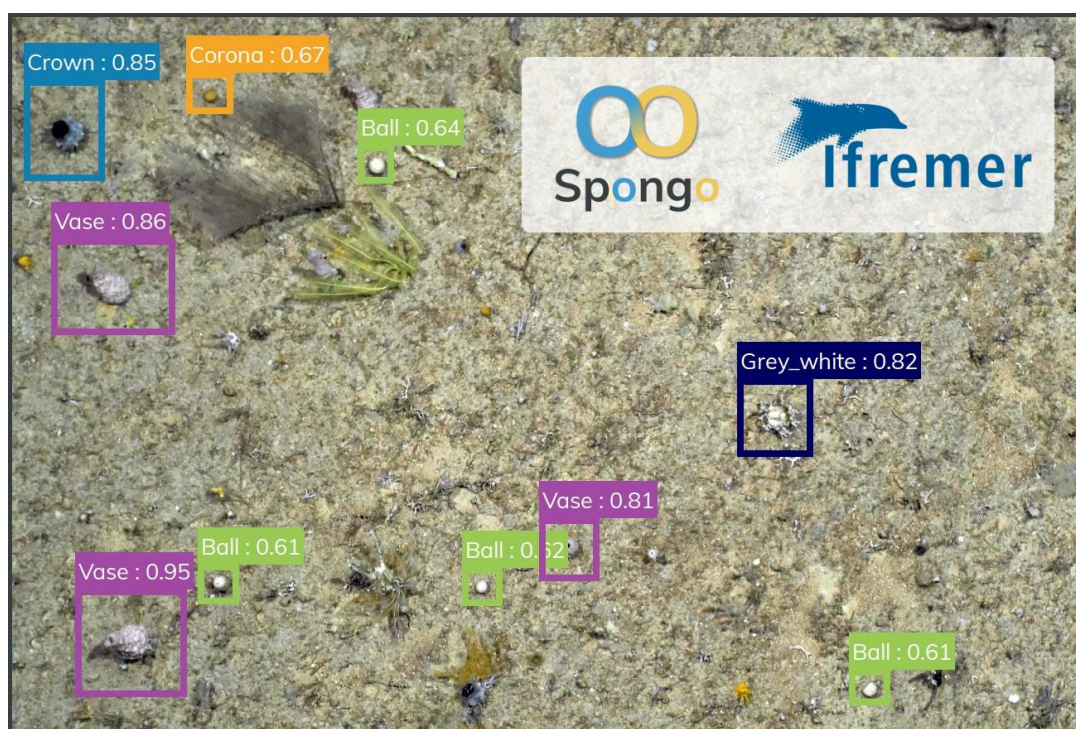


Classification automatique d'éponges marines profondes par imagerie



Proposé par : NAPOLÉON Thibault

Thématique : Image

DOUDET Margaux

Domaine professionnel : Génie logiciel

THOMAS Alexandre

Domaine professionnel : Génie logiciel

Résumé

Ce rapport traite de la reconnaissance et de la classification automatique de morphotypes d'éponges marines présentes sur des images de fonds marins.

Pour répondre à cette problématique, nous avons utilisé le transfer learning pour entraîner différents réseaux de neurones convolutifs et comparer leurs performances. Nous nous sommes tournés vers l'architecture YOLO (You Only Look Once), état de l'art actuel qui permet la classification d'objets sur des images en temps réel. Plus particulièrement, nous avons testé les versions les plus récentes de YOLO, à savoir YOLOv4 ainsi que différentes variantes de YOLOv5. Par la suite, nous avons expérimenté différents hyperparamètres pour ces réseaux, ainsi que différents pré-traitements sur les images d'entrée. L'évaluation de ces réseaux a ensuite permis de sélectionner le plus adapté pour la classification des morphotypes d'éponges.

Dans l'objectif de pouvoir lancer rapidement et simplement une analyse d'images de fonds marins, une interface homme-machine a été réalisée. Cette dernière a été réalisée en Python avec le framework PySide2 afin d'être la plus portable possible. L'implémentation du réseau de neurones dans l'application a été faite avec le framework PyTorch, permettant d'exploiter la technologie CUDA des cartes graphiques NVIDIA pour accélérer les calculs.

Termes clés : images, morphotypes d'éponges marines, deep learning, transfer learning, YOLOv4, YOLOv5, PySide2, PyTorch.

Abstract

This paper deals with the recognition and classification of sea sponges present on seabed images based on their morphotypes.

To address this problem, we used transfer learning to train different convolutional neural networks and compare their performances. We focussed on the YOLO (You Only Look Once) architecture, current state of art that allows objects classification on real-time images. More specifically, we evaluated the most recent versions of YOLO, namely YOLOv4 as well as different variants of YOLOv5. Afterwards, we experienced different parameters on these networks, as well as different pre-treatments on the input images. The evaluation of these networks allowed us to select the more appropriate one for the classification of sea sponges based on their morphotype.

In order to start quickly and easily an analysis of seabed images, a human-machine interface was created. It was done so using Python with the framework PySide2 so that it is the most portable possible. The implementation of the neural network into the application was achieved with the framework PyTorch, making it possible to exploit the CUDA technology of NVIDIA graphic cards, speeding the calculations.

Key words: images, seabed sponges morphotypes, deep learning, transfer learning, YOLOv4, YOLOv5, PySide2, PyTorch.

Remerciements

Nous souhaitons tout d'abord remercier notre encadrant M. Thibault NAPOLÉON qui a contribué au succès de ce projet et qui nous a aidé lors de la rédaction de ce rapport. Le temps qu'il nous a accordé ainsi que ses conseils ont été d'une grande aide pour mener à bien ce travail. De plus, les nombreux échanges que nous avons pu partager durant ces quelques semaines furent très enrichissants.

Nous souhaitons aussi remercier l'Ifremer, et plus particulièrement Karine Olu, Melissa Hanafi-Portier et Catherine Borremans pour le temps accordé et l'intérêt porté à notre travail. Leurs annotations des images réalisées en amont ont été d'une grande aide pour mener à bien ce projet.

Table des matières

1.	Glossaire.....	7
2.	Introduction	8
2.1.	Présentation de l'Ifremer et du LEP	8
2.2.	Description du problème	8
2.3.	Mise en œuvre	9
3.	Cahier des charges.....	10
3.1.	Analyse du besoin	10
3.1.1.	Définition du besoin	10
3.1.2.	Justification du produit	10
3.1.3.	Liste des fonctionnalités	11
3.2.	Déroulement du projet.....	13
3.2.1.	Ressources allouées	13
3.2.2.	Méthodologie de projet	13
3.2.3.	Outils utilisés	13
3.2.4.	Planning prévisionnel	14
4.	Données fournies.....	16
4.1.	Description des données	16
4.1.1.	Résumé	16
4.1.2.	Sélection des classes.....	16
4.2.	Difficultés liées aux annotations	18
4.2.1.	Annotations incomplètes	18
4.2.2.	Annotations carrées	18
4.2.3.	Taille des annotations	19
4.2.4.	Irrégularité des annotations	19
5.	Réseau de neurones	20
5.1.	Le machine learning	20
5.2.	Les réseaux de neurones convolutifs (CNN).....	20
5.2.1.	Présentation des CNN	20
5.2.2.	Fonctionnement d'un CNN pour la classification d'images.....	21
5.3.	La recherche d'un réseau.....	23
5.3.1.	Les algorithmes existants	23
5.3.2.	Pré-sélection des réseaux.....	24
5.4.	Expérimentations.....	25
5.4.1.	Protocole expérimental.....	25
5.4.2.	Phase 1 : Tests des réseaux.....	27
5.4.3.	Phase 2 : Optimisation du réseau.....	30
5.4.4.	Conclusion des expérimentations	32
6.	Interface Homme-Machine.....	33
6.1.	Introduction	33
6.1.1.	Objectifs	33
6.1.2.	Technologies utilisées	33
6.2.	Page « Paramètres »	34
6.2.1.	Entrée.....	34
6.2.2.	Seuil de détection	35
6.2.3.	Processeur.....	35
6.2.4.	Sélection des morphotypes	36
6.2.5.	Sauvegarde des images.....	36
6.3.	Page « Analyse »	37

6.3.1.	Explication de l'application.....	37
6.3.2.	Implémentation du réseau de neurones	38
6.3.3.	Calcul du score d'intérêt.....	38
6.4.	Page « Historique »	39
6.4.1.	Sauvegarde des analyses.....	39
6.4.2.	Fonctionnalités de la page	39
6.5.	Exportation des données.....	40
6.5.1.	Rapports résumés	40
6.5.2.	Rapports détaillés	41
6.5.3.	Annotations	41
6.6.	Autres fonctionnalités.....	42
6.6.1.	Confort d'utilisation lors de l'analyse	42
6.6.2.	À propos et licences	42
6.7.	Déploiement et distribution	43
6.7.1.	Déploiement	43
6.7.2.	Distribution.....	43
7.	Gestion de projet	44
7.1.	Approche agile	44
7.2.	Difficultés rencontrées	44
7.3.	Finalisation du projet.....	45
7.4.	Disponibilité du projet et licence.....	45
8.	Conclusion.....	46
8.1.	Retour sur le projet	46
8.2.	Perspectives possibles	46
9.	Bibliographie	47
10.	Annexes.....	48
10.1.	Comparaison de la maquette originale et de l'application finale	48
10.2.	Exemple de rapport résumé en PDF	50

Table des tableaux

Tableau 1 - Méthode "QQOQCP" de définition du besoin	10
Tableau 2.1 - Fonctionnalités à haute priorité	11
Tableau 2.2 - Fonctionnalités à priorité moyenne.....	11
Tableau 2.3 - Fonctionnalités à priorité basse	12
Tableau 3 - Légende du planning prévisionnel.....	14
Tableau 4.1 - Planning prévisionnel de la semaine 4 (25/01 - 29/01) à la semaine 12 (22/03 - 26/03)	14
Tableau 4.2 - Planning prévisionnel de la semaine 13 (29/03 - 02/04) à la semaine 17 (26/04 - 30/04)	15
Tableau 5 - Dates limites et objectifs à atteindre	15
Tableau 6 - Exemple de 3 annotations d'éponges provenant de la PL07.....	16
Tableau 7 - Liste des morphotypes avec leur nombre d'annotations respectifs (première version du fichier)	16
Tableau 8 - Liste des morphotypes avec leur nombre d'annotations respectifs (seconde version du fichier)	17
Tableau 9 - Mesures prises sur les différents réseaux testés	28
Tableau 10 - Mesures prises sur les réseaux YOLOv4 avec différentes tailles d'entrée.....	30
Tableau 11 - Mesures prises sur les réseaux YOLOv4 avec et sans prédécoupage des images.....	31
Tableau 12 - Mesures prises sur les réseaux YOLOv4 après entraînement avec différentes tailles d'annotations.....	32

Table des illustrations

Figure 1 - Diagramme "Bête à cornes" du système désiré	10
Figure 2 - Exemple de chaque morphotype retenu	17
Figure 3 - Diagramme à barres du nombre d'annotations par morphotypes	17
Figure 4 - Exemple d'une annotation circulaire convertie en carré	18
Figure 5 - Comparaison annotation carrée et annotation rectangulaire sur une éponge Vase.....	18
Figure 6 - Histogramme de la répartition des tailles des annotations par rapport à la largeur de l'image.....	19
Figure 7 - Schéma de l'IA, du machine learning et du deep learning	20
Figure 8 - Schéma d'un réseau de neurones convolutif.....	21
Figure 9 - Schéma d'une carte de caractéristiques et d'un filtre convolutif	21
Figure 10 - Schéma d'une convolution d'une carte de caractéristiques d'entrée	21
Figure 11 - Schéma du pooling maximal sur une carte de caractéristiques d'entrée.....	22
Figure 12 - Schéma des couches flattening, fully connected et softmax.....	22
Figure 13 - Formules de la precision et du recall.....	25
Figure 14 - Exemple de matrice de confusion.....	26
Figure 15 - Évolution de la mAP durant l'apprentissage des réseaux YOLOv4 (a), YOLOv5 small (b), YOLOv5 medium (c) et YOLOv5 large (d)	27
Figure 16 - Matrices de confusions obtenues sur les réseaux YOLOv4 (a), YOLOv5 small (b), YOLOv5 medium (c), et YOLOv5 large (d)	28
Figure 17 - Matrices de confusions obtenues sur les réseaux YOLOv4 avec une entrée de 416x416px (a), 512x512px (b), et 832x704px (c).....	30
Figure 18 - Matrices de confusion obtenues sur les réseaux YOLOv4 sans prédécoupage des images (a) et avec un prédécoupage en 4 images (b).....	31
Figure 19 - Matrices de confusion obtenues sur les réseaux YOLOv4 après entraînement sur les annotations fournies (a), augmentées de 1,5 fois leur diamètre (b), et augmentées de 2 fois leur diamètre (c)	32
Figure 20 - Logo créé pour l'application.....	33
Figure 21 - Aperçu de la page "Paramètres".....	34
Figure 22 - Choix du processeur sur un PC ne disposant pas (a) et disposant d'un GPU compatible (b) CUDA	35
Figure 23 - Exemple d'une image obtenue lorsque la sauvegarde des images est activée	36
Figure 24 - Aperçu de la page "Analyse".....	37
Figure 25 - Formule du score d'intéressement	38
Figure 26 - Aperçu de la page "Historique".....	39
Figure 27 - Aperçu de la page "Exportation".....	40
Figure 28 - Icône de l'application lors d'une analyse.....	42
Figure 29 - Aperçu de la fenêtre de validation d'interruption de l'analyse en cours.....	42
Figure 30 - Fenêtre « À propos » de l'application avec sa présentation (a) et la liste des dépendances (b)	42
Figure 31 - Aperçu de l'installateur de l'application.....	43

1. Glossaire

Annotation : Légende assignée à un objet sur une partie d'une image.

Apprentissage automatique (trad. *Machine learning*) : Étude d'algorithmes informatiques qui s'améliorent automatiquement grâce à l'expérience et à l'utilisation de données.

Apprentissage profond (trad. *Deep learning*) : Ensemble de méthodes d'apprentissage automatique tentant de modéliser des données avec un haut niveau d'abstraction.

Average precision : Mesure permettant d'évaluer la performance d'un algorithme sur la classification d'une classe d'objets précis.

Bibliothèque (trad. *Library*) : Ensemble de fonctions déjà codées et utilisables par des programmes.

Carte de caractéristiques (trad. *Feature map*) : Matrice résultante d'une couche convolutive d'un réseau de neurones.

CUDA (Compute Unified Device Architecture) : Technologie de NVIDIA permettant d'exécuter certains calculs sur la carte graphique plutôt que sur le processeur.

Epoch : Cycle d'apprentissage d'un réseau de neurones pendant lequel la totalité des images est passée une fois dans le réseau.

Framework : Ensemble de composants logiciels servant à créer des logiciels. Un framework a généralement une portée plus large qu'une bibliothèque.

Interface Homme-Machine (IHM) : Moyen mis en place pour permettre la communication entre un humain et une machine. Il peut s'agir par exemple, d'une interface graphique.

Jeu de données (trad. *Dataset*) : Ensemble de valeurs (ou données) où chaque valeur est associée à une ou plusieurs variables.

Jeu de test (trad. *Test set*) : Dans l'apprentissage automatique, il s'agit d'un jeu de données utilisé pour évaluer de manière non biaisée les performances d'un modèle.

Jeu d'entraînement (trad. *Training set*) : Dans l'apprentissage automatique, il s'agit d'un jeu de données utilisé pour entraîner le modèle.

mAP (mean Average Precision) : Mesure permettant d'évaluer notamment un algorithme dédié à la classification d'objet. Sa valeur est comprise entre 0 et 1 et qui correspond à la moyenne de *l'average precision*.

Matrice de confusion : Matrice permettant de mesurer la qualité d'un système de classification.

Morphotype : critère physique selon lequel une espèce est classifiée.

Plongée : expédition sous-marine durant laquelle ont été prises des images de fonds marins.

Precision : Dans un réseau de neurones, taux de bonnes détections sur le nombre total d'éponges détectées.

Recall : Dans un réseau de neurones, taux de bonnes détections sur le nombre total d'éléments présents.

ROV (Remotely Operated underwater Vehicle) : véhicule sous-marin téléguidé utilisé par l'Ifremer pour prendre des images de fonds marins lors d'une plongée.

Réseau de neurones : Système informatique utilisé en machine learning et inspiré du fonctionnement des neurones biologiques.

Réseau de neurones convolutif : Modèle de réseau de neurones particulièrement utilisé dans l'analyse d'images.

Transfer learning : Technique permettant d'exploiter les connaissances acquises sur des tâches sources pour mieux traiter des tâches cibles.

YOLO (You Only Look Once) : Architecture de réseau de neurones convolutif profond dédié à la reconnaissance d'objets en temps réel.

2. Introduction

2.1. Présentation de l'Ifremer et du LEP

L'institut français de recherche pour l'exploitation de la mer (Ifremer) est reconnu dans le monde entier comme l'un des tout premiers instituts en sciences et technologies marine. L'Ifremer s'inscrit dans une perspective à la fois de développement durable et de sciences ouvertes. Les recherches menées ont pour but de créer des innovations afin de protéger et restaurer l'océan, ainsi que d'exploiter ses ressources de manière responsable.

Fondé en 1984, l'institut est un établissement public à caractère industriel et commercial. Il est placé sous la tutelle conjointe des ministères de l'Enseignement supérieur, de la Recherche et de l'Innovation, de la Transition écologique et solidaire, de l'Agriculture et de l'Alimentation.

Au sein de l'Ifremer, le laboratoire environnement profond (LEP) est une entité dédiée à l'étude des écosystèmes marins profonds. Le laboratoire est composé d'une équipe de 21 personnes aux métiers divers : chercheurs, ingénieurs et techniciens. L'objectif principal du LEP est de mieux connaître et caractériser la biodiversité marine présente sur le plancher océanique en fonction de différents contextes (géologiques, hydrodynamiques et trophiques) afin de favoriser une exploitation raisonnée des ressources.

Source : <https://www.ifremer.fr/L-institut>

2.2. Description du besoin

La cartographie des fonds marins et de leur écosystème a augmenté de façon exponentielle avec les capacités d'exploration des engins sous-marins et les progrès réalisés en imagerie optique. Les ROVs (*Remotely Operated underwater Vehicule*) sont en effet des outils essentiels pour caractériser de manière fiable et non destructrice les habitats marins et leur biodiversité.

La nécessité d'inventorier le plus largement possible les habitats marins (notamment les plus vulnérables), se traduit par une augmentation considérable des quantités de données à analyser. Les méthodes de traitement manuel d'image étant très chronophages, des moyens d'automatiser ces traitements sont de plus en plus recherchés. Quelques tentatives d'analyse d'images de fonds marins ont été menées en interne par l'Ifremer, mais celles-ci restent ponctuelles et peu adaptées aux images complexes. Cependant, les progrès considérables de l'intelligence artificielle et de l'apprentissage automatique peuvent à présent permettre l'analyse des images acquises.

À la suite de différentes études, le laboratoire environnement profond de l'Ifremer a remarqué une bonne corrélation entre la nature du substrat, les espèces d'éponges marines présentes et les populations d'autres espèces (comme certains poissons) sur les monts marins. Ainsi, connaître les populations d'éponges, et les zones où elles sont présentes pourrait permettre de mieux cartographier la biodiversité des fonds marins et apporter de nombreuses informations sur les espèces qui y vivent.

L'objectif du projet est donc de développer un outil permettant de détecter et de classifier automatiquement les éponges présentes sur les milliers d'images prises par les ROVs lors des plongées. La reconnaissance d'espèces précises étant trop complexe, l'outil s'attachera plutôt à détecter des « formes » d'éponges, appelées morphotypes.

Pour cela, l'Ifremer a déjà annoté 2445 éponges sur 485 images différentes avec le logiciel BIIGLE. Au total, vingt morphotypes ont été recensés sur ces images, mais on ne s'attachera dans un premier temps qu'à la détection de six d'entre eux.

2.3. Mise en œuvre

Les données fournies consistaient en un jeu d'images de fonds marins associé à un fichier d'annotations, listant les différentes éponges présentes, leur morphotype ainsi que leurs coordonnées. La première étape du projet a donc été de trier les différentes données fournies par l'Ifremer. Ce premier traitement est essentiel à la fois pour éliminer les annotations non exploitables, mais aussi pour chercher la méthode la plus adaptée pour la reconnaissance des morphotypes.

Une fois les annotations traitées, le cœur du projet a consisté en la recherche et l'expérimentation de différents réseaux de neurones, et plus particulièrement de réseaux de neurones dits « convolutifs », adaptés à la reconnaissance d'images. Différents réseaux avec des architectures et des paramètres divers ont ainsi été entraînés à classifier les morphotypes d'éponges. La comparaison des performances des différents réseaux expérimentés a permis de sélectionner celui qui proposait les meilleurs résultats.

Enfin, la dernière étape a consisté en la réalisation d'une interface homme-machine (IHM) afin de proposer une application claire et ergonomique pour les biologistes de l'Ifremer. Cette application devait donc implémenter le réseau de neurones entraîné auparavant pour pouvoir lancer une analyse sur un dossier d'images de fonds marins. De plus, l'interface devait aussi permettre de générer des rapports compilant les informations collectées durant une analyse, afin qu'ils puissent être exploités par les chercheurs de l'Ifremer.

Chacune des étapes ci-dessus, c'est-à-dire le traitement des données, les expérimentations des réseaux de neurones, la réalisation de l'interface ainsi que l'exportation des données est détaillée dans ce rapport.

3. Cahier des charges

3.1. Analyse du besoin

3.1.1. Définition du besoin

On cherche à définir le besoin à travers différentes questions afin de proposer une solution adaptée (voir Tableau 1).

Questions	Réponses
Quoi ?	Un logiciel permettant la détection et la classification automatique d'éponges provenant d'images de fonds marins.
Qui ?	Le logiciel doit être utilisable par les employés de l'Ifremer.
Où ?	Le logiciel doit être utilisable sur les ordinateurs de l'Ifremer.
Quand ?	Le logiciel doit mettre au plus 3 secondes pour traiter chaque image afin de pouvoir analyser l'ensemble des images en un temps raisonnable.
Comment ?	Le logiciel doit être utilisable à travers une interface graphique claire et ergonomique.
Pourquoi ?	Le logiciel a pour objectif d'automatiser le travail des chercheurs de l'Ifremer afin d'améliorer leurs connaissances concernant la biodiversité marine.

Tableau 1 - Méthode "QOOQCP" de définition du besoin

3.1.2. Justification du produit

Le diagramme « Bête à cornes » (Figure 1) permet d'exprimer la recherche du besoin :

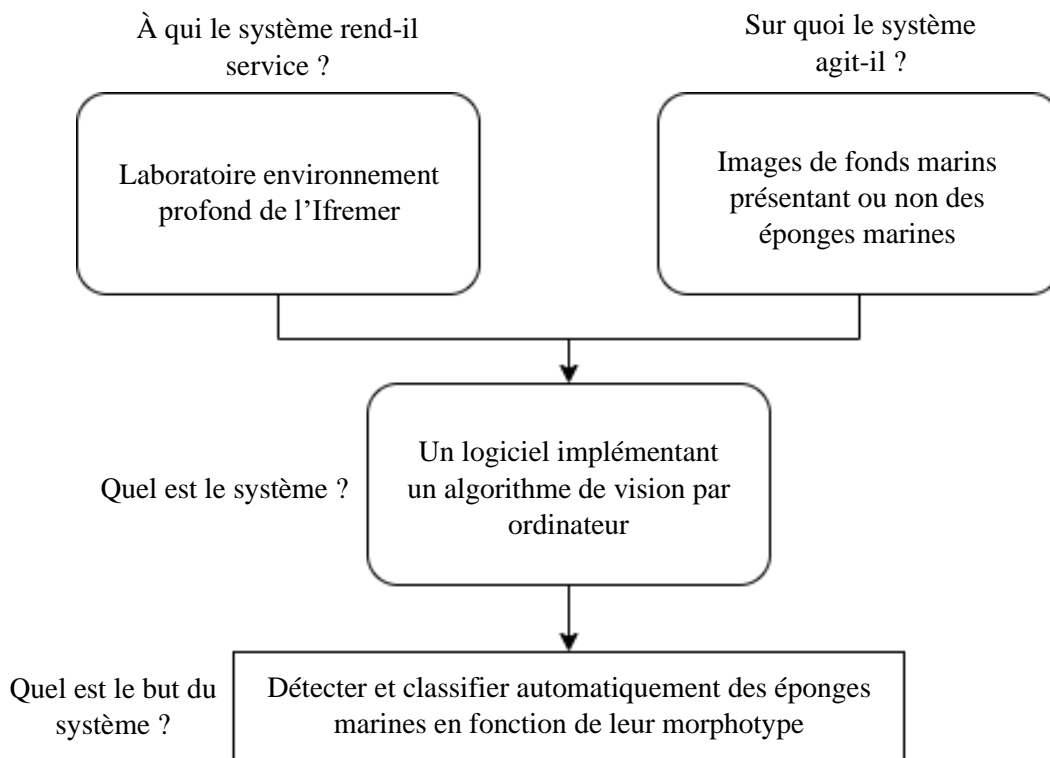


Figure 1 - Diagramme "Bête à cornes" du système désiré

3.1.3. Liste des fonctionnalités

Les fonctionnalités que doit proposer le logiciel ont été listées dans les tableaux ci-dessous en fonction de leur priorité et de leur catégorie.

Le tableau suivant (Tableau 2.1) présente les fonctionnalités à haute priorité, elles constituent le cœur du projet : la détection et la classification de morphotypes d'éponges

Catégorie	Fonctionnalités
Classification	Détecter la présence d'éponges marines sur un dossier d'images de fonds marins fournies en entrée du logiciel.
	Classifier les morphotypes d'éponges demandées les plus reconnaissables : <ul style="list-style-type: none">- Porifera_vase- Porifera_crown- Red
Utilisation	Dans un premier temps, pouvoir lancer l'analyse des images depuis une commande dans le terminal.
Sortie	Obtenir un bilan de l'analyse dans un fichier au format texte listant le nombre d'éponges détectées de chaque morphotype.

Tableau 2.1 - Fonctionnalités à haute priorité

Le tableau ci-dessous (Tableau 2.2) liste les fonctionnalités à priorité moyenne. Elles constituent des améliorations substantielles pour rendre l'outil utilisable par l'Ifremer.

Catégorie	Fonctionnalités
Classification	Classifier tous les morphotypes d'éponges demandés : <ul style="list-style-type: none">- Porifera_vase- Porifera_crown- Porifera_ball- Yellow- Grey- Red
Interface	Disposer d'une Interface Homme-Machine (IHM) pour lancer l'analyse des images.
	Laisser la possibilité de régler les paramètres d'analyse (chemin du dossier d'images à analyser, taux de confiance pour la détection, ...) directement depuis l'interface.
	Afficher la progression de l'analyse dans l'interface.
	A la fin de l'analyse, l'IHM doit afficher le nombre d'éponges détectées de chaque morphotype.
Utilisation	Proposer un protocole d'installation avec peu de manipulations (quelques commandes à entrer dans un terminal).
Sortie	Générer un fichier CSV similaire à celui des annotations listant les éponges détectées, leur morphotype, l'image dans laquelle elles ont été détectées ainsi que leurs coordonnées sur l'image.

Tableau 2.2 - Fonctionnalités à priorité moyenne

Enfin, ce dernier tableau (Tableau 2.3) propose des fonctionnalités liées à l'amélioration non essentielle des fonctionnalités principales ou à l'aspect esthétique du logiciel. Ces dernières seront réalisées en fonction de l'avancée dans le projet.

Catégorie	Fonctionnalités
<i>Classification</i>	Classifie d'autres morphotypes non demandés, mais avec suffisamment d'annotations comme Porifera_corona par exemple.
<i>Interface</i>	Pendant l'analyse, afficher les résultats préliminaires en direct.
	Pendant et à la fin de l'analyse, afficher un graphique avec les morphotypes détectés au cours du temps.
	Dans le résumé de l'analyse, afficher des images d'intérêts. Il peut s'agir des images présentant le plus d'éponges, ou celles avec le plus de morphotypes différents.
	Avoir un historique des analyses réalisées avec les informations recueillies sur chacune d'elle.
	Avoir une interface claire et intuitive.
<i>Utilisation</i>	Proposer un protocole d'installation très simple, avec un seul fichier à exécuter ou un installateur.
<i>Sortie</i>	Permettre d'exporter un résumé d'une analyse sous la forme d'un rapport au format PDF.
	Permettre de récupérer les résultats détaillés d'une analyse dans d'autres formats que le CSV, comme le JSON ou le XML.
	Laisser la possibilité d'enregistrer les images avec les boîtes de détections autour des éponges.

Tableau 2.3 - Fonctionnalités à priorité basse

De plus, on cherchera en perspective à ce que l'outil développé soit le plus évolutif possible afin de pouvoir être réutilisé pour d'autres groupes taxonomiques, comme des coraux par exemple.

3.2. Déroulement du projet

3.2.1. Ressources allouées

Ce projet sera réalisé par deux étudiants en Master 1 de l'ISEN Brest pendant les 9 semaines allouées. En plus de ce temps réservé à la réalisation du projet, celui-ci pourra être avancé sur le temps personnel, pendant les semaines de cours ou la semaine de vacances d'hiver. Le projet sera rendu, au plus tard en fin de semaine 16, c'est-à-dire le vendredi 23 avril 2021.

Le projet ne nécessitant pas de matériel à acheter, celui-ci sera réalisé sur les ordinateurs personnels des étudiants. Pour les longs calculs, il sera aussi possible d'utiliser les plateformes cloud, comme Google Colaboratory qui permet d'exécuter du code python gratuitement sur des serveurs distants. Enfin, au besoin, un accès au serveur graphique de l'ISEN pourra être octroyé temporairement.

3.2.2. Méthodologie de projet

La méthodologie Agile sera utilisée pour la gestion de ce projet, celle-ci possédant de nombreux avantages. Tout d'abord, la flexibilité de l'approche Agile permet de s'adapter aux imprévus et aux éventuels changements. Cette flexibilité est d'autant plus importante que des algorithmes utilisant l'apprentissage automatique pourront être utilisés, et dont la qualité des résultats est difficilement anticipable. De plus, cela permettra aussi aux clients (le tuteur de projet et l'Ifremer) de constater l'avancée du projet et ainsi de pouvoir ajuster les demandes en fonction des besoins.

Afin d'avoir cette approche Agile, la périodicité des réunions durant les semaines de projet a été fixée comme suit :

- Avec le tuteur de projet : toutes les semaines
- Avec l'Ifremer : toutes les deux à trois semaines en fonction de l'avancée

En dehors des semaines de projets, les réunions seront fixées en fonction de l'avancement sur le temps personnel.

3.2.3. Outils utilisés

La communication entre étudiants se fera par Discord, outil déjà maîtrisé et adapté aux conversations informelles. De plus, les échanges avec le tuteur se feront via Teams, et ceux avec l'Ifremer par courriel.

La gestion de projet se fera elle aussi sur Teams afin de pouvoir réunir au même endroit tous les documents liés au projet. En outre, Teams propose de nombreux outils utiles comme Tasks pour lister et organiser les tâches à réaliser, ou Wiki pour prendre des notes rapidement.

Enfin le code sera versionné avec l'outil git et hébergé sur le service GitHub. Ces outils permettront de revenir rapidement à une version fonctionnelle du projet en cas de problème, mais aussi à pallier une éventuelle panne de la machine hébergeant le code. Dans un objectif de clarté, deux dépôts git seront créés :

- Un dépôt hébergeant les scripts utilisés pour effectuer des traitements sur les données fournies (lister les annotations, récupérer les images contenant des annotations, ...)
- Un dépôt hébergeant le code de l'interface

3.2.4. Planning prévisionnel

Le planning s'étend sur 14 semaines, du début du projet aux évaluations. Ce dernier est présenté ci-après (Tableau 4.1 et Tableau 4.2). La légende est donnée ci-dessous (Tableau 3).

S4, S10-S16	Semaines dédiées au projet		Tâche commune
S5-S9	Semaines de cours et vacances		Tâche Alexandre
S17	Semaine d'évaluation		Tâche Margaux

Tableau 3 - Légende du planning prévisionnel

Tâche	S4	S5-S9	S10	S11	S12
Recherches et expérimentations préliminaires					
Récupération et premier tri des données fournies					
Premières recherches sur les réseaux de neurones et leur application pour la détection d'objet					
Extraction de quelques éponges de chaque morphotype parmi les annotations					
Création de divers scripts pour automatiser des traitements sur les annotations					
Expérimentation sur un réseau de neurones					
Algorithme de classification des morphotypes					
Recherches sur différents réseaux de neurones dédiés à la classification d'objet					
Rédaction d'un protocole de test pour comparer les différents réseaux de neurones trouvés					
Comparaison de différents réseaux de neurones					
Expérimentation de différents paramètres sur le réseau de neurones choisi					
Interface					
Réalisation d'une première interface permettant de lancer une analyse sur un dossier d'images					
Ajout d'une barre de progression pendant l'analyse					
Ajout d'une page "Paramètres" permettant de régler les paramètres d'analyses					
Affichage d'un résumé de l'analyse une fois celle-ci terminée					
Affichage d'un graphique des morphotypes détectés durant l'analyse en fonction du temps					
Ajout d'un historique des analyses réalisées					
Sortie					
Génération d'un fichier texte résumant l'analyse					
Génération d'un fichier CSV listant toutes les détections					
Gestion de projet					
Rédaction et validation du cahier des charges					

Tableau 4.1 - Planning prévisionnel de la semaine 4 (25/01 - 29/01) à la semaine 12 (22/03 - 26/03)

Tâche	S13	S14	S15	S16	S17
Interface					
Préparation et application d'une charte graphique à l'interface					
Sortie					
Permettre d'exporter un résumé de l'analyse sous la forme d'un rapport au format PDF					
Permettre de récupérer les résultats d'une analyse dans des formats comme JSON ou XSMML					
Permettre d'enregistrer les images avec les boites de détection					
Utilisation					
Préparation d'un protocole d'installation simple					
Préparation d'un fichier exécutable ou d'un installateur					
Intégration					
Phase d'intégration					
Phase de test					
Gestion de projet					
Rédaction du document destiné aux utilisateurs de l'outil					
Rédaction du rapport de projet					
Préparation aux oraux de présentation					

Tableau 4.2 - Planning prévisionnel de la semaine 13 (29/03 – 02/03) à la semaine 17 (26/04 - 30/04)

Les lignes rouges correspondent à des dates limites pour lesquelles nous avons fixé des objectifs. Ces dernières permettront, si nécessaire, d'allouer plus de temps à des fonctionnalités jugées comme prioritaires. Les 3 dates limites que nous nous sommes fixées sont les suivantes (Tableau 5).

Date	Objectif
Vendredi 05/03 (fin de semaine 9)	Avoir une « version 0 » montrant que les technologies choisies fonctionnent et permettent d'obtenir des premiers résultats.
Vendredi 02/04 (fin de semaine 13)	Atteindre une version fonctionnelle implémentant, au minimum, toutes les fonctionnalités de priorité « haute » et « moyenne ».
Vendredi 23/04 (fin de semaine 16)	Délivrer la version finale du projet avec toutes les fonctionnalités envisagées, et avoir terminé le rapport de projet.

Tableau 5 - Dates limites et objectifs à atteindre

4. Données fournies

4.1. Description des données

4.1.1. Résumé

Au début du projet, l'Ifremer nous a fourni un disque dur contenant deux dossiers d'images de fonds marins correspondant chacun à une plongée complète. Le premier dossier, nommé PL05, est composé d'environ 15 000 images non annotées. Le second dossier, nommé PL07, est lui composé d'environ 10 000 images, dont 500 annotées. Au total, cela représente un peu moins de 2500 éponges annotées pour entraîner le réseau de neurones. Les images sont toutes au format JPEG et ont une taille fixe de 4243x2828 pixels, qui correspond à la résolution de la caméra du ROV. Les annotations sont fournies dans un fichier à part, au format CSV. Le Tableau 6 ci-dessous montre des exemples d'annotations (les colonnes superflues ont été retirées pour plus de lisibilité).

label_name	filename	shape_name	points
Porifera_vase	20190916T133642.097511Z.jpg	Circle	[3884.58,1628.44,75.33]
red	20190917T073045.347189Z.jpg	Circle	[2850.77,2459.22,96.64]
Porifera_corona	20190916T200731.059006Z.jpg	Circle	[37.29,1108.42,97.4]

Tableau 6 - Exemple de 3 annotations d'éponges provenant de la PL07

Chaque éponge est décrite avec les informations nécessaires à son identification : son morphotype (*label_name*), le nom de l'image sur laquelle elle se trouve (*filename*), la forme de son annotation (*shape_name*) et ses coordonnées (*points*). Toutes les annotations sont de formes circulaires, et ses coordonnées sont de la forme [x, y, D] avec x, y les coordonnées du centre du cercle par rapport au coin supérieur gauche de l'image et D le diamètre en pixels.

4.1.2. Sélection des classes

La première étape pour nous a été de traiter le fichier d'annotations afin de déterminer le nombre d'annotations pour chaque morphotype. Le Tableau 7 indique les classes présentes dans le fichier CSV avec leur quantité d'annotations respectives. Cette lecture du fichier, ainsi que tous les traitements des annotations qui suivent ont été effectués grâce à des scripts en Python.

Morphotypes	Nb	Morphotypes	Nb	Morphotypes	Nb
Grey	285	Thin	154	Cnidaria	35
Porifera_ball	243	Thick	117	Porifera_weird	34
Porifera_vase	239	Pheronema_sp	105	Cavity	24
Porifera_amorphous	228	Yellow	103	Asteroida	2
Porifera_corona	226	Porifera_vase2	101	Chordata	1
Porifera_crown	220	Porifera	57	Crustacea	1
Red	217	Porifera_amorphous+cavity	53		

Tableau 7 - Liste des morphotypes avec leur nombre d'annotations respectifs (première version du fichier)

Cependant, beaucoup de ces classes n'étaient pas exploitables. Dans un premier temps, nous avons donc fait le choix d'éliminer les classes avec moins de 200 annotations. Cela permettait ainsi d'éliminer les morphotypes pour lesquels le réseau risquait d'avoir le plus de mal à apprendre.

À la suite de la première réunion avec l'Ifremer, les chercheurs nous ont renvoyé un nouveau fichier CSV. Ce dernier comprenait les mêmes annotations, mais avec plusieurs modifications sur les morphotypes. Tout d'abord, le préfixe « Porifera » (qui signifie « éponge ») a été supprimé des noms. Ensuite, certains morphotypes moins utiles ont simplement été supprimés. Enfin, certaines classes ont été fusionnées (par exemple « Porifera_vase » et « Porifera_vase2 » furent réunis en « Vase »).

Les morphotypes de cette nouvelle version sont listés dans le Tableau 8 ci-dessous.

Morphotypes	Nb	Morphotypes	Nb
Vase	338	Crown	220
Encrusted_yellow	248	Red	217
Ball	243	Grey_white	163
Corona	226	Yellow	109

Tableau 8 - Liste des morphotypes avec leur nombre d'annotations respectifs (seconde version du fichier)

À l'origine, l'Ifremer souhaitait détecter les morphotypes Vase, Ball, Crown, Red, Grey_white et Yellow. Cependant, le faible nombre d'annotations pour la classe Yellow ne permettait pas d'entraîner le réseau de neurones correctement, nous l'avons donc retiré.

Nous avons aussi exclu la classe Encrusted_yellow (qui est différent de « Yellow »). En effet, même si ce morphotype a un nombre d'annotations intéressant, les éponges annotées avaient des formes complexes. Il s'agissait d'éponges très petites, légèrement jaunes et très difficilement distinguables du fond marin. Cette classe n'étant pas demandée par l'Ifremer, nous avons préféré la mettre de côté.

En revanche, la classe Corona présentait, elle, une forme et une couleur bien reconnaissable (une boule jaune) ainsi qu'un nombre d'annotations intéressant. Nous avons donc conservé ce morphotype.

Finalement, nous avons donc sélectionné six morphotypes : **Grey_white**, **Ball**, **Vase**, **Corona**, **Crown** et **Red**. Ceux-ci sont présentés sur la Figure 2.



Figure 2 - Exemple de chaque morphotype retenu

La Figure 3 ci-dessous décrit la présence de chaque morphotype sélectionné dans le jeu de données fourni. Pour des raisons expliquées dans la partie 4.2.1, des annotations ont été rajoutées par nos soins ce qui explique les variations dans le nombre d'annotations par rapport au Tableau 8.

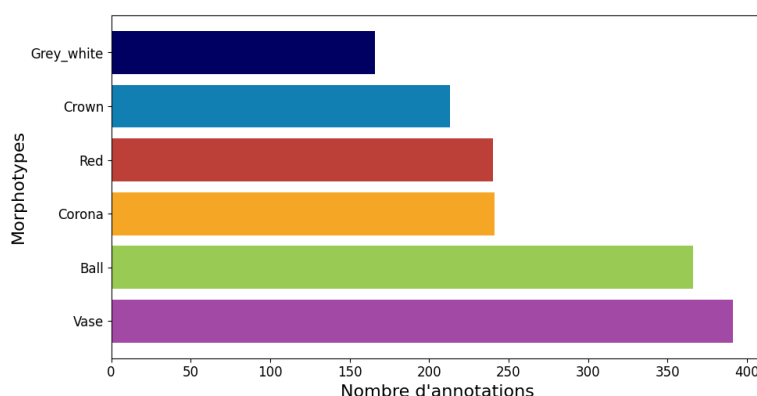


Figure 3 - Diagramme à barres du nombre d'annotations par morphotypes

On remarque que le nombre d'annotations n'est pas homogène entre tous les morphotypes. Certains, comme « Vase » sont sur-représentés avec plus de deux fois plus d'annotations que « Grey_white ». Au total, nous disposons d'environ 1600 annotations sur lesquelles entraîner le réseau de neurones.

4.2. Difficultés liées aux annotations

4.2.1. Annotations incomplètes

Le problème le plus important auquel nous avons eu affaire est que sur une image, il arrivait que toutes les éponges présentes ne soient pas annotées. Cela fut un problème majeur, car lors de l'apprentissage, un réseau qui détecte correctement une éponge non annotée va considérer qu'il s'agit d'une erreur. Il va donc tenter de se corriger alors que sa prédiction était bonne. Lorsqu'il y a trop d'éponges non annotées, les performances du réseau se retrouvent alors grandement impactées.

Pour remédier à ce problème, nous avons décidé de compléter les annotations nous-mêmes à la main. Cela fut faisable pour les morphotypes les plus reconnaissables, comme Ball ou Vase. Cependant, n'étant pas biologistes, nous avons préféré ne pas rajouter d'annotations pour Grey_white qui est beaucoup plus difficile à identifier. Nos corrections ont permis d'améliorer les résultats notamment sur les morphotypes les moins bien annotés.

4.2.2. Annotations carrées

Comme expliqué auparavant, les annotations fournies étaient toutes de forme circulaire. Cependant les réseaux de neurones dédiés à la reconnaissance d'objets prennent généralement en entrée des annotations rectangulaires. La première étape pour nous a donc été de convertir les annotations circulaires en annotation carrées. Pour cela, nous avons simplement pris le carré tel que le cercle de l'annotation soit inscrit à l'intérieur de ce dernier (voir Figure 4).



Figure 4 - Exemple d'une annotation circulaire convertie en carré

La plupart des éponges étant relativement rondes, avoir une annotation en carré ne pose pas de problèmes. Cependant, pour certains morphotypes comme Vase, une annotation en rectangle aurait permis d'être plus précis. Comme le montre la Figure 5, l'annotation en carré impose des marges importantes autour de l'éponge lorsque cette dernière n'est pas carrée. Cette imprécision peut nuire à l'apprentissage, le réseau ayant plus de difficulté à discriminer l'élément d'intérêt dans l'annotation. Néanmoins, nos expérimentations ont montré que le nombre important d'annotations pour le morphotype Vase a permis de largement compenser cette difficulté.



Figure 5 - Comparaison annotation carrée et annotation rectangulaire sur une éponge Vase

4.2.3. Taille des annotations

La taille des annotations est un élément important à prendre en compte pour l'apprentissage du réseau de neurones. L'histogramme présenté en Figure 6 représente la distribution des tailles d'annotations par rapport à la largeur de l'image.

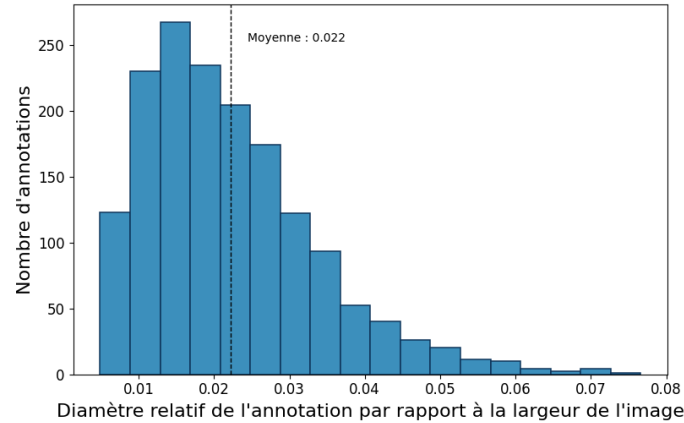


Figure 6 - Histogramme de la répartition des tailles des annotations par rapport à la largeur de l'image

On constate alors que toutes les annotations font moins de 8% de la largeur de l'image (soit 340 pixels). La majorité d'entre elles ont même une largeur relative comprise entre 0.5% et 3%, soit entre 20 et 130 pixels. Cela signifie que les éponges sont pour la plupart petites par rapport à la taille importante des images (4243x2828 pixels). Il s'agit d'un élément important à prendre en compte pour l'apprentissage du réseau de neurones. En effet, ces derniers ont généralement moins de difficultés à reconnaître les éléments relativement gros, ce qui n'est pas le cas des annotations d'éponges.

4.2.4. Irrégularité des annotations

Un autre problème que nous avons rencontré est lié à l'irrégularité de la taille des annotations. Dans certains cas, l'annotation faisait exactement la taille de l'éponge alors que dans d'autres cas elle était deux fois plus grande. Les conséquences sont les mêmes que pour les annotations carrées : cela signifie des marges importantes autour de l'éponge et donc des difficultés supplémentaires pour le réseau à apprendre et reconnaître le morphotype de l'éponge étudiée.

De la même manière, certaines annotations n'étaient pas parfaitement placées sur leur éponge. Certaines, éponges dépassaient de la zone annotée, avec parfois plus de la moitié de l'éponge en dehors de la zone. Cela est problématique pour l'apprentissage d'un réseau de neurones, en effet il peut avoir des difficultés à distinguer quel est l'élément important à reconnaître.

Nous avons donc modifié les quelques annotations pour lesquelles la taille ou le placement ne correspondait le moins à l'éponge. Cependant, nous ne sommes pas repassés sur toutes les annotations avec un léger défaut. Le temps nécessaire aurait en effet été trop important.

5. Réseau de neurones

5.1. Le machine learning

L'apprentissage machine (en anglais : *machine learning*), est « l'étude d'algorithmes informatiques qui s'améliorent automatiquement grâce à l'expérience et à l'utilisation de données » [1]. Ce champ d'études est une branche de l'intelligence artificielle et permet à des ordinateurs d'effectuer des tâches sans y être explicitement programmés [2].

L'apprentissage machine se divise généralement en deux phases. Dans un premier temps, on présente à l'algorithme de nombreuses données structurées. En utilisant ces données, et grâce aux retours effectués par un humain, le système va alors « comprendre » comment organiser et traiter les données similaires. C'est la phase d'apprentissage. Dans un second temps, une fois l'algorithme entraîné, celui-ci peut être utilisé sur de nouvelles données pour automatiser certaines tâches [2].

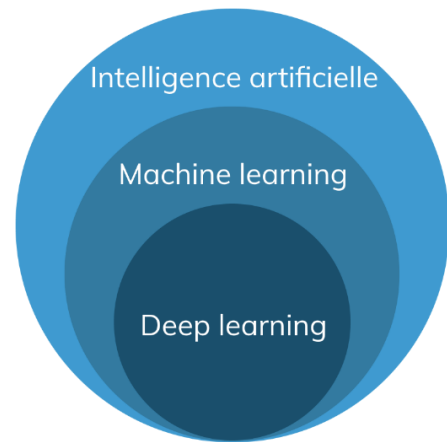


Figure 7 - Schéma de l'IA, du machine learning et du deep learning

Il existe de nombreuses méthodes utilisant le machine learning, mais l'une des plus en vogue ces dernières années est l'apprentissage profond (en anglais : *deep learning*). Le système est alors constitué d'un empilement de couches (un réseau) de neurones artificiels. Le fonctionnement de ces derniers est inspiré des neurones biologiques que l'on retrouve chez les animaux. Chaque neurone est une unité simple qui ne fait qu'émettre ou non un signal en fonction des signaux des neurones précédents. Cependant, la multiplication des couches permet d'extraire progressivement les caractéristiques de données brutes. L'intérêt d'un tel algorithme est alors considérable : pendant l'entraînement le système apprend seul à déterminer quelles caractéristiques de la donnée sont intéressantes pour son traitement [3]. La Figure 7 représente l'emboîtement des champs d'études englobant le Deep Learning.

Aujourd'hui, l'apprentissage profond est utilisé dans de nombreux domaines : vision par ordinateur, traitement du langage naturel, filtrage sur les réseaux sociaux, bio-informatique, etc. Néanmoins, le recours au Deep Learning à un coût : des milliers, voire des millions d'exemples annotés sont nécessaires pour entraîner le réseau de neurones.

5.2. Les réseaux de neurones convolutifs (CNN)

5.2.1. Présentation des CNN

Depuis ces dernières années, et grâce aux nombreux travaux de chercheurs, de nouveaux modèles de réseaux de neurones ont été mis au point et perfectionnés. L'un d'eux a connu ses débuts dans les années 1980 : il s'agit des réseaux de neurones convolutifs (en anglais *CNN*, abréviation de *Convolutional Neural Network*).

A l'origine, les CNN ont été inspirés par l'agencement des neurones biologiques dans le cortex visuel des animaux. Chaque neurone est responsable d'une petite partie du champ visuel, et répond à des stimulus qui ne proviennent que de cette zone, appelée « champ réceptif ». Les champs réceptifs de chaque neurone se chevauchent, permettant ainsi de couvrir l'ensemble du champ visuel.

Les réseaux de neurones convolutifs sont aujourd'hui utilisés dans de nombreux domaines, mais sont le plus souvent utilisés pour l'analyse d'images. Ils constituent donc une technologie de choix pour notre problème de reconnaissance et de classification de morphotypes d'éponges. [4] [5]

5.2.2. Fonctionnement d'un CNN pour la classification d'images

Schématiquement, un réseau de neurones convolutifs pour la classification d'images peut être représenté comme ci-dessous (Figure 8). Celui-ci est composé d'une succession de modules, dont chacun a un rôle. L'explication qui suit ainsi que les schémas sont basés sur le cours de Google Developers. [6]

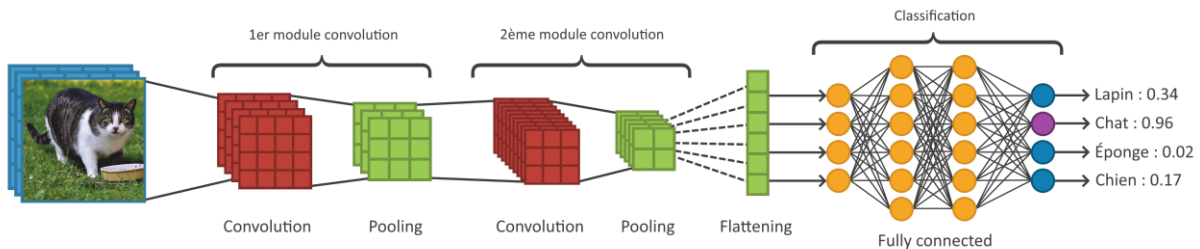


Figure 8 - Schéma d'un réseau de neurones convolutif

Entrée du réseau

En entrée, on retrouve l'image. Celle-ci est représentée comme une matrice tridimensionnelle de dimensions correspondant à la largeur et la hauteur de l'image en pixels. Pour une image en couleur, la dernière dimension est 3 et correspond aux canaux RGB (rouge, vert et bleu) de l'image. Cette matrice est appelée « carte de caractéristique ».

Convolution

La convolution est l'étape qui donne au CNN son intérêt. Elle consiste à faire glisser horizontalement et verticalement une matrice appelée « filtre » sur la carte de caractéristiques d'entrée. La Figure 9 ci-dessous représente un exemple simplifié d'une carte de caractéristiques d'entrée et d'un filtre convolutif.



Figure 9 - Schéma d'une carte de caractéristiques et d'un filtre convolutif

À chaque étape, le filtre glisse sur la carte de caractéristiques. Les éléments de la carte de caractéristiques sont alors multipliés par les éléments du filtre superposé, puis les produits sont additionnés pour obtenir une seule valeur. Cette valeur est alors placée dans la carte de caractéristiques de sortie. Le filtre est ensuite décalé et les mêmes étapes sont répétées jusqu'à ce que la totalité de l'image ait été filtrée. Une étape de la convolution est représentée sur la Figure 10.

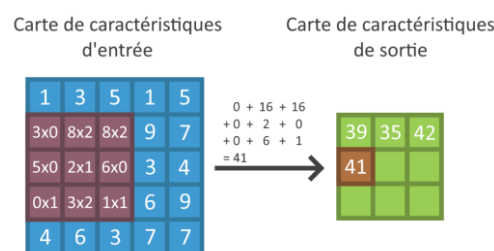


Figure 10 - Schéma d'une convolution d'une carte de caractéristiques d'entrée

Tout l'enjeu de l'apprentissage est donc de régler finement les valeurs des filtres afin d'extraire les caractéristiques les plus intéressantes de l'image.

Pooling

Après la convolution, l'étape de « pooling », permet de sous-échantillonner la carte de caractéristiques. Cela permet de limiter le nombre de valeurs (et donc le temps de traitement), tout en préservant les informations les plus importantes.

Il existe plusieurs algorithmes de pooling, mais l'un des plus utilisés reste le pooling maximal. Comme pour la convolution, celui-ci consiste en un filtre (souvent de dimension 2x2) qui est glissé sur l'image. À chaque étape, le filtre ne garde que la valeur maximale sous le filtre et la transmet à la carte de caractéristiques de sortie. Le fonctionnement du pooling maximal est représenté sur la Figure 11.

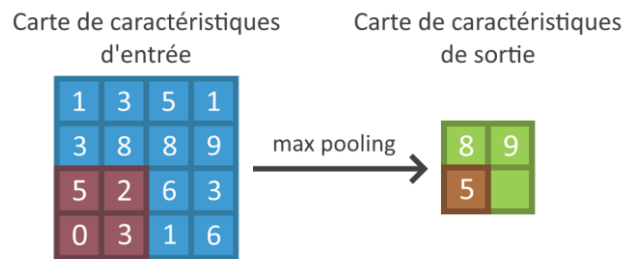


Figure 11 - Schéma du pooling maximal sur une carte de caractéristiques d'entrée

Les étapes de convolution et de pooling sont regroupées dans un « module de convolution ». Généralement, plusieurs de ces modules sont mis les uns à la suite des autres. Cela permet d'extraire à chaque étape des caractéristiques de l'image de plus en plus complexes (forme, texture, etc...).

Fully connected

Après les multiples modules de convolution, la dernière caractéristique de sortie est « aplatie » pendant l'étape de « flattening ». On passe alors d'une matrice à un long vecteur.

On retrouve ensuite une ou plusieurs couches dites « entièrement connectées ». Il s'agit des couches classiques que l'on retrouve dans presque tous les réseaux de neurones, où chaque neurone d'une couche est connecté à tous ceux de la couche suivante. Le rôle de cette partie est d'utiliser les caractéristiques extraites par les convolutions pour effectuer une classification.

Enfin, on retrouve en sortie du réseau une couche « softmax », celle-ci fournit alors une valeur de probabilité comprise entre 0 et 1 pour chaque classe.

La Figure 12 représente ces trois dernières couches du réseau permettant la classification.

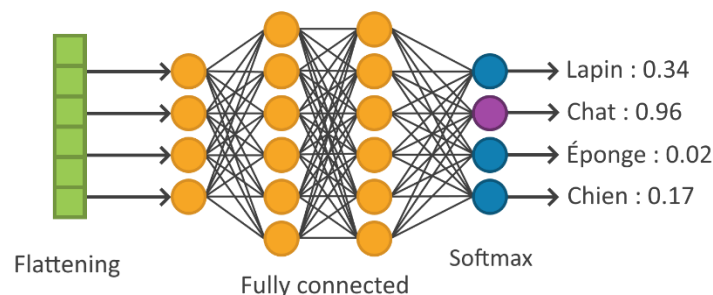


Figure 12 - Schéma des couches flattening, fully connected et softmax

Le fonctionnement expliqué précédemment est bien entendu une version simplifiée de la réalité. Les réseaux actuels ont généralement des couches supplémentaires (dropout, upsampling, ...) avec différents rôles permettant d'améliorer les qualités de prédictions à différents niveaux. Cependant, les couches de convolution restent l'élément central de ces réseaux.

5.3. La recherche d'un réseau

5.3.1. Les algorithmes existants

La détection d'objet sur des images est un problème très courant en intelligence artificielle. Cependant, le développement du Deep Learning et des Big Data a permis des progrès dans ce domaine. Régulièrement, des chercheurs mettent au point de nouveaux algorithmes toujours plus performants.

Utiliser un de ces algorithmes est rapidement apparu comme très intéressant, voire nécessaire, pour notre besoin. En effet, concevoir un CNN de bout en bout nous semblait hors de portée, et n'aurait de toute façon pas été aussi performant que ceux développés par des chercheurs. Cela est d'autant plus vrai que les images des plongées sont relativement complexes. De plus, utiliser des algorithmes déjà faits nous permettait de gagner beaucoup de temps, et donc de pouvoir faire plus d'expérimentations afin de déterminer quel est le plus performant. Enfin, le dernier avantage majeur est lié au fait que l'on puisse utiliser la méthode de l'apprentissage par transfert (en anglais : *transfer learning*).

Le principe du transfer learning est relativement simple. Un réseau de neurones est tout d'abord entraîné sur un jeu de données (en anglais : *dataset*) important. Par exemple, le dataset MS COCO qui comporte plus de 1,5 million d'objets annotés répartis en 80 catégories (voiture, humain, chaise, ...) [7]. Cet entraînement permet au réseau de neurones d'apprendre à reconnaître des éléments qui ne nous intéressent pas. Puis, dans un second temps, le réseau est réentraîné pour détecter les éléments voulus (comme des morphotypes d'éponges). Grâce à cette méthode, le second entraînement est beaucoup plus rapide, car les premières couches du réseau ont déjà appris à extraire les formes, les couleurs et les textures. Cela permet donc d'obtenir un réseau performant avec un nombre limité d'images annotées.

Nous avons donc effectué des recherches sur les différents algorithmes existants pour la détection d'objet. Il en existe de nombreux, mais parmi eux trois familles sont particulièrement utilisées :

Faster R-CNN

Contrairement aux deux autres, la détection par l'architecture Faster R-CNN se déroule en deux étapes. Tout d'abord, un premier réseau de neurones appelé *region proposal network* (RPN) s'occupe d'extraire les régions jugées les plus intéressantes. Dans un second temps, ces régions sont redimensionnées puis un second réseau s'occupe de traiter chaque région une par une pour détecter les objets. Cette architecture a l'avantage de pouvoir détecter des objets plus petits, cependant son fonctionnement en deux temps la rend plus lente que les autres. [8] [9]

SSD (Single Shot Detector)

Contrairement à Faster R-CNN, l'architecture SSD n'est constituée que d'un seul réseau. Ce dernier est composé d'une multitude de couches convolutives permettant d'extraire les informations de l'image à différentes échelles. Ce dernier permet d'atteindre des temps de traitement beaucoup plus courts que Faster R-CNN au prix de moins bonnes détections. [8] [9]

YOLO (You Only Look Once)

Comme pour le SSD, un seul réseau est utilisé par YOLO pour la détection. La différence entre les deux réside dans l'utilisation des couches internes et ne sera pas détaillée ici. Cette architecture a connu des progrès significatifs ces dernières années avec la publication de YOLOv3 en 2018 [10], YOLOv4 en avril 2020 [11] et YOLOv5 en juillet 2020 (ce dernier n'ayant pas fait l'objet d'une publication).

Cela le rend environ 5 fois plus rapide que Faster R-CNN et 1,5 fois plus rapide que SSD [9]. En dépit de son temps de traitement très court, l'architecture YOLOv4 arrive à atteindre des performances très intéressantes, dépassant même Faster R-CNN et SSD [11].

5.3.2. Pré-sélection des réseaux

Au début du projet, nous avons effectué quelques expérimentations avec le réseau de neurones YOLOv4. Celui-ci est en effet très récent (moins d'un an) et a été plébiscité pour ses performances à la fois en termes de qualité de reconnaissance et de temps d'exécution. Il s'agissait donc pour nous d'un bon point de départ pour effectuer nos premiers tests. Un article du site Toward Data Science fournissait tout le nécessaire pour entraîner et tester un réseau YOLOv4 [12]. Ces premiers tests nous ont permis de nous approprier le sujet et de mieux comprendre comment fonctionnait un réseau de neurones.

Par la suite, nous avons souhaité suivre un protocole expérimental plus strict afin de pouvoir évaluer rigoureusement les performances des différents réseaux. Le but était ainsi d'obtenir le meilleur algorithme possible pour l'intégrer dans notre application.

Dans un premier temps, nous souhaitions évaluer 2 grandes familles de réseaux parmi celles citées précédemment : YOLO et Faster R-CNN. Leur fonctionnement étant assez éloigné, comparer l'un et l'autre nous semblait pertinent. Cela est d'autant plus vrai que Faster R-CNN est réputé pour être assez efficace pour la détection de petits objets, comme nos éponges.

Cependant, nous avons finalement décidé de ne pas tester Faster R-CNN, mais plutôt de nous concentrer sur l'amélioration de YOLO. Ce choix fut motivé par plusieurs raisons. La première est que Faster R-CNN est très différent de YOLO. Cela demandait donc beaucoup de temps pour trouver une implémentation qui fonctionne, la comprendre, formater nos données pour ce réseau et s'en servir. Or nous avons déjà perdu plusieurs jours à cause des problèmes liés aux annotations. De plus, nos premiers tests avec YOLOv4 donnaient déjà des résultats très prometteurs. Enfin, YOLOv4 étant très récent, il existe très peu d'articles sur lesquels nous baser afin de savoir si Faster R-CNN vaut le coup d'être testé face à YOLO. Nous avons donc fait le choix de concentrer notre travail sur l'amélioration de YOLO, plutôt que de prendre le risque de perdre du temps sur une architecture plus ancienne (2015) et que nous ne connaissions pas du tout.

Afin de quand même avoir différents éléments de comparaison, nous avons décidé d'expérimenter les réseaux YOLOv4 et YOLOv5. Ces deux versions ont été réalisées par des équipes différentes à quelques mois d'intervalles, la version 5 n'est donc pas une version améliorée de la version 4. L'avantage de tester YOLOv5 plutôt que Faster R-CNN est que le premier utilise des annotations au même format que YOLOv4 et a un fonctionnement global similaire. Ainsi, la prise en main de YOLOv5 a été relativement rapide.

Une autre différence clé entre ces deux réseaux est leur implémentation. YOLOv4, comme ses prédécesseurs, est basé sur le framework Darknet. Ce dernier est un framework open-source en langage C servant à la création de réseaux de neurones. En revanche YOLOv5 est lui réalisé en Python avec le framework PyTorch. Comme pour YOLOv4, les tests avec YOLOv5 ont été réalisés en se basant sur un article de medium [13]. Enfin, le réseau YOLOv5 est disponible en plusieurs versions (small, medium, large et xlarge). D'après leurs auteurs, les versions plus larges sont plus lentes, mais permettent aussi d'obtenir de meilleurs résultats.

Finalement, nous avons donc décidé d'expérimenter le réseau YOLOv4 ainsi que les variantes small, medium et large de YOLOv5.

5.4. Expérimentations

5.4.1. Protocole expérimental

Phases de l'expérimentation

Suite à cette pré-sélection de différents réseaux, nous avons établi un protocole expérimental afin de déterminer quelle configuration permettait d'obtenir les meilleurs résultats. Nous avons globalement 3 leviers pour tenter d'améliorer les performances : le choix du réseau en lui-même, les paramètres du réseau et les données en entrée. Nous avons donc préparé un test en deux phases. D'abord, nous avons cherché à comparer les réseaux présélectionnés pour déterminer quel semblait être le plus adapté. Puis, dans un second temps, nous avons expérimenté plusieurs paramètres sur le réseau sélectionné ainsi que différentes optimisations sur les données d'entrée. Les deux phases ainsi que leurs résultats sont détaillés dans les parties 5.4.2 et 5.4.3.

Création des jeux de données

Nous avons tout d'abord divisé les images annotées en un jeu d'entraînement et un jeu de test. Celui servant à l'entraînement comprenait environ 95% des données soit 217 images. Les 5% restants, soit 11 images ont été réservées pour les tests. La répartition a été faite aléatoirement, nous nous sommes cependant assuré que le jeu de test avait plusieurs éponges de chaque morphotype. L'intérêt d'avoir deux jeux est de pouvoir tester les performances du réseau sur des images avec lesquels il n'a jamais appris, afin de s'assurer qu'il est bien capable de détecter des éponges qu'il n'a jamais vues. Dans l'idéal, un jeu de test plus important aurait permis d'obtenir des résultats plus précis, mais au vu du faible nombre d'annotations, nous avons préféré ne garder que 5%. Les mêmes jeux d'entraînement et de test ont été utilisés pour toutes les expérimentations afin d'obtenir des résultats comparables.

Évaluation des résultats

Afin d'évaluer les résultats de nos expérimentations, nous avons décidé de nous baser sur 3 mesures : la mAP, la matrice de confusion et le temps pour 11 images. La signification et le fonctionnement de ces mesures sont justifiés ci-après. Le but en procédant ainsi était de pouvoir déterminer, à partir de données objectives, quelle est la meilleure configuration.

La métrique la plus utilisée pour évaluer les algorithmes de détection d'objet est la mAP (mean average precision). Sans entrer dans les détails, la mAP se base sur deux autres mesures : la *precision* et le *recall*. La *precision* sert à mesurer à quel point les prédictions faites sont correctes, tandis que le *recall* permet de savoir à quel point l'algorithme arrive à trouver tout ce qu'il est censé trouver. Leur formule sont données sur la Figure 13.

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \end{aligned}$$

TP : Vrai positif (bonne détection)
FP : Faux positif (détection alors qu'il n'y a rien)
FN : Faux négatif (pas de détection alors qu'il y a un élément)

Figure 13 - Formules de la *precision* et du *recall*

Ces deux valeurs sont ensuite calculées pour chaque classe et pour différents seuils de confiance. À partir de ces valeurs, il est possible de tracer la « PR curve », c'est-à-dire la courbe de la *precision* en fonction du *recall*. On appelle alors *average precision* (ou AP) l'aire sous cette courbe. Il s'agit d'une valeur, comprise entre 0 et 1, qui permet de quantifier la qualité des détections pour une classe précise. La *mean average precision* (ou mAP) est la moyenne des AP de chaque classe. Elle permet d'avoir une valeur unique qui donne une idée générale de la qualité d'un réseau. [14]

La mAP est la mesure la plus utilisée pour évaluer les réseaux de neurones dédiés à la détection d'objet. En effet, elle présente l'intérêt de prendre en compte des données importantes : taux de faux positifs, de faux négatifs, seuil de détection, etc... Ainsi, la mAP permet d'avoir une valeur unique permettant de comparer différents réseaux de neurones. Cependant, cela peut aussi être un problème, car cette unique valeur ne permet pas de se rendre compte des spécificités de chaque réseau. Par exemple, deux réseaux avec la même mAP peuvent avoir des détections très différentes. L'un peut être largement meilleur que l'autre pour détecter une classe et inversement. De plus, il est toujours intéressant de connaître quelles classes ont le plus d'erreurs et de quel type sont ces erreurs (faux positifs ou faux négatifs).

Pour toutes ces raisons, nous avons décidé d'utiliser également les matrices de confusion pour comparer les réseaux. Celles-ci ont l'avantage de fournir plus d'informations sur la qualité des détections de chaque classe. De plus, pour réaliser ces matrices, nous avons nous-mêmes compté les prédictions des réseaux. Ainsi, cela permettait de s'assurer que le décompte était correct et non pas potentiellement basé sur des annotations erronées comme pour la mAP.

Les matrices de confusion permettent de mesurer la qualité des prédictions des réseaux de neurones. Chaque ligne correspond à une classe réelle, et chaque colonne à une classe estimée. La cellule ligne L, colonne C contient le nombre d'éléments de la classe réelle L qui ont été estimés comme appartenant à C par le réseau [15]. Dans notre cas, les valeurs des cellules sont normalisées par rapport aux lignes. Cela permet d'avoir, pour chaque classe, la proportion d'éléments correctement détecté et non détecté (faux négatif). Un exemple de matrice de confusion entre 4 classes nommées A, B, C et D est donné sur la Figure 14.

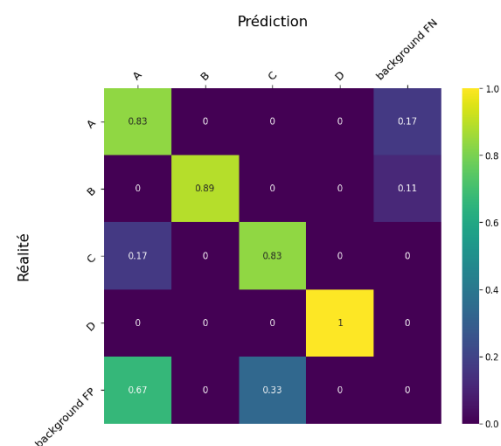


Figure 14 - Exemple de matrice de confusion

Les valeurs sur la diagonale sont liées à la proportion d'éléments correctement classifiés par le réseau. Celles sur la dernière colonne correspondent aux faux négatifs, c'est-à-dire aux éléments qui sont réellement présents, mais que le réseau n'a pas détecté. À l'inverse, celles sur la dernière ligne correspondent aux faux positifs. Dans ce cas, le réseau a détecté un élément qui n'est pas sur l'image. Enfin, les autres valeurs correspondent à des éléments qui ont été détectés, mais mal classés. En résumé, si les valeurs sur la diagonale sont proches de 1, et qu'à l'inverse les autres valeurs sont proches de 0, alors le réseau est performant.

Enfin, notre dernier critère de comparaison a été le temps de traitement pour 11 images. En effet, notre cahier des charges nous impose un temps de traitement par image de 3 secondes au maximum. Il s'agit donc d'un élément important à prendre en compte en plus de la qualité des détections.

Finalement, nous avons réalisé les évaluations des différents réseaux en plusieurs étapes. Tout d'abord, le réseau était entraîné jusqu'à ce que sa mAP n'évolue plus, signifiant que son apprentissage était terminé. L'algorithme ainsi entraîné était alors testé sur les 11 images du jeu de test. Les différentes prédictions étaient alors comptées afin de réaliser la matrice de confusion du réseau. Enfin, nous avons aussi conservé les mesures utiles directement fournies par le réseau : *precision*, *recall*, mAP ainsi que le temps de traitement pour les 11 images. Afin d'avoir des résultats comparables, tous les entraînements et tests ont été réalisés sur la plateforme Google Colaboratory.

5.4.2. Phase 1 : Tests des réseaux

Notre première phase de test avait pour objectif de déterminer quel réseau était le plus performant parmi ceux présélectionnés (YOLOv4 et les variantes *small*, *medium* et *large* de YOLOv5). Comme expliqué dans la partie précédente, tous les réseaux ont été entraînés avec les mêmes données afin d'avoir des résultats comparables.

Une différence notable liée à l'apprentissage est que le pas d'apprentissage pour YOLOv4 est l'itération, tandis que YOLOv5 utilise l'*epoch*. Sans entrer dans les détails, une *epoch* correspond au passage de tout le jeu d'entraînement dans le réseau (soit les 217 images). Tandis que l'itération est liée au passage d'un nombre précis d'images (dans notre cas 64) dans le réseau. Par conséquent, une *epoch* correspond à environ 3,4 itérations. Bien que cette différence soit importante pour comprendre les graphiques qui suivent, cela ne change pas les résultats finaux du réseau.

La Figure 15 ci-dessous présente les courbes d'apprentissages pour les 4 réseaux présélectionnés.

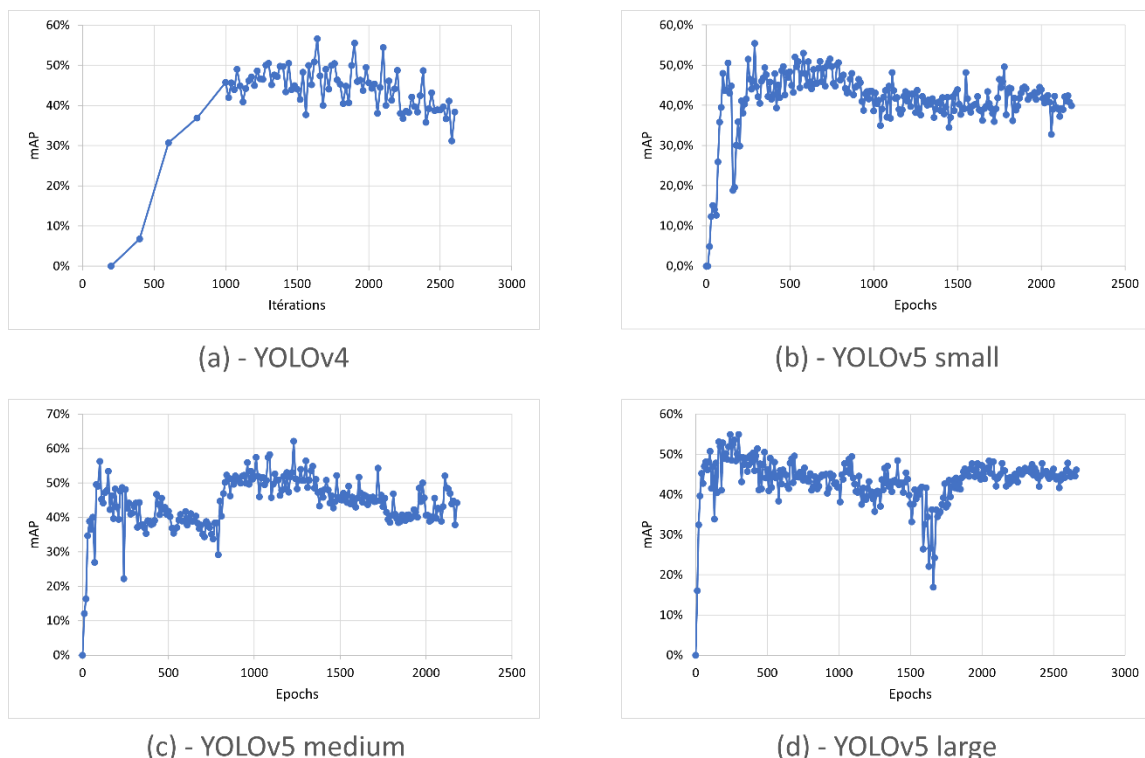


Figure 15 - Évolution de la mAP durant l'apprentissage des réseaux YOLOv4 (a), YOLOv5 small (b), YOLOv5 medium (c) et YOLOv5 large (d)

Dans l'ensemble, on constate que les courbes suivent des tendances assez similaires. La progression de la mAP est d'abord relativement rapide jusqu'à atteindre un plateau. Ensuite, les courbes ont tendance à redescendre progressivement. On peut cependant remarquer que les graphiques pour YOLOv5 présentent des sauts importants qu'on ne retrouve pas sur la courbe d'apprentissage de YOLOv4.

En termes de qualité de détection, il semblerait que YOLOv5 soit légèrement meilleur, notamment la variante medium qui présente une mAP autour de 53% sur son plateau haut, avec un pic allant jusqu'à 62%. Les moins bons semblent être YOLOv4 et YOLOv5 small avec une mAP plutôt centrée autour de 45% et des pics allant jusqu'à 56%.

Une différence importante cependant qui n'est pas visible sur les graphiques est le temps d'entraînement. L'apprentissage de YOLOv4 a duré environ 15h pour atteindre l'itération 2500. En revanche, atteindre l'epoch 2000 avec YOLOv5 (soit l'équivalent de l'itération 6800) a mis entre 5 et 10h selon les variantes. L'apprentissage de YOLOv5 est donc significativement plus rapide que celui de YOLOv4.

Une fois les entrainements terminés, les différents réseaux ont été évalués sur leur configuration ayant atteint la meilleure mAP. Le Tableau 9 regroupe les différentes mesures prises sur chacun des réseaux.

Réseau	mAP	Precision	Recall	Temps pour 11 images	Avec CUDA
YOLOv4	56,6%	52,0%	64,0%	26,4s	6,5s
YOLOv5 small	55,4%	74,1%	48,5%	9,5s	5,9s
YOLOv5 medium	62,1%	58,4%	62,2%	14,1s	7,4s
YOLOv5 large	55,0%	70,3%	57,8%	16,6s	7,3s

Tableau 9 - Mesures prises sur les différents réseaux testés

Le temps pour 11 images correspond au temps qu'a mis le réseau à traiter les 11 images du jeu de test. La valeur « avec CUDA » est liée au temps pour effectuer cette même opération sur la carte graphique. En effet, les cartes graphiques NVIDIA disposent d'une technologie nommée CUDA permettant d'accélérer les calculs en les effectuant sur la carte graphique plutôt que le processeur. Ce point sera abordé plus en détail dans l'intégration du réseau de neurones dans la partie 6.3.2.

En termes de temps de traitement, on remarque que YOLOv5 est entre 1,5 et 2,5 fois plus rapide que YOLOv4. Ce résultat est même encore amélioré avec l'utilisation de CUDA permettant d'effectuer les calculs environ deux à quatre fois plus vite que sur processeur.

En ce qui concerne la qualité des prédictions, les résultats montrent une grande disparité entre la *precision* et le *recall* des réseaux testés, malgré des mAP proches. On constate notamment que YOLOv4 a une *precision* plus faible, mais un *recall* plus élevé que les réseaux YOLOv5. Cela signifie que lorsque YOLOv4 détecte une éponge, il fait plus souvent une erreur, mais en contrepartie il rate moins d'éponges censées être détectées. Cela se retrouve dans les matrices de confusion obtenues (Figure 16).

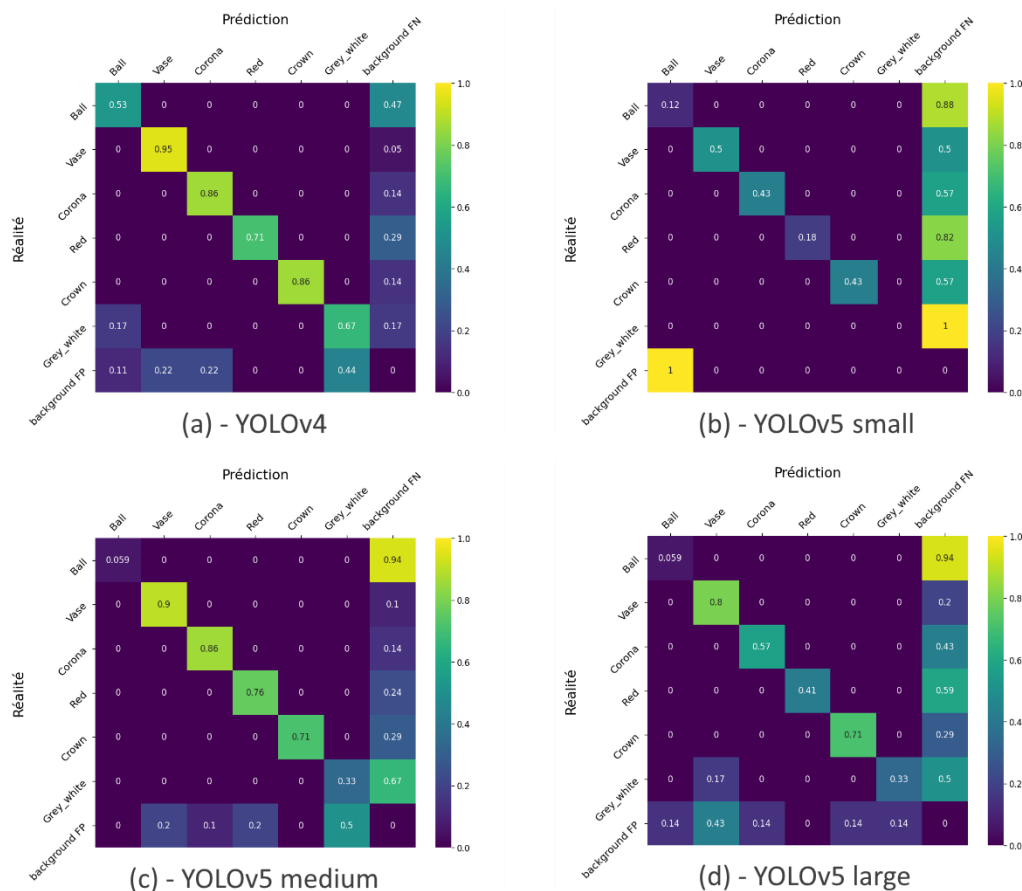


Figure 16 - Matrices de confusions obtenues sur les réseaux YOLOv4 (a), YOLOv5 small (b), YOLOv5 medium (c), et YOLOv5 large (d)

Ces matrices permettent de donner du sens au tableau précédent. En effet, on remarque que YOLOv5 présente beaucoup plus de faux négatifs que YOLOv4. Cela se remarque surtout pour les morphotypes les plus difficiles comme Ball avec environ 90% de faux négatifs sur les 3 variantes de YOLOv5 (contre 47% pour YOLOv4). On retrouve cette tendance pour les éponges Grey_white, avec 17% de faux négatifs pour YOLOv4 (mais avec plus de faux positifs) contre 50 à 100% pour YOLOv5.

Pour les morphotypes plus simples (comme Vase ou Crown), les résultats entre YOLOv4 et les versions medium et large de YOLOv5 donnent des résultats du même ordre de grandeur. Cela explique la mAP plus importante des réseaux YOLOv5 : ces derniers préfèrent ne jamais détecter les morphotypes les plus complexes plutôt que de risquer de faire une erreur. Cela leur permet d'atteindre une *precision* plus haute (au prix du *recall*), et ainsi d'obtenir des mAP comparables à YOLOv4 tout en étant moins performants.

Suite à ces résultats, nous avons choisi de conserver le réseau YOLOv4 qui présentait une matrice de confusion plus intéressante que les variantes de YOLOv5. Dans l'ensemble, ce réseau arrive quand même à atteindre des performances satisfaisantes avec plus de 50% de bonnes détections pour chaque morphotype. Sur certaines classes comme Crown, Corona et Vase, le réseau arrive même à atteindre plus de 85% de bonnes détections.

Cela montre aussi l'intérêt de comparer les réseaux avec les matrices de confusion et de ne pas se fier uniquement aux mAP.

5.4.3. Phase 2 : Optimisation du réseau

La seconde étape de nos expérimentations a consisté à tester différentes optimisations sur le réseau sélectionné afin de tenter d'obtenir de meilleurs résultats. Dans l'idéal, il aurait fallu tester toutes les optimisations sur tous les réseaux possibles (et non pas uniquement sur le meilleur), cependant cela n'était pas envisageable en raison de la durée limitée du projet.

Nous avons ainsi déterminé 3 optimisations possibles pour le réseau :

- La taille d'entrée du réseau : Il s'agit de la taille à laquelle le réseau redimensionne l'image avant de la traiter. Plus cette taille est petite, et plus l'image sera réduite et perdra donc en qualité et en information. Par défaut, cette valeur était de 416x416px.
- Le découpage des images : L'idée est de prédécouper les grandes images en plusieurs sous images avant de les traiter. Comme pour le point précédent, le but est de mieux détecter les petits éléments. En contrepartie, cela demande de traiter plus d'images. Par défaut, les images n'étaient pas découpées.
- La taille des annotations : Cette optimisation a pour objectif de compenser les problèmes rencontrés avec les annotations fournies, qui n'étaient parfois pas centrées sur l'éponge. Le principe consiste simplement à augmenter ou réduire le diamètre des annotations pour tenter d'obtenir de meilleures détections. Par défaut, nous avons décidé de multiplier le diamètre des annotations par 1,5.

Lors du test précédent, tous les réseaux étaient entraînés avec les valeurs par défaut. Dans les expérimentations qui suivent, seul le paramètre testé est changé, les autres gardent leur valeur par défaut.

Taille d'entrée du réseau

Par défaut, le réseau redimensionne les images en 416x416px. Or nous avons montré dans la partie 4.2.3 que beaucoup d'annotations étaient petites relativement à la largeur de l'image. Le risque de ce redimensionnement est donc que les éponges déjà petites ne deviennent que quelques pixels sur l'image redimensionnée. Nous avons donc modifié ce paramètre afin de tester une taille d'entrée de 512x512px et de 832x704px. Les résultats obtenus sont donnés à travers le Tableau 10 et la Figure 17.

Taille d'entrée	mAP	Precision	Recall	Temps pour 11 images	Avec CUDA
416x416 (défaut)	56,6%	52,0%	64,0%	26,4s	6,5s
512x512	59,9%	60,0%	67,0%	47s	N/A
832x704	62,6%	65,0%	63,0%	76s	N/A

Tableau 10 - Mesures prises sur les réseaux YOLOv4 avec différentes tailles d'entrée

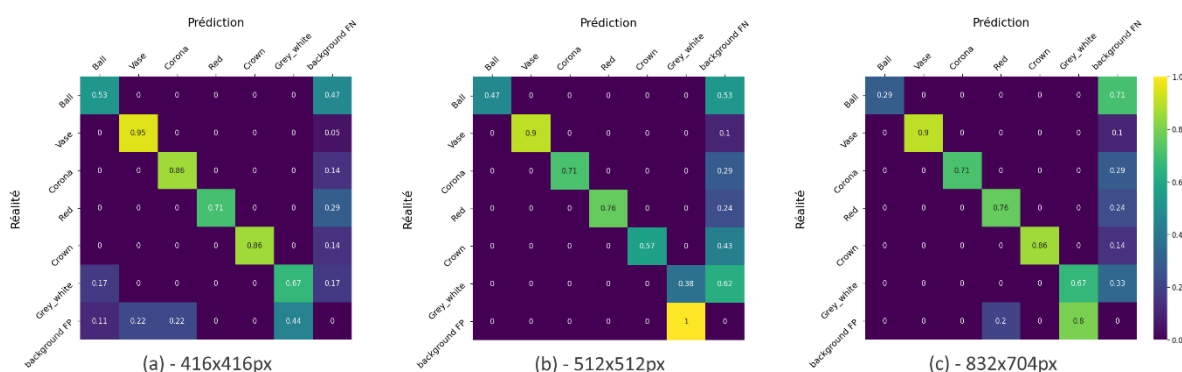


Figure 17 - Matrices de confusions obtenues sur les réseaux YOLOv4 avec une entrée de 416x416px (a), 512x512px (b), et 832x704px (c)

Contrairement aux attentes, les réseaux avec une plus grande entrée ne sont pas meilleurs. Dans l'ensemble, pour les morphotypes les plus simples (Vase, Corona, Red et Crown), les résultats sont équivalents, avec cependant moins de faux positifs. Cependant, pour les deux classes les plus difficiles (Ball et Grey_white), les résultats se dégradent avec l'augmentation de taille. De plus, le temps de traitement est plus important avec respectivement 4,2s et 6,9s par image pour les tailles d'entrées 512x512px et 832x704px. Par conséquent, au-delà de leurs détections moins bonnes, le temps de calcul est de toute façon trop long par rapport aux 3 secondes imposées par le cahier des charges.

Découpage des images

Comme pour la taille d'entrée du réseau, le but est d'aider à une meilleure détection des petits éléments. Le principe consiste simplement à découper les images en 4, ainsi au lieu d'avoir une grande image de 4243x2828px, on a 4 images de 2121x1414px. Cette technique a cependant deux risques. Tout d'abord, cela demande de traiter 4 fois plus d'images, ce qui augmente le temps de calcul. Le second est que des éponges peuvent se retrouver coupées entre 2 images et ne pas être détectées. Les résultats obtenus sont donnés dans le Tableau 11 et la Figure 18.

Découpage	mAP	Precision	Recall	Temps pour 11 images	Avec CUDA
Aucun (défaut)	56,6%	52,0%	64,0%	26,4s	6,5s
Par 4	65,2%	77,0%	68,0%	86,2s	N/A

Tableau 11 - Mesures prises sur les réseaux YOLOv4 avec et sans prédécoupage des images

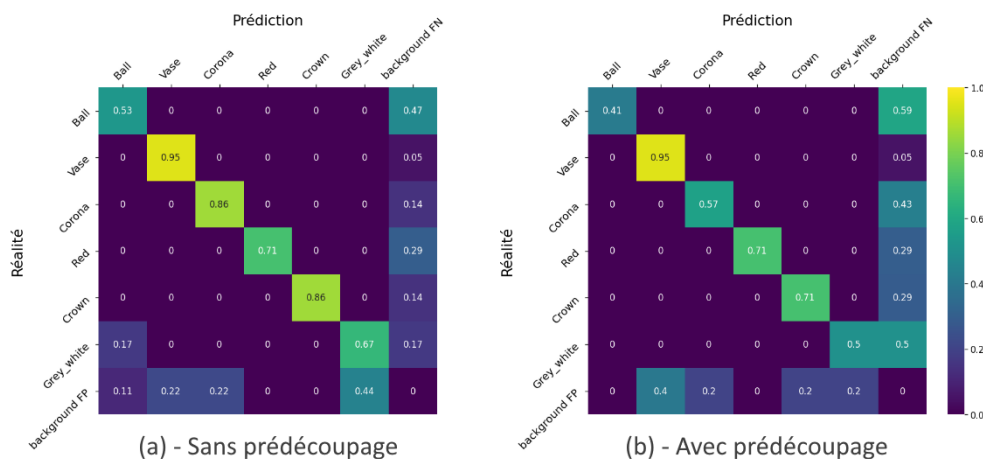


Figure 18 - Matrices de confusion obtenues sur les réseaux YOLOv4 sans prédécoupage des images (a) et avec un prédécoupage en 4 images (b)

Le constat est globalement le même que pour les réseaux avec une taille d'entrée plus importante. À première vue les mesures (mAP, *precision* et *recall*) sont meilleures, mais la matrice de confusion est moins bonne. Dans l'ensemble, on constate que le taux de bonnes détections est soit identique (comme pour Vase), soit inférieur (pour les autres). On constate aussi plus de faux négatifs, même sur des morphotypes plus simples comme Corona et Crown. De plus, comme prévu le temps de traitement est plus long avec près de 8s par image.

Ces deux premières tentatives d'optimisation sont intéressantes, car elles ont donné des résultats contraires aux attentes. En effet, on pouvait s'attendre que le réseau soit aidé par des images avec une plus haute résolution, mais ce n'est pas le cas. Bien qu'il soit difficile de savoir pourquoi le réseau est moins bon, nous pouvons faire une hypothèse. En effet, il est possible que les images en moins bonne qualité permettent d'aider le réseau en réduisant la quantité d'informations. On peut ainsi imaginer que cela lui permet d'atteindre une meilleure capacité de généralisation avec relativement peu d'annotations.

Taille des annotations

Comme expliqué dans la partie 4.2, l'irrégularité dans les tailles des annotations est un problème majeur. Certaines sont légèrement plus petites ou plus grandes que l'éponge. De plus, quelques annotations ne sont pas centrées sur l'élément, avec parfois plus de la moitié de l'éponge en dehors de la zone annotée. Afin de résoudre ce problème, nous avons tenté d'augmenter plus ou moins le diamètre des annotations. Les résultats obtenus sont donnés dans le Tableau 12 et sur la Figure 19.

Taille d'annot.	mAP	Precision	Recall	Temps pour 11 images	Avec CUDA
× 1	46,4%	56,0%	57,0%	28,2s	N/A
× 1,5 (défaut)	56,6%	52,0%	64,0%	26,4s	6,5s
× 2	55,9%	69,0%	59,0%	24,1s	N/A

Tableau 12 - Mesures prises sur les réseaux YOLOv4 après entraînement avec différentes tailles d'annotations

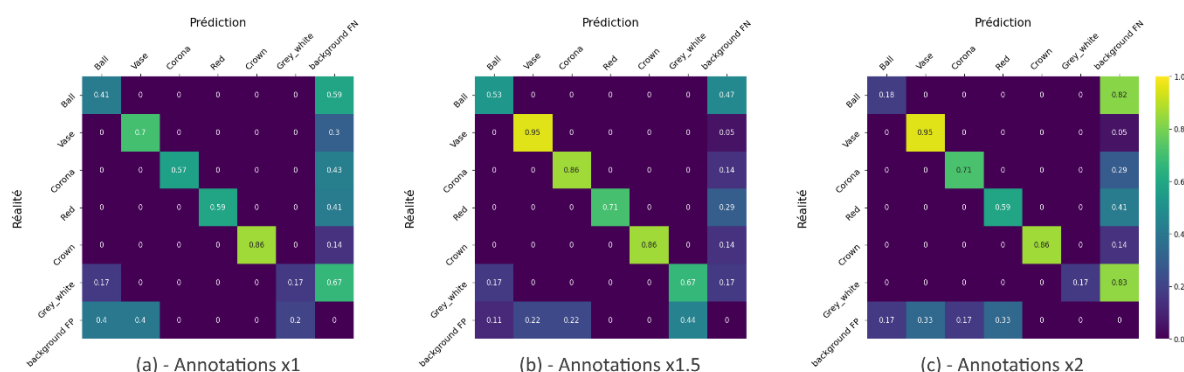


Figure 19 - Matrices de confusion obtenues sur les réseaux YOLOv4 après entraînement sur les annotations fournies (a), augmentées de 1,5 fois leur diamètre (b), et augmentées de 2 fois leur diamètre (c)

On peut remarquer que lorsque les annotations ne sont pas agrandies, les résultats sont plutôt mauvais avec plus de 40% de faux négatifs pour des morphotypes plus simples comme Corona et Red. De la même manière, le réseau entraîné avec des annotations dont le diamètre a été multiplié par deux n'est lui aussi pas très bon. Il fonctionne bien pour les plus gros morphotypes, mais est mauvais à détecter les Ball et les Grey_white. Le meilleur réseau d'après les matrices de confusion est celui qui a été entraîné avec des annotations dont le diamètre a été multiplié par 1,5. Nous pouvons faire l'hypothèse que cela permet de corriger les annotations mal placées en incluant une plus grande zone. De plus, il est possible qu'inclure le « contexte » autour de l'éponge aide le réseau. Enfin, le temps de traitement est similaire, ce qui est normal étant donné que la structure du réseau en elle-même n'a pas changé.

5.4.4. Conclusion des expérimentations

Suite à ces expérimentations, nous avons conclu que le meilleur modèle est un réseau YOLOv4, avec une entrée de 416x416px, sans prédécoupage et entraîné avec des annotations agrandies 1,5 fois. Même si ce n'est pas celui donnant les meilleures mesures (mAP : 56,6%, *precision* : 52,0% et *recall* : 64,0%), il s'agit de celui permettant d'obtenir le plus de bonnes détections tous morphotypes confondus. De plus, son temps de traitement pour 11 images de 26,4s, soit 2,4s par image. Les tests ont aussi montré l'intérêt de CUDA qui améliore encore la vitesse de calcul pour atteindre 0,6s par image.

Nous avons aussi conduit d'autres expérimentations mineures (comme essayer un découpage des images avec un réseau YOLOv5), mais les résultats n'étant pas concluants, nous avons décidé de ne pas en parler. De plus, ces résultats doivent être remis dans leur contexte. En effet, les jeux de test et d'entraînement étant relativement petits, une autre répartition aurait peut-être conduit à des résultats différents. Des tests avec plus d'annotations seraient donc utiles pour valider ou non nos conclusions.

6. Interface Homme-Machine

6.1. Introduction

6.1.1. Objectifs

Nous avons développé notre interface homme-machine (IHM) avec le principal objectif qu'elle soit claire, épurée et utilisable par une personne qui n'a aucune expérience préalable en informatique ou en intelligence artificielle. De plus, il était nécessaire qu'elle fournisse un maximum d'informations pour que celles-ci soient exploitées dans le cadre de recherches scientifiques par des biologistes de l'Ifremer.

Concernant la mise en page de l'application, nous avons dans un premier temps réalisé une maquette technique prévisionnelle. Le but était de placer les différentes fonctionnalités prévues avant de les implémenter. Lors de la première réunion avec l'Ifremer, nous leur avons montré cette maquette pour avoir leurs retours et leurs attentes. Cela a permis de répondre à leurs premières questions et d'obtenir des retours avant même d'avoir commencé à développer l'interface. Dans l'ensemble, ils ont apprécié et approuvé cette maquette. Elle a donc été notre modèle pour concevoir l'application. Pour plus de détails, la maquette proposée est disponible en annexe de ce document.

Enfin, nous avons cherché à développer un visuel autour de notre application. Nous lui avons donné le nom de « Spongo » et nous avons créé un logo (Figure 20) afin que celle-ci soit facilement identifiable. De plus, nous avons développé une charte graphique pour apporter une identité visuelle à l'interface et la rendre à la fois reconnaissable mais, aussi plus agréable à utiliser.



Figure 20 - Logo créé pour l'application

6.1.2. Technologies utilisées

L'application a été développée sur un environnement Windows 10. Nous avons décidé d'utiliser le langage Python et le framework graphique Qt grâce au module PySide2. Nous avons fait le choix de ces technologies pour plusieurs raisons.

Tout d'abord, le choix de travailler dans un environnement Windows 10 a principalement été motivé par le fait que l'Ifremer travaille sur ce même système d'exploitation. Bien que Python soit un langage interprété, il nous semblait plus pratique de travailler dans le même environnement que celui visé.

Le langage Python est un langage interprété, contrairement au C++ qui est compilé. L'avantage de ce type de langage est qu'il peut être exécuté sur plusieurs plateformes différentes, ce qui facilite grandement le déploiement des applications. C'était un point fondamental pour nous car il était nécessaire que notre application soit facilement installable par les biologistes de l'Ifremer. De plus, ce langage est orienté objet et a un haut niveau d'abstraction, utile pour développer rapidement avec relativement peu de code.

Python est aussi un langage qui possède de nombreuses bibliothèques open sources, et notamment certaines conçues spécialement pour de l'apprentissage profond (PyTorch), pour de l'analyse de données (Pandas) ou encore pour des calculs numériques (Numpy). Cette modularité était l'idéal pour implémenter notre réseau de neurones dans l'application de façon simple et rapide.

Enfin, Python et Qt sont des technologies avec lesquelles nous avons déjà eu l'occasion de travailler auparavant. Cette maîtrise a été un point important car cela nous permettait de mieux visualiser comment réaliser l'interface et ainsi mieux nous projeter en termes de temps et d'objectifs.

6.2. Page « Paramètres »

Cette page permet aux utilisateurs de définir les paramètres liés à l'analyse avant de commencer cette dernière. Le but étant de laisser le choix à l'utilisateur des réglages les plus importants, tout en restant clair et utilisable pour quelqu'un n'y connaissant rien en réseau de neurones.

Dans cette partie, nous expliquerons les paramètres que nous avons décidé d'intégrer et pourquoi nous avons laissé le choix à l'utilisateur de les modifier selon leurs attentes. La Figure 21 donne un aperçu de cette page.

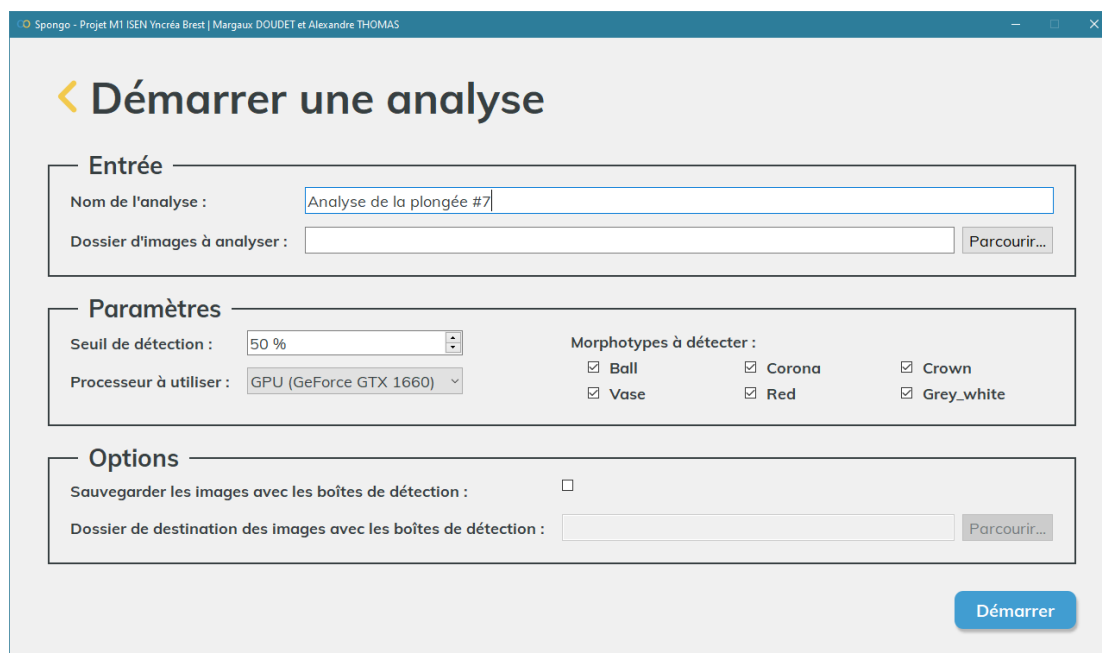


Figure 21 - Aperçu de la page "Paramètres"

6.2.1. Entrée

La partie supérieure sert à donner les informations de base pour l'analyse : son nom, et le dossier des images qui doivent être analysées.

Le nom sert simplement à retrouver cette analyse en particulier, notamment dans l'historique. Par défaut, un nouveau nom de la forme « Analyse de la plongée #X » est généré, avec X qui s'auto-incrémente en fonction du nombre d'analyses déjà réalisées.

Le dossier d'image à analyser est le seul paramètre obligatoire que doit définir l'utilisateur, tous les autres pouvant être laissés à leur valeur par défaut. Le bouton « Parcourir » ouvre une boîte de dialogue permettant de sélectionner le dossier contenant les images.

L'application cherche les images situées uniquement dans le dossier choisi, et non pas dans d'éventuels sous-dossiers. De plus, seuls les formats les plus communs comme le JPEG (extension .jpg ou .jpeg) et le PNG (extension .png) sont supportés. Si des fichiers non supportés (comme des fichiers .txt par exemple) sont présents dans le dossier, ceux-ci sont simplement ignorés.

6.2.2. Seuil de détection

Lors de l'analyse, l'application détecte les éponges sur les images et associe à chaque détection une valeur comprise entre 0 et 100%. Plus cette valeur est haute, plus l'application est certaine que sa détection est bonne. Le seuil de détection correspond au seuil à partir duquel l'application décide de conserver la détection réalisée. Un seuil haut garantit de ne garder que les meilleures détections, mais l'application risque de passer à côté de certaines éponges sur lesquelles elle est moins sûre (faux négatifs). À l'inverse, un seuil bas permettra de détecter plus d'éponges, mais avec le risque d'avoir plus d'erreurs également (faux positifs).

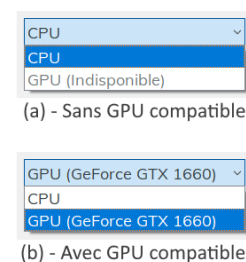
Il nous semblait important de laisser le choix à l'utilisateur de la valeur du seuil de détection car c'est un critère qui dépend beaucoup du ou des morphotypes à détecter. Certaines éponges sont plus difficiles à reconnaître pour le réseau, ce qui fait baisser les taux de confiance avec lesquelles il les détecte. De manière générale, les critères qui peuvent faire baisser le taux de confiance d'une détection sont sa taille et sa ressemblance avec son environnement. En effet, plus une éponge est petite et plus le réseau a du mal à la voir (morphotypes « Ball » et « Corona »). Dans la même logique, plus une éponge ressemble à son environnement, comme le sol ou un autre élément du fond marin, alors plus le réseau aura des difficultés à la distinguer parmi ces autres éléments (morphotype « Grey_white »). Les taux de confiance pour ces détections sont donc généralement plus faibles. Pour en rater un minimum, il est alors recommandé de choisir un seuil de détection plus faible (entre 30 et 60%).

À l'inverse, un morphotype désignant des éponges de grandes tailles et avec une forme singulière sont très bien détectées. C'est le cas des éponges « Vase », « Red » et « Crown » qui possèdent des taux de confiance importants. Dans ces cas-là, il est recommandé de choisir un seuil de détection plus élevé (entre 50 et 80%).

Par défaut, le seuil de détection est mis à 50%, car cette valeur nous semblait optimale d'après nos expérimentations. Il s'agit d'un seuil pour lequel les détections nous semblent les plus équilibrées afin d'obtenir le moins de faux positifs et de faux négatifs possibles.

6.2.3. Processeur

L'analyse des images demande de faire des calculs importants. Afin d'accélérer ces calculs, il est possible de les effectuer sur la carte graphique (aussi appelée GPU) si celle-ci supporte la technologie CUDA. Tous les ordinateurs ne supportant pas cette technologie, nous avons laissé le choix du composant sur lequel effectuer l'analyse : la carte graphique ou le processeur (aussi appelé CPU). L'aperçu présenté sur la Figure 22 montre les choix disponibles sur les PC ne disposant pas de GPU compatible avec CUDA et sur ceux disposant d'au moins une carte graphique compatible.



Par défaut, si un GPU est disponible, il sera sélectionné. Dans le cas contraire, si aucun GPU n'est détecté sur l'ordinateur, les calculs seront effectués sur le CPU, mais l'analyse prendra plus de temps. Cela permet à l'application de toujours s'adapter à l'environnement technique dans laquelle elle est exécutée pour avoir les meilleures performances possible.

Cela nous semblait pertinent de laisser le choix du processeur à utiliser. Ainsi les chercheurs de l'Ifremer peuvent immédiatement savoir si leur ordinateur est compatible pour effectuer l'analyse rapidement. De plus, cela nous permet aussi de comparer la différence de temps de calcul entre les processeurs.

L'implémentation du réseau de neurones et l'utilisation du GPU sont expliquées dans la partie 6.3.2.

6.2.4. Sélection des morphotypes

Bien que le nombre de morphotypes à détecter ne change pas la durée de l'analyse, nous avons choisi de ne pas imposer la détection de tous les morphotypes afin de nous adapter aux besoins des utilisateurs. Nous souhaitons lui laisser la liberté de choisir les éponges qu'il souhaite détecter. De cette manière, cela permet d'afficher et d'obtenir des résultats d'analyse seulement sur les informations jugées utiles.

Une autre raison pour laquelle nous avons laissé le choix des morphotypes est due au seuil de détection. En effet, tous les morphotypes n'obtiennent pas les mêmes taux de confiance. Par exemple, les taux de confiance des éponges « Ball » sont généralement moins élevés que pour les éponges « Vase » à cause de leur taille. Cela peut donc devenir problématique de choisir le bon seuil dans le cas où ces deux morphotypes souhaitent être détectés. En laissant le choix des morphotypes, il est alors possible de lancer une analyse individuellement sur chaque morphotype afin d'y associer le seuil de détection optimal.

6.2.5. Sauvegarde des images

Nous avons aussi laissé la possibilité de sauvegarder sur son ordinateur les images avec les boîtes de détection des éponges, générées et affichées lors de l'analyse. Cette action est utile dans le cas où l'utilisateur souhaite garder une trace visuelle des détections pour les exploiter ensuite. Ainsi, sur des petites analyses, les utilisateurs peuvent voir ce que l'algorithme a détecté et avec quel niveau de confiance. La couleur de la boîte de détection dépend du morphotype et le taux de confiance est la valeur comprise entre 0 et 1 située après le nom. La Figure 23 est un exemple d'une image sauvegardée présentant plusieurs morphotypes détectés.



Figure 23 - Exemple d'une image obtenue lorsque la sauvegarde des images est activée

Cependant, nous avons choisi d'intégrer cette fonctionnalité en tant qu'option pour deux raisons. La première, elle n'est pas toujours jugée nécessaire. Il est inutile de sauvegarder les images dans le cas où elles ne sont pas exploitées par la suite. La seconde, lors d'analyses importantes (plus d'une centaine d'images), sauvegarder toutes ces images, même redimensionnées, peut prendre beaucoup de place sur le disque dur. Il s'agit donc d'une option principalement intéressante pour faire des tests, mais pas lors d'une utilisation régulière de l'application.

6.3. Page « Analyse »

La page « Analyse » correspond à la page principale de l'application. Elle est affichée lorsque l'analyse est en cours et a pour but d'en indiquer la progression et d'afficher les principales informations. Un aperçu de cette page est proposé Figure 24.

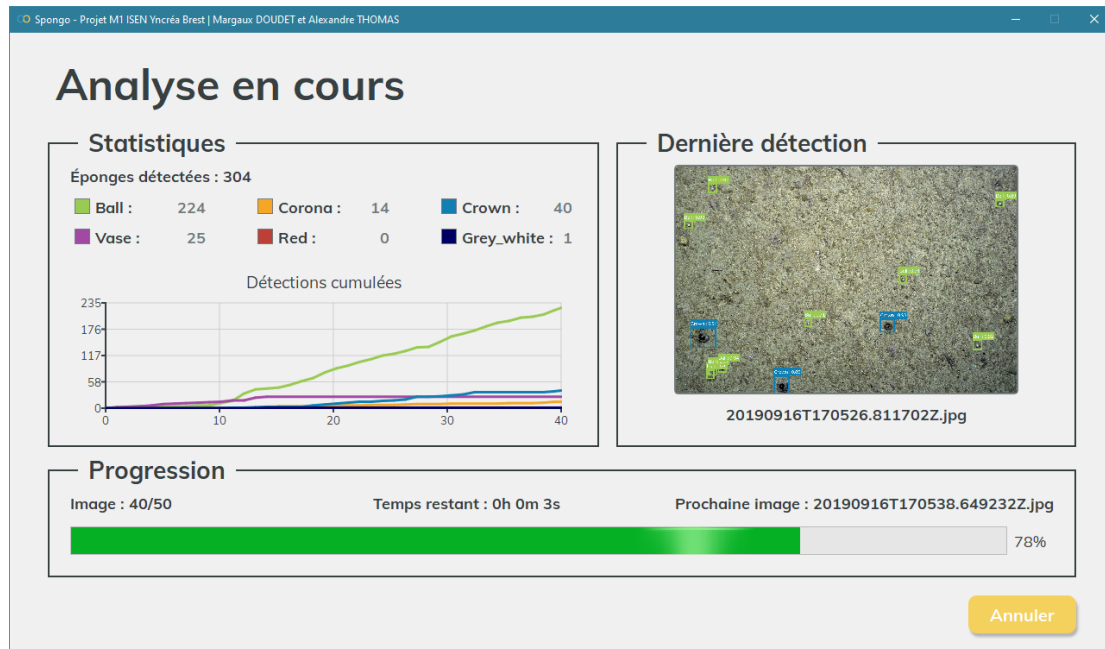


Figure 24 - Aperçu de la page "Analyse"

6.3.1. Explication de l'application

La page a été conçue dans le but d'être la plus visuelle et informative possible. Pour ce faire, nous avons séparé la page en trois sections : les statistiques, la dernière image avec des détections et la progression.

Statistiques

Pour obtenir un retour visuel des détections d'éponges, nous avons choisi d'intégrer un graphique affichant les détections cumulées des éponges. Cela a été motivé par le fait que beaucoup d'images ont peu ou pas d'éponges. Par conséquent, un graphique du nombre de détection à chaque image aurait présenté des courbes souvent à 0, avec de nombreux pics correspondant aux images avec des détections. Un tel graphique avec 6 courbes superposées aurait été très peu lisible, surtout lors des analyses avec beaucoup d'images. Nous avons donc opté pour un graphique utilisant les détections cumulées (c'est-à-dire la somme des détections des images précédentes). Cela permet d'avoir les courbes croissantes qui se séparent, ce qui rend le tout plus lisible et exploitable.

Dernière image avec des détections

Comme son nom l'indique, cette section affiche la dernière image contenant une ou plusieurs détections. Cela permet d'avoir un retour visuel sur les détections du réseau afin de vérifier que l'analyse se déroule comme prévu.

Progression

La section progression permet d'indiquer l'avancement de l'analyse. Le temps estimé est calculé en se basant sur le temps mis pour traiter les images déjà analysées. Ce calcul est possible car le temps de traitement est similaire pour chaque image. Par conséquent, la durée totale d'analyse est directement proportionnelle au nombre d'images.

6.3.2. Implémentation du réseau de neurones

Une étape cruciale de notre projet a été d'implémenter dans l'application le réseau de neurones entraîné et testé sur le service Google Colaboratory.

Dans un premier temps, nous avons repris le réseau YOLOv4 entraîné préalablement et nous l'avons intégré à l'IHM avec la bibliothèque OpenCV. Nous obtenions une application fonctionnelle, qui détectait efficacement les éponges sur les images qui lui étaient fournies. Le temps de traitement pour une seule image était alors de 2 à 3 secondes.

Par la suite, nous souhaitions aller plus loin et améliorer les performances du réseau en rendant possible son exécution sur la carte graphique. En effet, les tests réalisés ont montré que l'utilisation de CUDA permettait d'effectuer les calculs de YOLOv4 jusqu'à 4 fois plus vite. Il nous semblait pertinent de profiter de ce gain de temps non négligeable pour notre application, surtout lorsque certaines analyses avec des milliers d'images peuvent durer plusieurs heures.

La technologie CUDA (abréviation de *Compute Unified Device Architecture*) est supportée par la plupart des cartes graphiques NVIDIA. Elle est optimisée pour effectuer certaines opérations sur la carte graphique plutôt que sur le processeur, permettant ainsi d'accélérer les calculs. CUDA est notamment très utilisé pour les opérations de machine learning [16] qui nécessitent des calculs importants.

Nous avons donc cherché à utiliser OpenCV avec l'architecture CUDA comme nous l'avons fait lors des expérimentations. Cependant, il s'est avéré compliqué de mettre en place cette solution. En effet, elle demandait de recompiler OpenCV pour les ordinateurs de l'Ifremer afin de supporter CUDA. Cependant, la compilation est relativement longue (2 à 3 heures) et de nombreux problèmes peuvent survenir (dépendances manquantes, CUDA mal installé, etc). Cela n'était donc pas compatible avec notre contrainte de proposer une installation simple, nous avons donc cherché une autre méthode.

La meilleure solution était alors de trouver une alternative à OpenCV pour implémenter le réseau de neurones YOLOv4 dans l'application. Pour ce faire, nous nous sommes basés sur nos tests de réseaux et notamment sur le réseau YOLOv5. En effet, contrairement à YOLOv4 qui peut être implémenté avec OpenCV, YOLOv5 se base sur le framework PyTorch qui, lui, supporte CUDA par défaut. Nous avons donc cherché à implémenter YOLOv4 sur PyTorch. Cette solution était avantageuse car nous avions quelques bases sur ce framework grâce à nos expérimentations sur YOLOv5 et car nous connaissions ses bonnes performances. Suite à nos recherches, nous avons trouvé un code qui adapte YOLOv4 pour utiliser PyTorch. Ce dernier étant sous licence Apache2 (donc open-source), nous pouvions nous en servir [17]. Nous l'avons donc adapté puis intégré dans notre application.

D'après nos mesures, l'utilisation de CUDA a permis de faire passer le temps de calculs pour 50 images de 3 minutes sur CPU à 20 secondes sur GPU (GTX 1660), soit environ 9 fois plus rapide. À noter que ces performances peuvent varier en fonction du CPU et de la carte graphique NVIDIA en question.

6.3.3. Calcul du score d'intérêt

Pour chaque image, nous calculons également un score d'intérêt. Les images considérées comme les plus intéressantes sont celles qui contiennent le plus d'éponges provenant de morphotypes différents. Le score d'intérêt se calcule de la manière suivante (Figure 25) :

$$score = \sum_{i=0}^N \sqrt{m_i} \quad \begin{array}{l} N : \text{nombre de morphotypes} \\ m_i : \text{nombre d'occurrence du } i^{eme} \text{ morphotype sur l'image} \end{array}$$

Figure 25 - Formule du score d'intéressement

Ce score est utilisé dans la génération du fichier PDF, qui est expliquée dans les parties 6.4.1 et 6.5.1.

6.4. Page « Historique »

La page « Historique » a pour but d'afficher l'aperçu des résultats de chaque analyse effectuée précédemment et de proposer l'exportation des données. La page est accessible à la fin d'une analyse ou à partir du menu. La Figure 26 donne un aperçu de cette page.

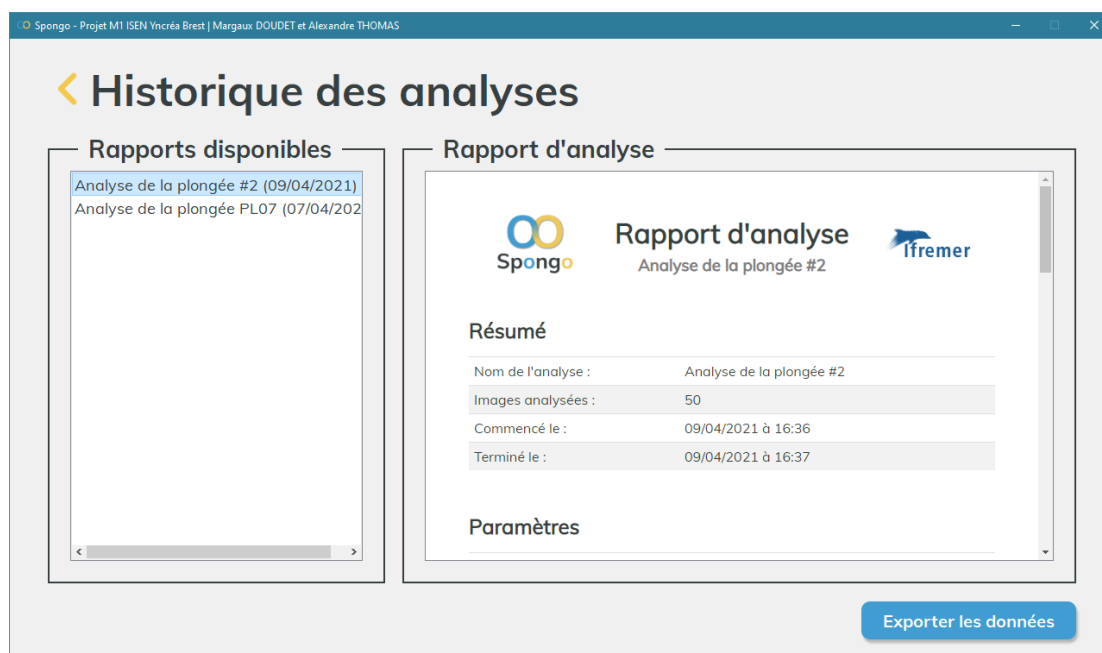


Figure 26 - Aperçu de la page "Historique"

6.4.1. Sauvegarde des analyses

L'objectif de l'historique est de permettre l'accès aux résultats des analyses précédemment réalisées. Il nous semblait en effet intéressant de disposer d'une page permettant de voir les résultats des analyses réalisées précédemment, et d'en exporter les données. Cela est d'autant plus vrai pour les analyses pouvant durer plus d'une heure, pour lesquelles ce n'était pas envisageable de les refaire depuis le début simplement pour revoir les résultats.

Pour cela, à la fin d'une analyse, les données sont sauvegardées dans un fichier JSON placé dans le dossier de l'application (par défaut : C:\Users\<username>\AppData\Roaming\Spongo). Ce fichier ne contient que les données liées à l'analyse ainsi que les 4 images considérées comme les plus intéressantes. Cela lui permet d'avoir une taille raisonnable d'environ 10Mo sur une analyse de 10 000 images. De plus, l'utilisateur peut à tout moment supprimer le fichier d'une analyse depuis l'historique.

6.4.2. Fonctionnalités de la page

La première chose que nous pouvons remarquer sur la page est l'aperçu du compte rendu de l'analyse. C'est une simple page HTML générées à partir des résultats de l'analyse. Elle permet de fournir un retour visuel de l'analyse sans avoir besoin de l'exporter.

Finalement, la principale fonctionnalité de la page est surtout l'exportation des résultats des analyses. En cliquant sur le bouton « Exporter les données », une fenêtre s'affiche permettant de présenter et de choisir parmi les différents formats de données proposés (PDF, CSV ou JSON entre autres). L'exportation des données est expliquée plus en détail dans la partie 6.5.

6.5. Exportation des données

Un des intérêts de l'application est de pouvoir exporter les résultats des analyses réalisées. Il nous semblait donc pertinent de proposer plusieurs « types » d'exportation différents afin de s'adapter aux besoins de l'Ifremer. Nous avons identifié 3 types d'exportation : les rapports résumés, les rapports détaillés et les annotations. Ces derniers sont présentés dans cette partie. Un aperçu de la page d'exportation est présenté sur la Figure 27.

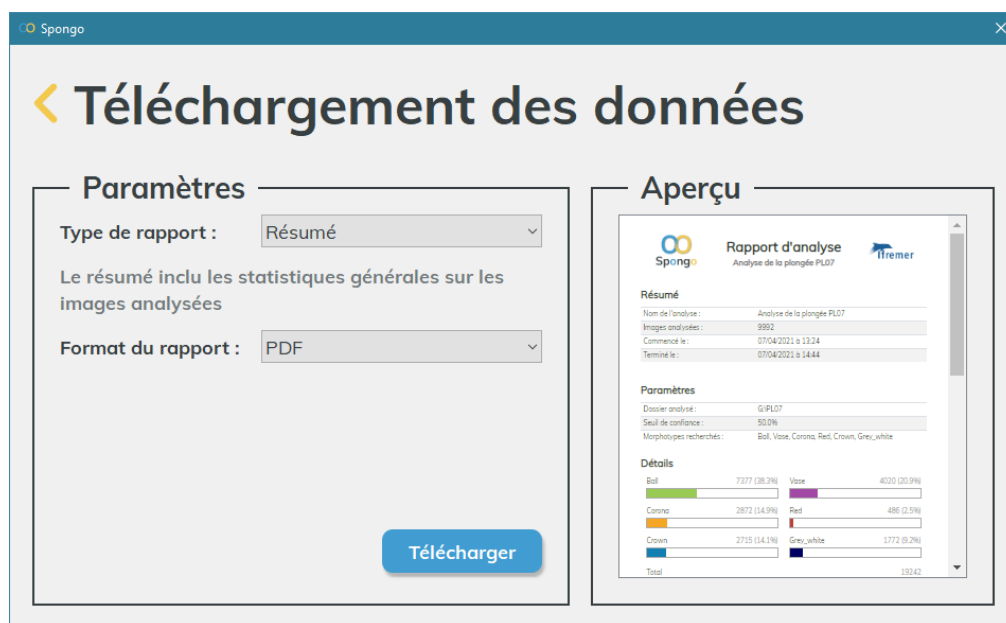


Figure 27 - Aperçu de la page "Exportation"

6.5.1. Rapports résumés

Comme leur nom l'indique, ces rapports ont pour but de donner un résumé général de l'analyse réalisée, avec des informations directement utilisables pour les chercheurs de l'Ifremer. Nous avons proposé deux formats pour ces rapports résumés : le PDF et le texte.

PDF

L'exportation au format PDF est la plus utile, le fichier donne les informations les plus importantes de l'analyse dans un format graphique et visuel. Un exemple de rapport PDF généré pour une analyse de 10 000 images est disponible en annexe.

La première partie « Résumé » présente les informations permettant de distinguer une analyse des autres (nom, nombre d'images, date, etc.). La seconde, « Paramètres », liste les paramètres définis pour cette analyse (seuil de détection, morphotypes sélectionnés, etc.).

Ensuite, la section « Détails » indique le nombre et le pourcentage de chaque morphotype. Cela permet d'avoir une idée des proportions de présence de chaque type d'éponges. Et dans un second temps, la partie « Détections cumulées » donne le même graphique que celui généré pendant l'analyse. Il permet de connaître la répartition des détections en fonctions des images.

Enfin, la dernière section « Images d'intérêts » présente les quatre images les plus intéressantes de l'analyse. Les images considérées comme les plus intéressantes sont celles avec le plus d'éponges provenant de morphotypes différents. Les chercheurs peuvent ainsi avoir un retour visuel concis de ce que le réseau a détecté.

Texte

Un rapport « résumé » peut aussi être exporté au format texte. Celui-ci est moins visuel que le PDF (il n'y a ni images ni graphiques), mais il est très léger, de l'ordre de quelques kilooctets. Celui-ci est utile si l'on souhaite garder les informations essentielles d'une analyse sans s'encombrer d'images.

6.5.2. Rapports détaillés

Les rapports détaillés ont été conçus pour donner un maximum d'informations sur l'analyse réalisée. Ils contiennent la liste de toutes les éponges détectées sur chaque image, avec leurs coordonnées en pixels. Ces types de rapports sont donc généralement peu lisibles. Cependant, ceux-ci ont surtout été pensés pour être utilisés par un autre logiciel. Nous proposons trois formats de données différents : CSV, JSON et XML. Tous sont très utilisés dans le traitement de données, mais chacun pour des utilisations différentes.

CSV

Le format CSV est le format avec lequel nous avons obtenu les annotations à l'origine. Il présente notamment l'intérêt d'être un format relativement simple. Chaque ligne du fichier décrit une détection, avec toutes les informations (nom de l'image, morphotype, etc.) séparées par un séparateur. Les fichiers CSV peuvent être lu par des tableurs, comme Excel.

Dans l'application, nous avons aussi laissé le choix du séparateur utilisé pour nous adapter à la syntaxe française de Excel, qui utilise un point-virgule, et à la syntaxe anglaise, qui utilise une virgule.

JSON et XML

Les formats JSON et XML ont pour objectif de représenter des données structurées. Ces derniers sont relativement peu lisibles et n'ont donc pas été conçus pour être directement utilisables par les chercheurs de l'Ifremer. Au contraire, nous avons fait le choix de proposer ce format au cas où quelqu'un voudrait développer un programme utilisant les données générées par notre application. En effet, ces formats standardisés sont très utilisés pour le transfert de données entre applications. Cette fonctionnalité a donc surtout été réalisée en perspective d'une potentielle utilisation future dans le cadre d'un autre projet.

6.5.3. Annotations

Enfin, l'exportation des annotations est le dernier format d'exportation que nous proposons. Contrairement aux deux autres, ce n'est pas un type de rapport mais une archive d'images annotées. Cette fonctionnalité a été réalisée à la fin du projet, à la suite d'une demande de l'Ifremer qui souhaitait pouvoir poursuivre l'apprentissage du réseau de neurones avec plus d'annotations. Le but étant de pouvoir améliorer le réseau en le réentraînant avec les éponges qu'il a lui-même détecté.

Le format « Annotations » permet donc d'obtenir une archive compressée contenant toutes les images ainsi que toutes les détections mises dans un certain format utilisable pour réentraîner le réseau. Les manipulations pouvant être techniques, nous avons rédigé un manuel pour l'Ifremer expliquant la marche à suivre pour réentraîner le réseau.

6.6. Autres fonctionnalités

Enfin, tout au long du projet nous avons développé quelques fonctionnalités mineures mais qui nous semblent importants d'évoquer.

6.6.1. Confort d'utilisation lors de l'analyse

Tout d'abord, nous avons intégré la progression de l'analyse sur l'icône de l'application (Figure 28) dans la barre des tâches Windows. Cela permet de visualiser où est rendu l'analyse même lorsque la fenêtre de l'application n'est pas ouverte. C'est une fonctionnalité que l'on retrouve sur de nombreuses applications Windows et qui nous semblait pertinent d'ajouter grâce aux outils offerts par la plateforme.



Figure 28 - Icône de l'application lors d'une analyse

Ensuite, nous souhaitons sécuriser l'utilisation de l'application. Ainsi, la boîte de dialogue ci-contre (Figure 29) apparaît lorsque l'utilisateur clique sur le bouton « Annuler » ou veut fermer l'application alors qu'une analyse est en cours. Elle permet de confirmer l'annulation de l'analyse afin de s'assurer que l'action est bien volontaire. Lorsque ce message est affiché, l'analyse continue en fond jusqu'à ce que l'utilisateur confirme son interruption.

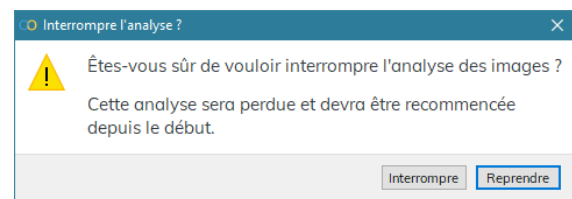


Figure 29 - Aperçu de la fenêtre de validation d'interruption de l'analyse en cours

6.6.2. À propos et licences

Finalement, il nous semblait indispensable de créer une fenêtre « À propos ». Elle est accessible depuis le menu et est divisée en deux parties. La première présente l'application dans sa globalité : les développeurs, les raisons de sa création, les objectifs et les technologies utilisées pour la développer. La seconde décrit toutes les informations nécessaires sur les dépendances utilisées, comme leur version, le site officiel sur lequel elles sont disponibles et leur licence. Cette seconde partie est essentielle. En effet, bien que toutes nos dépendances soient open-sources, certaines d'entre elles comme PySide2 demandent à être explicitement citées, et à ce que leur licence soit livrée avec l'application. Cette page nous assure donc d'être en conformité avec la loi et le souhait des auteurs. Les deux parties de cette fenêtre sont présentées sur la Figure 30 ci-dessous.

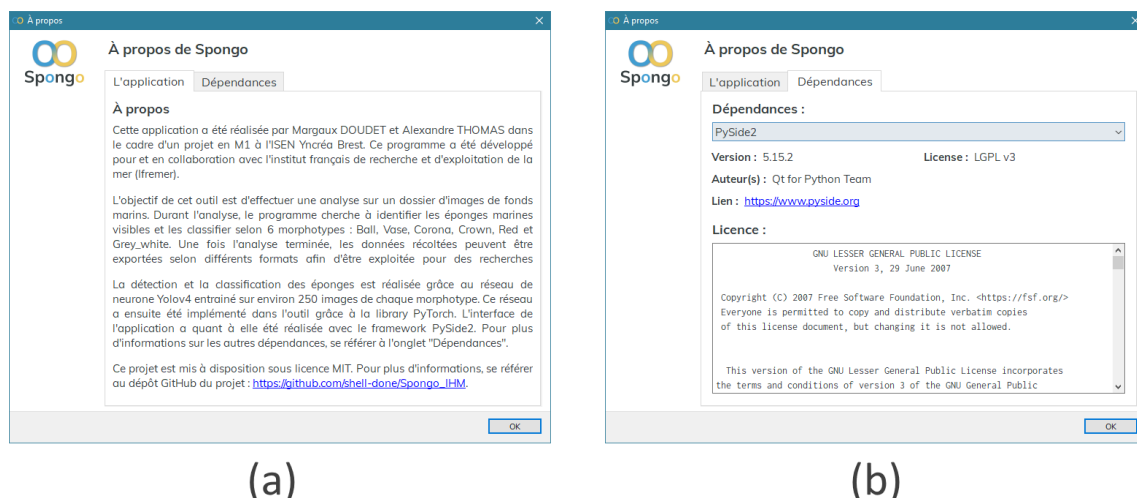


Figure 30 - Fenêtre « À propos » de l'application avec sa présentation (a) et la liste des dépendances (b)

6.7. Déploiement et distribution

6.7.1. Déploiement

Une fois l'application terminée de notre côté, nous devons la préparer pour que celle-ci soit facilement installable par les chercheurs de l'Ifremer. Notre objectif premier était que l'installation de l'application soit simple et ne demande que peu ou pas de connaissances en informatique. Pour ce faire, nous avons déployé l'application en deux étapes.

Tout d'abord, nous avons utilisé le programme PyInstaller qui permet de regrouper une application et toutes ses dépendances en un seul package. L'avantage principal est que l'utilisateur n'a pas à installer d'interpréteur Python ou de bibliothèques. Le dossier ainsi généré contient un exécutable du programme ainsi que toutes les dépendances nécessaires à son bon fonctionnement. Pour notre application, le package fait environ 3,6 Go, principalement à cause de PyTorch qui pèse à lui seul 2,9 Go.

Cependant, le dossier généré n'est pas idéal en termes de confort d'installation. En effet, il est relativement lourd et il contient de très nombreux fichiers, dont un seul est l'exécutable. Afin de résoudre ces problèmes, nous avons utilisé le logiciel InstallForge qui permet de créer des installateurs pour des logiciels. L'installation de l'application est alors similaire à celle des programmes que l'on peut trouver sur internet (Figure 31). L'installateur demande où installer les fichiers de l'application (par défaut : C:\Program Files (x86)\Spongo) et s'occupe lui-même de tout avant d'ajouter un raccourci sur le bureau et dans le menu « Démarrer ». Cette méthode a l'intérêt d'être extrêmement simple et peu propice aux erreurs étant donné que tout est automatisé. De plus, lors de la génération de l'installateur, les fichiers sont fortement compressés. Au final, l'exécutable d'installation ne pèse que 1,1 Go, soit 3 fois moins que le dossier original.

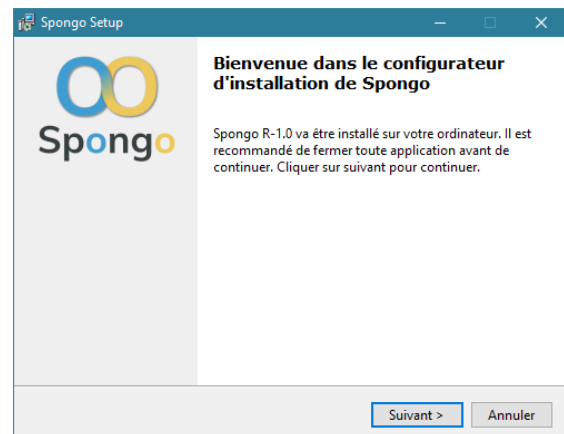


Figure 31 - Aperçu de l'installateur de l'application

6.7.2. Distribution

En plus de l'installateur, nous avons rédigé pour les chercheurs de l'Ifremer 3 guides d'utilisation :

- Le manuel d'installation : il explique la procédure pas à pas pour installer l'application et faire un premier test afin de vérifier que celle-ci fonctionne. En plus de la méthode « Avec l'installateur », deux autres méthodes d'installation sont présentées au cas où une erreur surviendrait.
- Le manuel d'utilisation : il détaille chaque partie de l'application afin que les chercheurs soient guidés dans leur première utilisation. Cela permet d'expliquer la signification des différentes informations présentes (notamment des paramètres) afin qu'ils sachent utiliser pleinement l'application.
- Le manuel de réapprentissage : il s'agit du manuel décrit dans la partie 6.5.3 et expliquant comment entraîner le réseau de neurones que nous avons utilisé. Les chercheurs peuvent ainsi, s'ils le souhaitent, réentraîner le réseau de neurones de l'application avec plus d'annotations pour tenter d'améliorer encore la qualité des détections.

Ces derniers ont ensuite été livrés avec l'installateur de l'application.

7. Gestion de projet

7.1. Approche agile

Durant le projet, nous avons tenté d'appliquer la méthodologie Agile dans notre travail. Le but était d'utiliser le planning prévisionnel réalisé au début comme une indication générale plutôt que comme une ligne directrice à suivre absolument. Au début de chaque semaine, nous définissions nos tâches respectives en fonction de l'avancée dans le projet. Le vendredi, nous faisions le point avec l'enseignant et les tâches qui n'avaient pas été terminées étaient déplacées à la semaine suivante.

L'intérêt de ce cycle de développement itératif est qu'il nous a permis de nous adapter aux difficultés que nous avons pu rencontrer pendant le projet. Certaines tâches ont en effet été beaucoup plus longues que prévu dans le planning. Par exemple, les tests des réseaux de neurones ont été deux fois plus longs qu'escomptés. En contrepartie, l'apprentissage d'un réseau étant plutôt long, cela nous a permis d'effectuer d'autres tâches en parallèle. Notre approche Agile du projet nous a permis de nous adapter face à ces imprévus.

Dans la même idée, nous avons développé l'application de manière incrémentale. Les différentes parties de l'interface ont par exemple été réalisées les unes après les autres et non pas parallèlement. Ce développement pas à pas avait plusieurs avantages. Tout d'abord, il permettait d'avoir chaque semaine une nouvelle version utilisable avec de nouvelles fonctionnalités. Les réunions hebdomadaires nous permettaient de présenter à l'enseignant cette nouvelle version et ainsi d'avoir une grande réactivité si quelque chose devait être changé. De plus, cela nous permettait de mieux nous rendre compte de ce qui risquait de prendre plus de temps que prévu, et de nous adapter en conséquence. Enfin, grâce à cette approche nous avons eu relativement peu de retouches et de modifications à faire lorsque le développement fut fini. Toutes les parties étaient fonctionnelles et avaient déjà été présentées et validées par l'enseignant.

7.2. Difficultés rencontrées

Durant le projet, nous avons rencontré plusieurs difficultés dont certaines ont été abordées précédemment dans ce rapport. Les principaux problèmes que nous avons eu étaient :

- **Les annotations :** Comme expliqué dans la partie 4.2, les annotations présentaient plusieurs problèmes. Elles étaient parfois mal placées sur les éponges, et certaines éponges n'étaient simplement pas annotées. Bien que ces problèmes étaient présents sur une minorité d'annotations, cela avait des impacts négatifs sur l'apprentissage du réseau de neurones. La solution la plus simple a été d'ajouter les annotations nous-mêmes, ce qui nous a pris environ 15 heures.
- **Google Colaboratory :** Il s'agit de la plateforme que nous avons utilisée pour entraîner et tester nos réseaux de neurones. Le service nous a beaucoup aidé en nous permettant d'exécuter le code sur des serveurs avec de bonnes cartes graphiques. Cependant, le service étant gratuit, nous étions régulièrement déconnectés par la plateforme afin que les serveurs soient utilisables par d'autres personnes. Nous avons donc dû nous adapter afin d'étaler les apprentissages sur une plus longue durée.
- **Multithreading :** Une des difficultés majeures de l'IHM a été de la rendre la plus fluide possible. Certaines opérations, comme l'utilisation du réseau de neurones ou l'enregistrement de l'analyse sont assez longs et bloquants. C'est-à-dire que lors de l'appel à ces fonctions, l'interface se bloquait complètement jusqu'à ce que la fonction soit terminée. Afin de pallier ce problème, nous avons utilisé des threads pour exécuter le code « lourd » en parallèle et ainsi ne pas bloquer l'interface.

7.3. Finalisation du projet

À partir de la semaine 14, soit 3 semaines avant le rendu, nous avons commencé à anticiper la fin du projet afin de s'assurer de tout terminer à temps.

L'application étant quasiment finie, nous avons proposé une réunion avec l'enseignant et l'Ifremer afin de faire une démonstration de l'application dans sa version presque finale. Cela nous a permis d'avoir de nombreux retours et de répondre aux questions des chercheurs. Cela nous a ensuite permis de faire des retouches mineures sur l'application afin de prendre en compte leurs remarques.

Nous avons aussi commencé à lister et résoudre tous les bugs présents. Pour cela nous avons effectué des tests variés avec différents cas d'utilisation : différents environnements, différents formats, différentes images avec ou sans détection. Au total, cette phase de test nous a permis de trouver et résoudre une cinquantaine de bugs. Enfin la dernière étape du projet en lui-même a été de préparer son déploiement pour l'Ifremer. Nous avons donc réalisé l'installateur et rédigé les différents guides de l'application.

Ainsi, nous avons terminé la réalisation du projet en fin de semaine 14. Après vérification du cahier des charges, seule une fonctionnalité n'est pas remplie à 100%. En effet, notre application ne permet pas de détecter un des six morphotypes de base : Yellow. Cela est principalement dû au fait qu'avec seulement 109 annotations fournies pour cette classe, il était impossible d'entraîner le réseau de neurones correctement. Cependant, en contrepartie nous détectons le morphotype Corona qui lui avait assez d'annotations. Toutes les autres fonctionnalités (de tous les niveaux de priorité) ont quant à elles été implémentées.

Enfin, la réalisation étant terminée dans les temps, nous avons gardé les semaines 15 et 16 pour rédiger ce rapport et la semaine 17 pour nous préparer à la soutenance.

7.4. Disponibilité du projet et licence

Aujourd'hui, le projet est accessible à tous sur la plateforme GitHub à l'adresse ci-dessous. Nous avons décidé d'y appliquer une licence MIT. Cela signifie que tout le monde peut télécharger, réutiliser ou modifier notre projet librement (sous réserve de citation). Ainsi, d'autres personnes peuvent, s'ils le souhaitent, reprendre notre code source et y apporter des modifications afin de l'adapter à leur utilisation.

Adresse du dépôt GitHub : <https://github.com/shell-done/Spongo>

8. Conclusion

8.1. Retour sur le projet

À travers ce projet, nous avons développé une application permettant de détecter et classifier certains morphotypes d'éponges marines profondes à partir d'images. Ce projet a été initié par l'Ifremer dans le cadre de recherches scientifiques pour aider à l'étude et à la reconnaissance de la biodiversité marine présente dans les fonds marins.

Pour mener à bien ce projet, nous avons tout d'abord commencé par trier et traiter les données fournies afin d'éliminer les annotations non-exploitable. En accord avec les chercheurs de l'Ifremer, nous avons ensuite sélectionné les morphotypes sur lesquels nous allions entraîner le réseau de neurones. Suite à cela, nous avons gardé 6 morphotypes différents : Ball, Vase, Corona, Grey_white, Crown et Red. Par la suite, nous avons fait des recherches et conduit différentes expérimentations sur des réseaux de neurones et leurs paramètres afin de déterminer lequel serait le plus adapté à notre problématique. À partir de ces expérimentations, nous avons choisi l'architecture YOLO et plus particulièrement la version YOLOv4. Ce choix a été motivé par sa rapidité d'exécution ainsi que la qualité de ses prédictions. Enfin, la dernière étape a été de développer une IHM qui intégrait ce réseau de neurones pour proposer une interface utilisable par l'Ifremer. Cette dernière a été réalisée en Python en attachant une grande importance au confort d'utilisation. De plus, l'utilisation de la technologie CUDA nous a permis d'atteindre des performances très intéressantes avec un temps de traitement par image de l'ordre de la demi-seconde. Pour finir, nous avons préparé l'application afin que celle-ci soit performante sur tous les environnements et installable facilement par n'importe qui.

8.2. Perspectives possibles

Bien que le projet réponde au cahier des charges, nous avons établi quelques points qu'il pourrait être intéressant d'explorer afin d'améliorer l'application :

- **Tester d'autres réseaux**, notamment Faster R-CNN que nous avons rapidement mis de côté pour nous focaliser sur YOLO en raison du temps limité. Il serait intéressant de le tester afin d'évaluer ses performances comparées à notre réseau actuel et pour, pourquoi pas, le remplacer si ses résultats sont meilleurs.
- **Augmenter le nombre d'annotations** au jeu de données d'entraînement. C'est un point sur lequel nous étions plutôt limités. Ajouter des annotations permettrait un meilleur entraînement du réseau de neurones et donc de meilleures détections. C'est d'ailleurs une fonctionnalité que nous avons déjà préparée et qui facilement réalisable grâce à l'export d'annotations YOLOv4 que nous avons mis en place à travers l'application.
- **Ajouter d'autres morphotypes** possiblement détectables par le réseau. Aujourd'hui, nous nous sommes limités à 6 morphotypes au vu des annotations à notre disposition. Cependant, cette valeur n'est pas limitée et il serait possible d'adapter l'application pour détecter plus de morphotypes.
- **Exploiter l'application pour d'autres situations que les éponges marines**. Dans la même logique que le point précédent, il est également envisageable d'entraîner le réseau sur d'autres espèces marines comme des coraux ou des poissons par exemple.

Nous avons également conçu l'application de manière qu'elle ne soit pas figée pour une seule et même tâche. Elle a été développée pour être évolutive et modulable. Ainsi, une personne avec suffisamment de connaissances en informatique peut tout à fait reprendre le code source et l'adapter afin d'y implémenter les propositions précédentes.

9. Bibliographie

- [1] T. Mitchell, Machine Learning, New York: McGraw Hill, 1997.
- [2] Wikipédia, «Apprentissage automatique», 10 avril 2021. [En ligne]. Available: https://fr.wikipedia.org/wiki/Apprentissage_automatique. [Accès le 12 avril 2021].
- [3] Wikipédia, «Apprentissage profond», 10 avril 2021. [En ligne]. Available: https://fr.wikipedia.org/wiki/Apprentissage_profond. [Accès le 12 avril 2021].
- [4] Wikipédia, «Réseau neuronal convolutif», 21 février 2021. [En ligne]. Available: https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif. [Accès le 12 avril 2021].
- [5] Wikipédia, «Convolutional neural network», 1 avril 2021. [En ligne]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network. [Accès le 12 avril 2021].
- [6] Google, «Travaux pratiques sur le machine learning : Classification d'images», 2021. [En ligne]. Available: <https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks?hl=fr>.
- [7] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick et P. Dollár, «Microsoft COCO: Common Objects in Context», *arXiv:1405.0312* (available at <https://arxiv.org/abs/1405.0312>), 2015.
- [8] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama et K. Murphy, «Speed/accuracy trade-offs for modern convolutional object detectors», *arXiv:1611.10012* (available at <https://arxiv.org/abs/1611.10012>), 2017.
- [9] S. A. Sánchez, H. J. Romero et A. D. Morales, «A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework», *IOP Conference Series: Materials Science and Engineering* (available at <https://iopscience.iop.org/article/10.1088/1757-899X/844/1/012024>), vol. 844, n°012024, 2020.
- [10] J. Redmon et A. Farhadi, «YOLOv3: An Incremental Improvement», *arXiv:1804.02767* (available at <https://arxiv.org/abs/1804.02767>), 2018.
- [11] A. Bochkovskiy, C.-Y. Wang et H.-Y. M. Liao, «YOLOv4: Optimal Speed and Accuracy of Object Detection», *arXiv:2004.10934* (available at <https://arxiv.org/abs/2004.10934>), 2020.
- [12] Q. Nguyen, «YOLOv4 on Google Colab: Train your Custom Dataset (Traffic signs) with ease», 23 juillet 2020. [En ligne]. Available: <https://towardsdatascience.com/yolov4-in-google-colab-train-your-custom-dataset-traffic-signs-with-ease-3243ca91c81d>. [Accès le 15 avril 2020].
- [13] P. K. Sahu, «Training a Custom Object Detection Model With Yolo-V5», 1 Janvier 2021. [En ligne]. Available: <https://medium.com/analytics-vidhya/training-a-custom-object-detection-model-with-yolo-v5-aa9974c07088>. [Accès le 15 avril 2021].
- [14] S. Yohanandan, «mAP (mean Average Precision) might confuse you!», 9 janvier 2020. [En ligne]. Available: <https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2>. [Accès le 16 avril 2021].
- [15] Wikipédia, «Matrice de confusion», 21 novembre 2020. [En ligne]. Available: https://fr.wikipedia.org/wiki/Matrice_de_confusion. [Accès le 18 avril 2021].
- [16] Wikipédia, «CUDA», 16 avril 2021. [En ligne]. Available: <https://en.wikipedia.org/wiki/CUDA>. [Accès le 17 avril 2021].
- [17] Tianxiaomo, «Pytorch-YOLOv4», 2 septembre 2020. [En ligne]. Available: <https://github.com/Tianxiaomo/pytorch-YOLOv4>. [Accès le 16 avril 2021].

10. Annexes

10.1. Comparaison de la maquette originale et de l'application finale

Page « Menu »

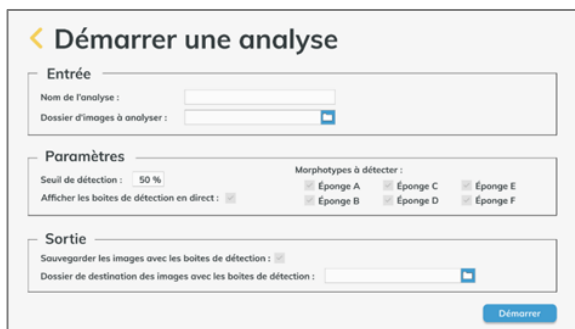


Maquette originale

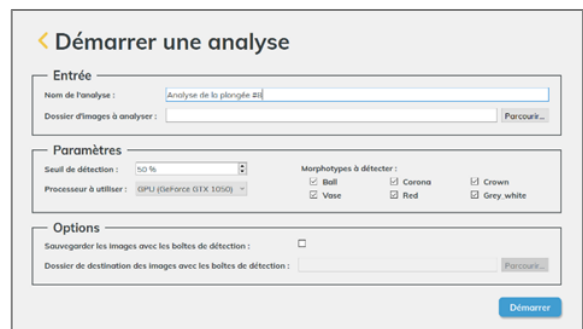


Application finale

Page « Paramètres »

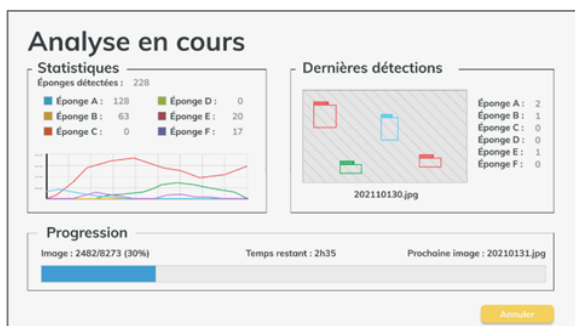


Maquette originale

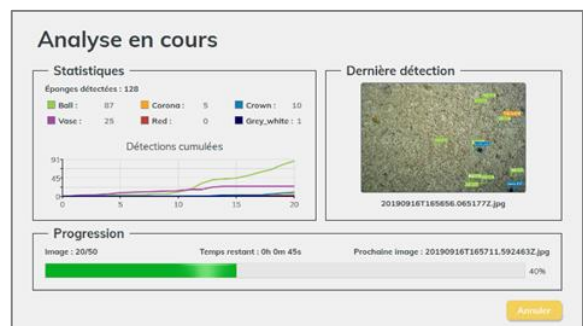


Application finale

Page « Analyse »

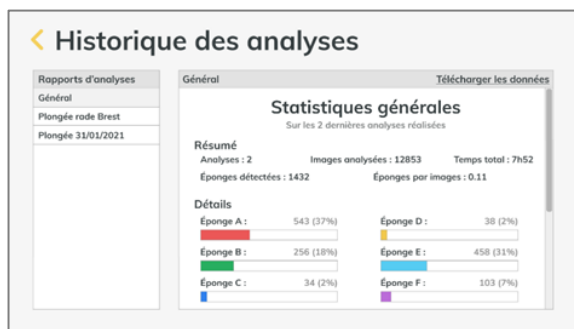


Maquette originale

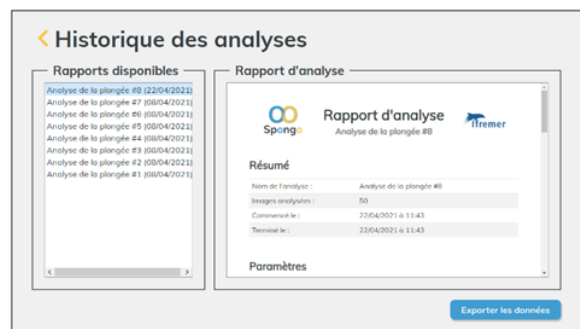


Application finale

Page « Historique »



Maquette originale



Application finale

Fenêtre « Exportation des données »

Maquette originale de la fenêtre « Télécharger les données ». Elle est structurée en deux colonnes principales.

- Section « Paramètres » :**
 - Type de rapport : Sélectionné « Résumé ».
 - Format du rapport : Sélectionné « CSV ».
 - Chemin de destination : Champ de texte avec un bouton de sélection de dossier.
- Section « Aperçu » :** Espace réservé pour un aperçu du contenu du rapport.
- Barre d'action (en bas) :** Bouton « Télécharger ».

Maquette originale

Application finale de la fenêtre « Téléchargement des données ». Elle intègre des éléments de design plus sophistiqués.

- Section « Paramètres » :**
 - Type de rapport : Sélectionné « Résumé ».
 - Format du rapport : Sélectionné « PDF ».
- Section « Aperçu » :** Affiche un aperçu visuel du rapport à télécharger.
- Barre d'action (en bas) :** Bouton « Télécharger ».

Application finale

10.2. Exemple de rapport résumé en PDF



Rapport d'analyse

Analyse de la plongée PL07



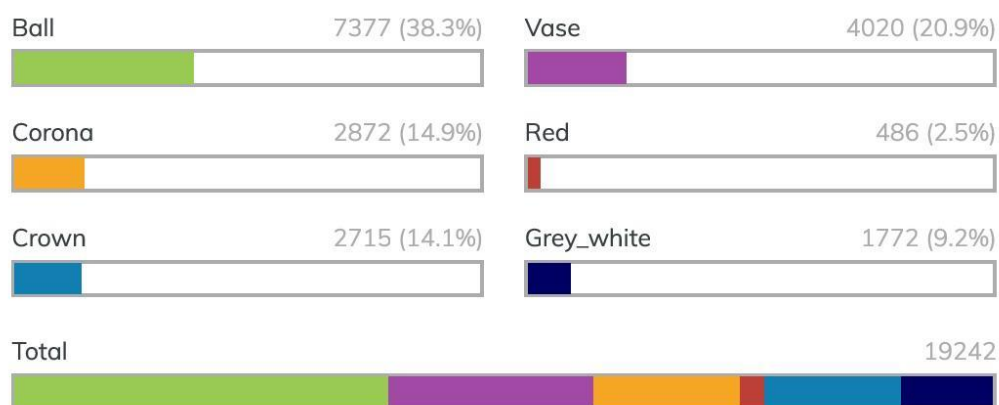
Résumé

Nom de l'analyse :	Analyse de la plongée PL07
Images analysées :	9992
Commencé le :	07/04/2021 à 13:24
Terminé le :	07/04/2021 à 14:44

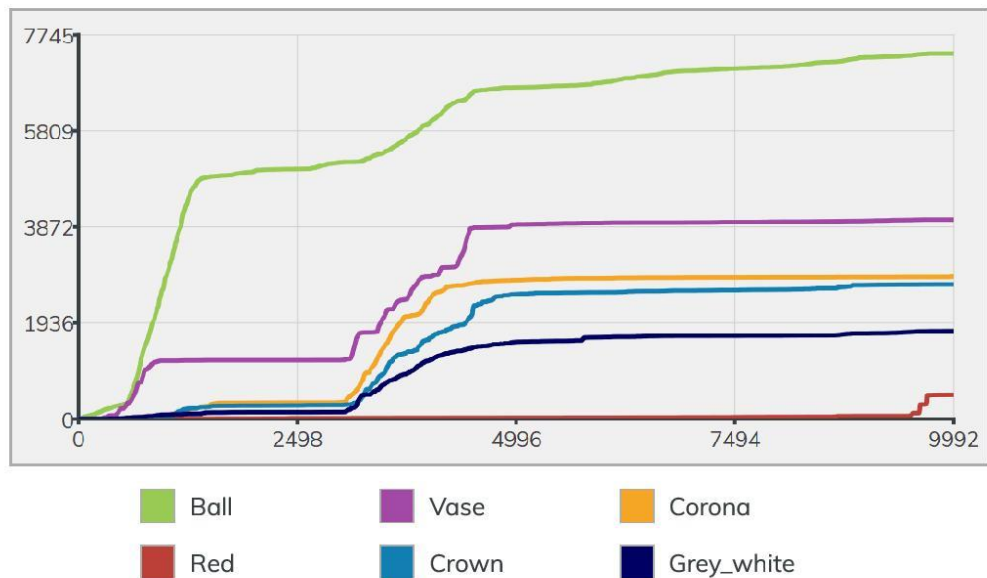
Paramètres

Dossier analysé :	G:\PL07
Seuil de confiance :	50.0%
Morphotypes recherchés :	Ball, Vase, Corona, Red, Crown, Grey_white

Détails



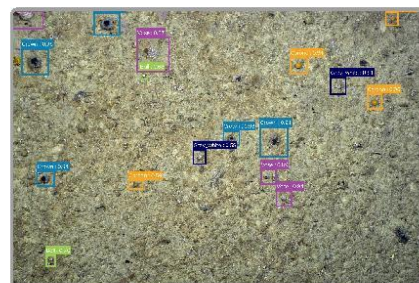
Détections cumulées



Images d'intérêt



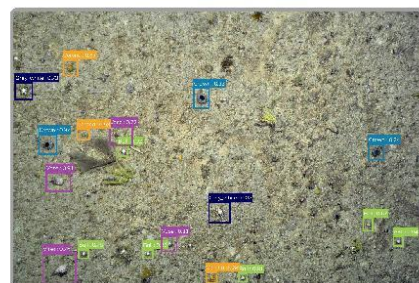
20190916T201810.081934Z.jpg



20190916T201949.103324Z.jpg



20190916T204749.624731Z.jpg



20190916T204752.621405Z.jpg

Analyse de la plongée PL07

Page 2/2