

Entraînement de YOLOX pour la Détection de Cibles Maritimes

Auteur :
BOIDIN Benoît

Client :
MAXSEA INTERNATIONAL
Référent :
GOHLEN Ronan

Remerciements à mon maître de stage Ronan Golhen pour m'avoir donné l'opportunité de travailler chez MaxSea International et m'avoir encadré tout au long du stage, à ma tutrice Zoé Varin pour son aide et son soutien, à mon collègue Victor Opter pour son accueil et ses conseils, et à toute l'équipe de MaxSea International pour leur bienveillance.

Résumé

Le domaine maritime, avec tous les enjeux qu'il comporte, présente un grand nombre de problématiques qui peuvent être résolues par les nouvelles technologies. On trouve parmi celles-ci le routage, l'exploration des fonds marins ou encore la surveillance.

MaxSea International, en association avec Furuno, fait partie des acteurs qui fournissent des solutions innovantes aux professionnels de ce domaine. L'entreprise, localisée à Bidart dans le Pays Basque, compte 70 employés dont une majorité de développeurs. Elle est à l'origine des logiciels de la gamme TimeZero, qui comptent, entre autres :

- **TZ Professional**, qui permet le routage des navires de pêche et commerciaux, l'analyse de la bathymétrie ou encore la gestion d'appareils d'acquisition ;
- **TZ Maps**, une collection de cartes précises aux formats vectoriel et raster dans le monde entier ;
- **TZ iBoat**, l'application iOS pour créer des itinéraires de plaisance.

Le logiciel qui bénéficiera de notre travail est **TZ Coastal Monitoring**, destiné aux ports et zones industrielles côtières. Il permet la surveillance des navires par différents moyens, et contient un module de gestion des caméras de surveillance.

Les principaux éléments permettant de détecter un bateau sont le radar et l'AIS. Le radar fonctionne grâce à des ondes radio courtes (source : Furuno Radar) et permet de détecter toutes sortes d'objets, en particulier des bateaux. L'AIS, acronyme de Automatic Identification System (source : IMO AIS) est un système de balise embarquée permettant d'éviter les collisions entre les bateaux, et d'indiquer des informations aux infrastructures côtières. Ces deux systèmes, bien qu'étant performants, ont des défauts indéniables. Le radar par exemple, n'apporte pas d'information précise sur la nature de l'objet détecté, souffre d'une latence importante et est sensible aux conditions météo. L'AIS, quant à lui, peut facilement être désactivé par l'équipage. C'est pour cela que MaxSea International s'intéresse à la détection automatique d'objets, en particulier par réseaux de neurones. Un tel système permettrait de nombreuses fonctionnalités nouvelles, en profitant du matériel déjà présent, donc à moindre coût pour les clients ; pilotage automatique des caméras, alertes en temps réel, ou encore enregistrement d'images précises lors d'incidents.

Comme il n'existe pas de proposition spécifiquement conçue pour le métier maritime, nous avons décidé de construire notre propre solution, afin qu'elle soit parfaitement adaptée. Pour accomplir cette mission, nous avons mis en place un pipeline de machine learning qui permet d'aller de la collecte de datasets jusqu'à la production d'un modèle optimisé, prêt à être utilisé. Concernant le matériel, nous avions à notre disposition un poste fixe fonctionnant sous Windows, sur lequel nous avons installé WSL2 afin de profiter de l'environnement de travail de Linux. Depuis ce poste, nous avions accès à deux machines distantes équipées de cartes graphiques. Les solutions apportées par le stage sont une base de données de plus de 390 000 images de bateaux, des systèmes de preprocessing, un environnement d'entraînement du modèle YOLOX, et différents modèles optimisés pour les processeurs cibles.

Les chapitres suivants présentent en détails les éléments précédemment mentionnés, le travail réalisé et les conclusions auxquelles nous avons abouti.

Table des matières

1	Introduction et contexte	1
1.1	Présentation de l'entreprise	1
1.2	Travail de l'ancien stagiaire	2
2	Cahier des charges	3
2.1	Précision de la détection	3
2.2	Vitesse de traitement	3
2.3	Visualisation	3
2.4	Facilité d'utilisation	3
3	Méthodes et outils	4
3.1	Matériel	4
3.2	Planning du stage	4
3.3	Gestion du projet	6
3.4	Organisation de l'équipe	7
4	Travail réalisé	8
4.1	Création de l'environnement virtuel	8
4.2	Documentation initiale	8
4.3	Datasets	10
4.4	Prise en main des outils	11
4.5	Développement du pipeline d'entraînement	15
4.6	Prétraitement	16
4.7	Entraînements	19
4.8	Inférence	20
5	Résultats	23
5.1	Tests	23
5.2	Tuilage	23
5.3	Entraînement avec images vides	24
5.4	Scinder la classe boat	24
5.5	Image facile à identifier	26
5.6	Résultats précédents	26
6	Conclusions	28
6.1	Solutions retenues	28
6.2	Développement durable	28
7	Perspectives	29
	Appendices	33

Chapitre 1

Introduction et contexte

1.1 Présentation de l'entreprise

Il y a 35 ans, Brice Pryszo a fondé MaxSea International et a créé le premier logiciel de navigation embarqué permettant de stocker des cartes marines sur un ordinateur. Depuis, l'entreprise n'a cessé de proposer des solutions toujours plus innovantes pour les professionnels de la mer, dans plus de 25 pays. Ses clients aussi bien les plaisanciers que les pêcheurs ou la marine marchande.

Elle propose notamment les solutions de navigation TimeZero Navigator (destiné aux plaisanciers) et TimeZero Professional (destiné aux professionnels de la mer), appuyées par TimeZero Maps, une cartographie marine de haute qualité en raster et, depuis peu, en vecteur. Ces cartes sont également accessibles sur l'application iOS TimeZero iBoat. Tous ces produits profitent également de leur service météo, qui donne accès à des précisions météorologiques très complètes, fournies par les modèles météo mondiaux les plus fiables.

Tous ces logiciels s'appuient sur des appareils d'acquisition haut de gamme, plus particulièrement ceux fabriqués par Furuno, partenaire principal de la société depuis 2007. Cette collaboration bénéficie également à TimeZero Coastal Monitoring, qui permet de surveiller les zones côtières.

C'est sur ce dernier produit que j'ai effectué mon stage de fin d'études, au sein de l'équipe de recherche et développement. Plus particulièrement, j'ai développé une fonctionnalité de détection de navires, qui permettra de compléter les informations acquise par le radar et l'AIS, dont on peut voir un exemple sur la figure ci-après 1.1.

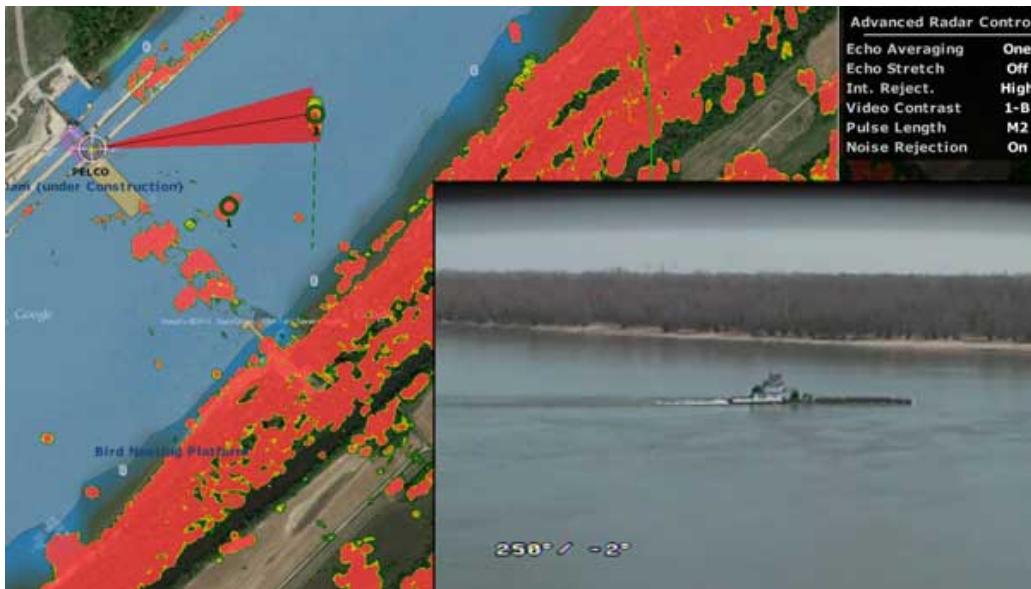


FIGURE 1.1 – Exemple de vue radar, AIS et camera dans TimeZero Coastal Monitoring

L'équipe que j'ai rejoint était composée de 2 personnes, dont Ronan Golhen, mon tuteur de stage et directeur technique, et Victor Opter, développeur logiciel. Le reste de l'équipe n'étant pas initié aux techniques d'apprentissage machine, j'ai travaillé en autonomie sur le projet, tout en profitant de la grande expertise métier de mon tuteur, qui connaît non seulement la partie logiciel, mais qui a également une grande expérience en navigation.

1.2 Travail de l'ancien stagiaire

Ce projet a commencé en 2023, avec un premier stagiaire qui a travaillé sur la détection de navires.

Ses réalisations comprenaient tout d'abord le choix d'un modèle de machine learning nommé YO-LOX. Ce choix a été guidé par la recherche de performances, aussi bien du côté de la précision que de celui de la rapidité, mais également par des contraintes légales. Ce modèle étant sous licence Apache 2.0, il est possible de l'utiliser de façon commerciale.

Après avoir choisi ce modèle, il a été décidé qu'une phase d'entraînement était nécessaire. Il a donc commencé à récolter des datasets, réunissant environ 8000 images de navires. Trois entraînements ont été réalisés en utilisant des machines virtuelles sur le cloud.

Les travaux réalisés comportaient néanmoins quelques limites. Entre autres, les scripts contenait un grand nombre de chemins d'accès à des fichiers non relatifs, ce qui rendait la reprise du projet difficile. De plus, la documentation très succincte ne permettait pas de connaître les paramètres exacts utilisés pour l'entraînement. De plus, l'achat de matériel spécialisé a rendu obsolète le code relatif à l'utilisation du cloud.

Après avoir pris connaissance de ces travaux, j'ai donc entrepris d'utiliser les outils choisis l'année dernière, en écrivant des scripts plus modulaires et en documentant plus précisément les étapes de l'entraînement.

Chapitre 2

Cahier des charges

L'objectif principal du stage était de créer un système de détection de navires pour enrichir la fonction de vidéo-surveillance de Coastal Monitoring. Cet objectif peut être décomposé en plusieurs éléments :

2.1 Précision de la détection

La précision est le point principal de ce projet. Il est important que, lorsque le système détecte un objet, cet objet soit en effet un navire : du point de vue du client, les fausses alertes entraînent grandement la qualité perçue, même si elles sont rares. Il faut donc que le modèle soit le plus "prudent" possible, tout en étant quand même capable de détecter les navires.

2.2 Vitesse de traitement

La vitesse de traitement est un autre point important. Le système doit être capable de détecter les navires en temps réel, c'est-à-dire que le temps de traitement d'une image doit être assez rapide pour être utile en vidéo (environ 30 images par seconde). Il est nécessaire de préciser que ces performances doivent être réalisées sur des machines similaires à celles utilisées par les clients, c'est à dire sans carte graphique dédiée, et avec des processeurs n'était pas nécessairement de dernière génération.

2.3 Visualisation

Pour permettre l'analyse qualitative des résultats, il est nécessaire de pouvoir visualiser, non seulement les résultats de la détection, mais aussi les datasets qui serviront à entraîner. Pour ce faire, j'ai utilisé le logiciel FiftyOne tout au long du développement.

2.4 Facilité d'utilisation

MaxSea ne comptant pas de développeur familier avec ces technologies, il est important que le système soit facile à utiliser et à maintenir. J'ai pour cela utilisé, comme le stagiaire précédent, des Jupyter Notebooks. Ce projet n'étant pas encore destiné à être intégré dans le produit final, les notebooks sont un bon compromis entre ergonomie et rapidité de mise en place. Ceux-ci permettent d'exécuter pas à pas des scripts Python, et la recherche de bug est plus aisée.

Chapitre 3

Méthodes et outils

3.1 Matériel

Le matériel fourni pour le stage comprenait un ordinateur de bureau sous Widnows, qui permettait de se connecter à deux machines distantes servant aux entraînement : la première était dotée d'une carte graphique NVIDIA RTX 4060, et la seconde d'une RTX 4070 Ti Super (plus puissante). Ces machines étaient accompagnées d'un accès à un serveur de stockage de données.

J'ai choisi l'éditeur VS Code pour le développement, et Git pour la gestion de version. Git m'a permis de synchroniser toutes les machines, et de créer des branches pour l'ajout de nouvelles fonctionnalités, ce qui a eu comme bénéfice de toujours conserver une version stable pour les entraînements. Enfin, le code était stocké sur GitHub pour faciliter le partage et la récupération du travail en cas de perte.

3.2 Planning du stage

3.2.1 Datasets

La première étape pour améliorer le système était d'entraîner le réseau de neurones YOLOX sur un datasets plus important. La début du stage était donc consacré à la recherche d'image de bateaux. L'entraînement de ce modèle nécessite un dataset au format COCO¹; ces recherches sont donc accompagnées du développement de scripts de conversion pour les datasets qui ne sont pas au bon format.

3.2.2 Prise en main des outils

En parallèle des tâches mentionnées précédemment a eu lieu la prise en main des outils choisi l'année dernière.

La création de l'environnement de développement a commencé par l'installation de WSL²

1. Le format COCO (Common Objects in Context) est un format d'annotation d'images très utilisé dans le domaine de la détection d'objets. Il s'agit d'un fichier json qui accompagne les photos.

2. WSL (Windows Subsystem for Linux) est un outils permettant d'utiliser Linux sur une machine Windows

puis des drivers CUDA³, et enfin des librairies nécessaires à YOLOX et FiftyOne. Ce dernier outil a été particulièrement difficile à prendre en main, car certaines fonctions ne renvoient que parfois des messages d'erreur lorsqu'elles sont mal utilisées.

Des recherches ont été nécessaires pour comprendre le fonctionnement précis de YOLOX, notamment les paramètres disponibles et les méthodes de data augmentation⁴ intégrées.

FiftyOne étant un outil très complet, il a fallu apprendre à l'utiliser pour connaître les fonctionnalités disponibles, et en tirer le meilleur parti.

J'ai documenté toutes ces étapes afin que mon travail puisse être repris par un autre développeur si nécessaire.

3.2.3 Travail de recherche

En plus de l'augmentation du dataset, nous avons identifié plusieurs points à améliorer :

- la qualité du dataset ;
- le choix des paramètres d'entraînement ;
- les optimisations post entraînement ;

Pour cela, je me suis basé sur mes connaissances acquises lors de mon master (notamment grâce aux cours de deep learning et de traitement d'image), ainsi que sur des recherches et des expérimentations. Ces dernières ont permis de mettre à jour des caractéristiques inhérentes à la détection de bateaux.

Pour être le plus rigoureux possible, j'ai proposé à l'équipe de procéder de la manière suivante : pour tester une hypothèse (par exemple, l'effet de la data augmentation), j'effectue deux entraînements avec une sous partie du dataset pour réduire le temps de calcul et apporter une première réponse rapidement. Le premier correspond à H_0 , c'est à dire l'hypothèse nulle, et le second à H_1 , l'hypothèse alternative qui correspond à notre tentative d'amélioration.

Ceci nous a permis d'isoler les variables et de valider ou d'invalider rapidement des hypothèses.

Après avoir optimisé les entraînements et donc le modèle, nous avons cherché d'autres moyens, applicables en production, pour rendre la détection plus efficace.

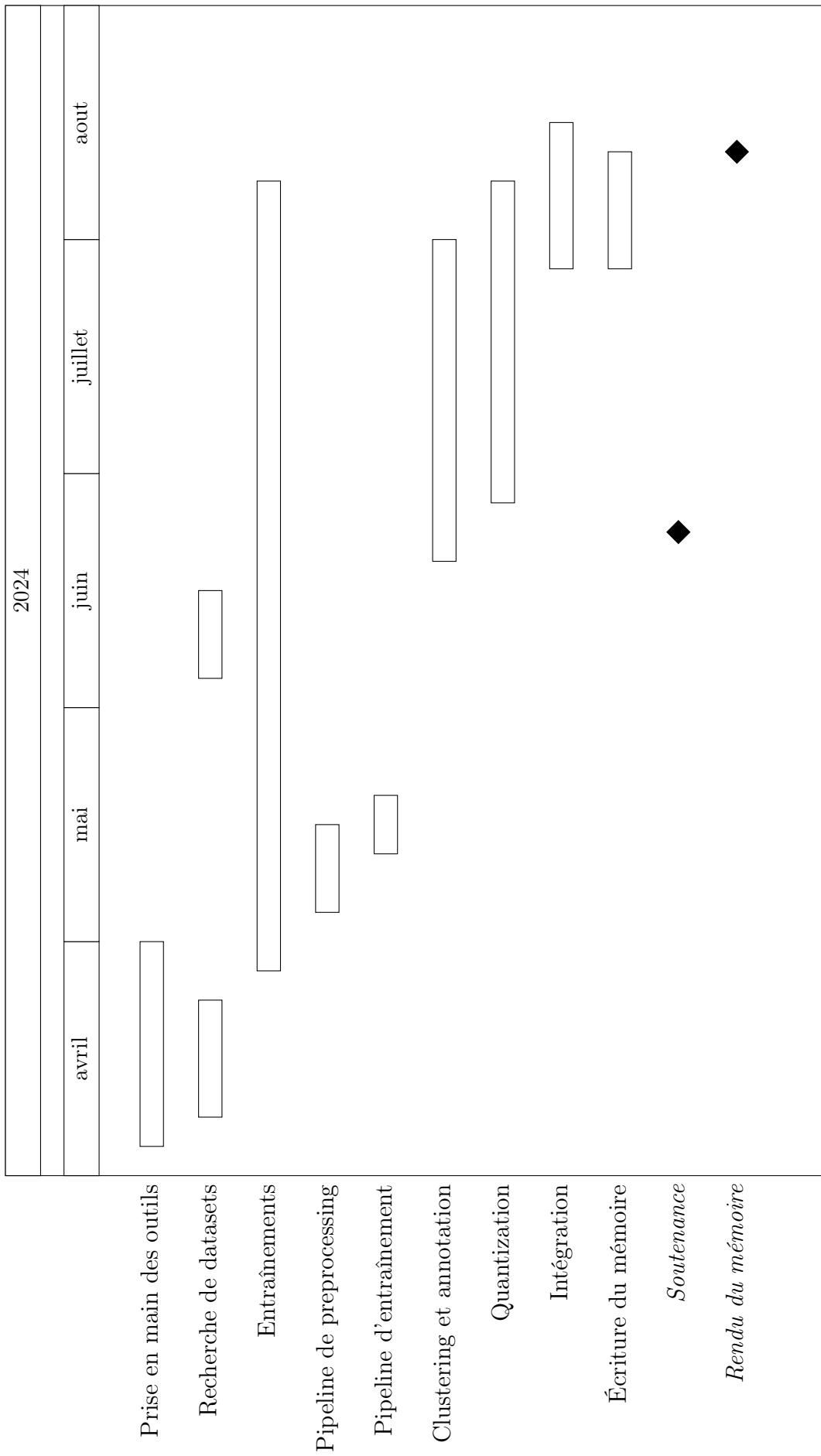
3.2.4 Diagramme de Gantt

La répartition du temps de travail correspondant aux tâches décrites ci-dessus est décrite par le diagramme de Gantt en page suivante.

3. CUDA est l'API utilisée par les cartes graphiques NVIDIA pour bénéficier de la puissance de calcul parallèle de leur carte graphique

4. La data augmentation est une technique qui consiste à dupliquer puis modifier les images du dataset pour augmenter sa taille et améliorer la généralisation du modèle.

3.3 Gestion du projet



3.4 Organisation de l'équipe

L'entreprise MaxSea international utilise les services Microsoft, en particulier OneNote, qui m'a servit à partager des informations avec le reste de l'équipe, et OneDrive, pour le partage de fichiers.

Pour l'organisation du temps de travail, une réunion SCRUM a lieu tous les matins à 9 :30 avec tous les membres de l'équipe. Nous en profitons pour partager les tâches réalisées la veilles, et les objectifs de la journée. Pour appuyer cette réunion, nous utilisons l'outil Trello⁵.

Mon travail étant encore à un stade de recherche, je n'ai pas été soumis par l'entreprise aux tests unitaires, ni à des conventions de nommage de variables ou autres contraintes de génie logiciel.

5. Trello est un outil permettant d'incarner le système des "sprints", et dans lequel un objectif est représenté par une carte qui contient plusieurs tâches à réaliser.

Chapitre 4

Travail réalisé

Ce chapitre présente le travail réalisé durant le stage. Toutes les tâches relatives au planning décrit précédemment ont pu être effectuées.

4.1 Création de l'environnement virtuel

Pour la création de l'environnement virtuel, nous nous sommes d'abord tourné vers Anaconda. Bien que très puissant, cet outil est assez lourd et il est complexe de lister des dépendances sans préciser les versions, ce qui nous a posé des problèmes de compatibilité entre les différentes librairies. Nous avons donc décidé de créer un simple fichier `requirements.txt` qui liste toutes les dépendances, et permet de les installer en utilisant la commande `pip install -r requirements.txt`.

Ce fichier a permis de porter l'environnement de développement sur toutes les machines à ma disposition, et permettra au prochain développeur de rapidement travailler sur le projet.

4.2 Documentation initiale

Étant la seule personne familière avec les techniques de machine learning, j'ai pris soin de documenter et présenter à l'équipe les différents concept utilisés durant le stage. Cette section présente les éléments que j'ai abordé.

4.2.1 COCO (Common Objects in COntext)

Le format COCO [?] est un format d'annotation d'images très utilisé dans le domaine de la détection d'objets, qui a été proposé par Microsoft. Il est l'un des deux formats acceptés pour l'entraînement de YOLOX, avec le format VOC. Un dataset COCO est composé d'un dossier comprenant des images, et d'un fichier json qui contient les annotations. Ce dernier est structuré en quatre parties :

- des informations générales sur le dataset (date de création, version, licence, etc.) ;
- les catégories d'objets présents dans le dataset ;
- les informations sur les images (nom, taille, etc.) ;
- les annotations des objets détectés dans les images.

Les annotations peuvent contenir un paramètre booléen `iscrowd` qui indique si la détection contient plusieurs objets en même temps. Ce paramètres nous a permis d'écartier ce genre de détection, car notre tâche consiste en la détection de bateaux individuels, et non d'ensemble de

bateaux.

En plus d'être un format de dataset, COCO est aussi un dataset en lui-même, disponible en ligne : <https://cocodataset.org/#home>. Il est composé de plus de 200 000 images annotées pour différentes sortes de tâches d'apprentissage machine. Parmi toutes ces images se trouvent environ 3000 images de bateaux, qui ont été utilisées l'année dernière pour entraîner le modèle YOLOX.

Lors du téléchargement de ce dataset, nous avons remarqué que la partie test n'était pas annotée. Ceci est dû au fait qu'elle est destinée à être utilisée pour des compétitions de détection d'objets : les participants soumettent leurs prédictions sur cette partie, et les résultats sont calculés par les serveurs de COCO.

4.2.2 Data augmentation

Afin d'améliorer la généralisation du modèle, nous avons exploré les différentes méthodes de data augmentation, et nous avons été chargé de les exposer à l'équipe. Nous nous sommes basés sur les travaux de [?].

4.2.2.1 Techniques principales

Les techniques les plus classiques sont basées sur la géométrie de l'image :

- rotation ;
- translation ;
- recadrage ;
- miroir.

Il existe aussi des techniques basées sur la modification des couleurs :

- luminance ;
- saturation ;
- contraste ;
- teinte.

Enfin, il est possible d'ajouter du bruit à l'image, appliquer un flou gaussien ou encore un flou de bouger.

4.2.2.2 Data augmentation dans YOLOX

La documentation de YOLOX (<https://yolox.readthedocs.io/en/latest/>) ne liste pas de manière exhaustive les techniques de data augmentation intégrées, mais nous avons pu en identifier quelques unes en étudiant le code source.

Mixup est une technique proposée par Zhang et al. [?] qui consiste à combiner deux images en réduisant leur opacité. La mosaïque est une technique qui consiste à accoller plusieurs images pour en former une seule.

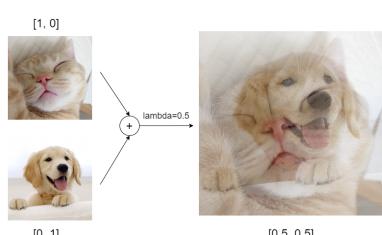


FIGURE 4.1 – Exemple de mixup



FIGURE 4.2 – Exemple de mosaïque

Bien que cette technique ne soit pas explicitement mentionnée dans la documentation, nous avons pu identifier des transformations de couleurs dans le code source, ainsi qu'un paramètre associé dans le fichier de configuration. C'est une technique de modification des couleurs.

Dans le code source, il est possible de spécifier un nombre d'époques sans augmentation. Nous n'avons pas trouvé de documentation sur cette technique.

Après discussion avec l'équipe, il a été décidé que les techniques déjà intégrées à YOLOX couvraient la plupart des augmentations possibles, et qu'il n'était pas nécessaire d'investir du temps de développement pour en ajouter d'autres. De plus, cela était également déconseillé par des développeurs ayant déjà travaillé avec YOLOX.

4.2.3 Métriques d'évaluation d'un modèle de détection

Afin de mesurer la performance de notre modèle de façon objective, nous avons utilisé les métriques suivantes :

- mAP (mean Average Precision) : moyenne des précisions pour chaque classe ;
- mAR (mean Average Recall) : moyenne du rappel pour chaque classe ;

Ces métriques sont calculées de la façon suivante :

$$\text{mAP} = \frac{VP}{VP + FP}$$

$$\text{mAP} = \frac{VP}{VP + FN}$$

avec VP les vrais positifs, FP les faux positifs et FN les faux négatifs (*voir illustration en annexe ??*).

Les prédictions du modèle consistent en des boîtes englobantes, qui sont des rectangles délimitant l'objet détecté. Pour mesurer la précision et le rappel, il faut déterminer si la prédition est vraie ou fausse. Pour cela on utilise un seuil IoU (Intersection over Union) :

$$IoU = \frac{A_{\text{intersection}}}{A_{\text{union}}}$$

Si l'IoU est supérieur à un certain seuil, la prédition est considérée comme vraie.

Dans notre cas, la précision a une grande importance : comme mentionné précédemment, un faux positif a des conséquences très négatives sur la perception du produit par le client.

Un élément important à noter est que ces deux métriques sont dépendantes : plus la précision est élevée, plus le rappel est faible, et inversement. Lors de l'inférence du modèle, chaque détection est accompagnée d'un score de confiance, qui permet de déterminer si la détection est fiable ou non. Appliquer un seuil sur ce score permet de régler le compromis entre précision et rappel.

4.3 Datasets

L'entraînement d'un modèle de machine learning nécessite, plus particulièrement de détection, un grand nombre d'images. La recherche de datasets de bateaux annotés a duré

plusieurs semaines. Le résultat de ces recherches est une combinaison de plusieurs datasets existants :

- ABOships-PLUS
- boat_computer_vision_project
- coco_boats
- dataset_GLSD
- lajolla_v5
- marvel_single_v1
- mcsips
- mods
- official_buoy_detection
- open_images
- open_images_lighthouse
- orda
- SeaDronesSee
- SeaShips
- singapore_maritime
- SMD Plus
- vais_old
- vessel_detection_v21

L'ensemble des données récoltées représente plus de 117 000 images annotées, contenant environ 390 000 bateaux. Nous avons été vigilant lors de la collecte à n'utiliser que des datasets qui, selon leurs auteurs, étaient libres de droits (sans pour autant pouvoir le vérifier). Pour pouvoir comparer nos résultats d'entraînements, nous avons décidé de constituer un dataset de test unique. Nous avons pour cela utilisé 10% de toutes nos images, et nous avons conservé ce dataset jusqu'à la fin du stage.

4.4 Prise en main des outils

4.4.1 FiftyOne

FiftyOne est un outil open source permettant de visualiser des datasets, les filtrer, détecter les images similaires, les erreurs de détections et bien d'autres fonctionnalités. Il a été d'une grande utilité lors de nos travaux, en particulier pour explorer facilement des datasets contenant des dizaines de milliers d'images, et montrer les résultats d'inférences à des collègues non spécialistes.

Après avoir récolté des datasets, nous avons entrepris de les importer dans ce logiciel. Cette étape a nécessité le développement de plusieurs scripts de conversion, car il existe un grand nombre de format d'annotation pour la détection d'objets. FiftyOne était compatible avec COCO, c'est celui-ci que nous avons choisi pour travailler tout au long du stage.

4.4.2 YOLOX

4.4.2.1 Présentation

YOLOX est un modèle de détection d'objets en temps réel, basé sur YOLOv5. Il permet de rapidement localiser et identifier des objets dans une image.

Les réseaux de neurones de la famille YOLO (You Only Look Once) sont composés de trois parties [?] : head, neck et backbone. La partie backbone permet d'extraire les features¹ de l'image, la partie neck est chargée de retrouver les informations spatiales, et la partie head permet de prédire les boîtes englobantes, les classes et les scores de confiance.

Dans leur article, les auteurs de YOLOX [?] introduisent une nouvelle méthode pour améliorer la détection d'objets : la décorrélations entre la classification de l'objet et sa localisation. Cette méthode a permis d'améliorer significativement les performances du modèle.

Ce dernier est disponible en plusieurs versions (https://yolox.readthedocs.io/en/latest/model_zoo.html#standard-models) :

- YOLOX-nano ;
- YOLOX-tiny ;
- YOLOX-S (small) ;
- YOLOX-M (medium) ;
- YOLOX-L (large) ;
- YOLOX-X (extra large) ;
- YOLOX-Darknet53.

Les modèles les plus légers sont les plus rapides, mais ont une précision plus faible. L'année précédent, le choix du modèle s'est porté sur YOLOX-S, qui est un bon compromis entre vitesse et précision.

Les poids des modèles sont disponibles en ligne. J'ai été chargé au début du stage de les télécharger et de les tester une première fois sur des images de bateaux, afin de vérifier le bon fonctionnement de l'environnement.

4.4.2.2 Entrainement

Après avoir vérifié la validité de l'inférence², nous avons entrepris d'entraîner le modèle sur nos datasets spécialisés pour les bateaux. Cet étape a été complexe, car certaines fonctions n'était pas documentées, et la structure du dossier pour le dataset d'entraînement était implicite. Nous avons donc pris soin de documenter ces informations. Après avoir réussi à entraîné le modèle, nous avons activé l'outil Weight and Biases (<https://wandb.ai/home>) pour suivre l'évolution de l'entraînement.

Pour vérifier la cohérence des résultats, nous avons effectué trois entraînements avec 3000, 5000 et 10000 images. Nous nous attendons à une augmentation de la précision avec l'augmentation du dataset, ce qui était le cas :

-
1. Les features sont une représentation des caractéristiques de l'image sous forme de vecteur.
 2. L'inférence est le processus de prédiction de l'objet dans une image.

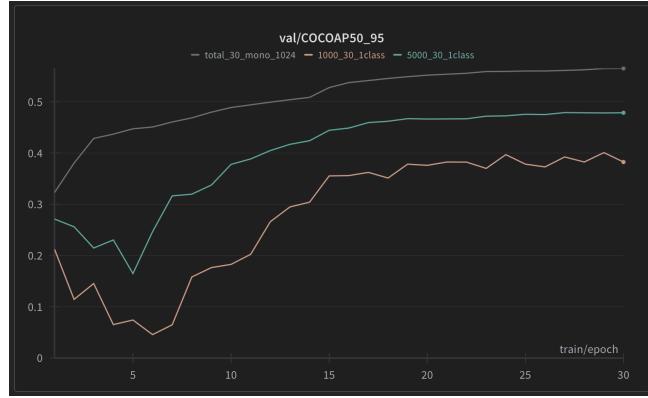


FIGURE 4.3 – Évolution de la précision en fonction de la taille du dataset

Cet expérimentation nous a également donné l'opportunité d'analyser la courbe de précision sur le dataset de validation : on peut distinguer une baisse de mAP pendant les cinq premières époques appelée "warmup", et une augmentation de la précision 15 époques avant la fin, qui correspond à l'arrêt de la data augmentation.

4.4.2.3 Évaluation

Pour obtenir des scores représentatifs de la performance du modèle, nous avons évalué YOLOX sur le dataset de test. Il est nécessaire d'ajouter le paramètre `-test` à la commande d'évaluation, sinon le modèle est évalué sur le dataset de validation (ce qui n'était pas mentionné dans la documentation). Voici le résultat d'une évaluation du modèle :

```
Average forward time: 27.18 ms, Average NMS time: 2.22 ms, Average inference time: 27.18 ms
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.699
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.945
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.766
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.396
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.623
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.832
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.331
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.713
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.736
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.501
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.680
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.858
per class AP:
| class | AP |
|:-----|:----|
| boat | 69.932 |
per class AR:
| class | AR |
|:-----|:----|
| boat | 73.551 |
```

On retrouve les métriques mAP et mAR, avec différents seuils d'IoU, et pour différentes tailles d'objets :

- small : $\text{area} < 322$ pixels ;
- medium : $322 < \text{area} < 962$ pixels ;
- large : $\text{area} > 962$.

Pour vérifier notre méthode, il m'a été demandé d'entraîner le modèle sur le même dataset que celui utilisé dans l'article, pour comparer les résultats. Nous avons évalué les deux modèles sur la partie du dataset COCO validation 2017 (121 images, 1201 détections) qui est constituée uniquement des images contenant des bateaux. Bien que les deux modèles n'aient pas été entraînés de la même façon (300 époques pour le modèle téléchargé, 30 pour celui que nous avons entraîné), nous avons obtenu des résultats très proches :

TABLE 4.1 – YOLOX entraîné par notre équipe comparé à YOLOX entraîné par les auteurs.

	YOLOX-S entraîné	YOLOX-S téléchargé
mAP	24.326	25.361
mAR	45.118	43.585

On peut donc en conclure que notre entraînement est efficace.

Nous avons par la suite testé sur de petits datasets, puis documenté, l'effet de différents hyper-paramètres :

- `input_size` : définit la taille du tenseur (correspondant à l'image) accepté en entrée du modèle ;
- `n_batch` : définit le nombre d'images utilisé à chaque itération au sein d'une époque.
- `random_size` : paramètre de data augmentation ;
- `multiscale_range` : paramètre de data augmentation.

Ces essais ont été très utiles car ils nous ont permis de déterminer la taille de batch optimale [?] pour chaque machine avec différentes cartes graphiques : 7 pour la RTX 4070 Ti Super et 3 pour la RTX 4060. Les tailles de batch sont proportionnelles à la taille de la mémoire graphique disponible sur chaque carte. Une taille de batch trop petite demanderait de diminuer le "learning rate"³, et une taille de batch trop grande saturerai la mémoire vidéo, ce qui impliquerait d'utiliser la mémoire classique de l'ordinateur. Cette dernière situation n'est pas souhaitable car elle multiplie par plusieurs dizaines le temps d'entraînement (la mémoire était bien plus lente que la mémoire vidéo).

Enfin, à la demande de notre maître de stage, nous avons comparé deux entraînements pour évaluer l'effet de l'index du label sur les performances. En effet, les modèles utilisés sont pré-entraînés avec 80 classes, et la huitième correspondant au label "boat". Le premier entraînement a été effectué avec le label boat et son index d'origine, le second a été fait avec le label boat à l'index 0. Pendant les cinq premières époques, le premier avait des performances plus élevées (évaluation sur le dataset de validation), mais aucune différence ne subsistait à la fin de l'entraînement. On peut supposer que les poids d'origine ont une influence au début.

4.4.2.4 Inférence

Pour terminer la vérification de notre environnement, nous avons écrit des scripts permettant de transférer les résultats des inférences de nos modèles entraînés dans FiftyOne, afin de mieux visualiser nos résultats. Voici une comparaison entre un des premiers modèles que nous avons entraîné, avec 30 000 images de navires, et d'autres modèles classiques :

3. Le "learning rate" détermine la vitesse à laquelle le réseau peut adapter ses poids pour apprendre des exemples qui lui sont fournis.

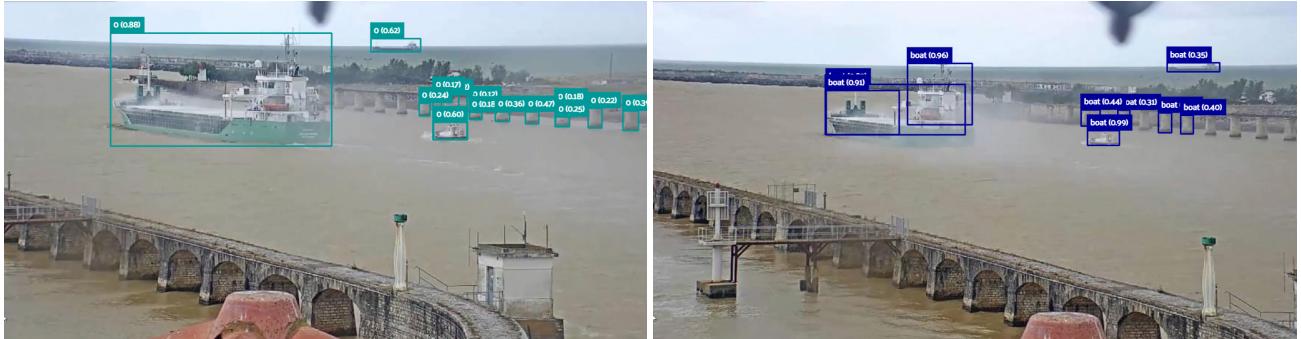


FIGURE 4.4 – Comparaison entre notre premier modèle entraîné (*à gauche*) et ResNet50 (*à droite*).

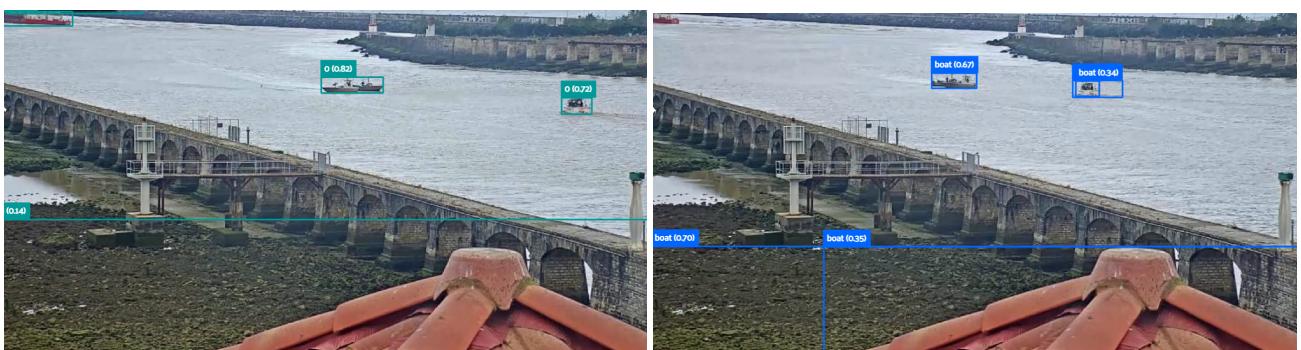


FIGURE 4.5 – Comparaison entre notre premier modèle entraîné (*à gauche*) et YOLOv8 (*à droite*).

Après avoir analysé les images nous avons conclu que tous les réseaux, mis à part leurs scores mAP et mAR respectifs, avaient des problèmes similaires : détection de piliers et du toit comme des bateaux. Ceci valide encore une fois la cohérence de nos travaux par rapport à l'état de l'art.

4.5 Développement du pipeline d'entraînement

Cette section présente le fonctionnement du système que nous avons développé, de l'import des données jusqu'à l'évaluation du modèle, en passant par l'entraînement.

Pour guider nos développements, nous nous sommes renseigné sur les principes MLOps⁴. Cette approche nous a conduit à créer un parcours stable qui permet de lancer le preprocessing⁵ puis l'entraînement en quelques minutes, en centralisant tous les paramètres.

En résumé, le pipeline fonctionne de la manière suivante :

1. les datasets sources (provenant de différents fournisseurs de données) sont stockés au format COCO ;

4. MLOps est l'hybridation de "Machine Learning" et "DevOps". Il s'agit d'une approche qui combine les processus de développement logiciel traditionnels avec ceux du machine learning pour améliorer la qualité, la productivité, et la sécurité des modèles de machine learning.

5. Le préprocessing, également appelé "prétraitement" ou "préparation des données", est un processus important dans l'analyse de données et l'apprentissage automatique. Il consiste à mettre en forme et à nettoyer les données pour qu'elles soient utilisables par les algorithmes de machine learning.

2. les datasets sources sont importés dans FiftyOne, où différents traitements (comme l'annotation ou le filtre de similarité) peuvent leur être appliqués ;
3. les datasets sont en suite réunis en un seul ensemble d'images appelé "input dataset" ;
4. "input dataset" est exporté dans un dossier pour pouvoir être utilisé par YOLOX.

Comme énoncé en section 2.4, le pipeline doit être facile d'utilisation. Ce sont donc deux Jupyter Notebooks qui permettent de lancer tous les scripts, le premier correspondant au prétraitement, le second à l'entraînement. Le prétraitement contient des opérations tels que la suppressions d'images similaires, l'agglomération de tous les datasets ou encore du clustering. Le script d'entraînement permet de lancer l'entraînement du modèle, mais aussi l'activation du logger⁶.

L'entièreté du pipeline est contrôlé par un fichier de configuration (*voir annexe ??*) au format **json**.

Il contient les paramètres suivants pour le prétraitement : datasets à utiliser, nombre d'images pour l'entraînement, tuilage des images, labels à conserver, différents filtres pour les images (seuil de similarité, contraste, bruit...). Il permet aussi d'activer le calcul des embeddings⁷ et régler le nombre de clusters⁸ pour la visualisation.

Pour l'entraînement, on trouve entre autres les paramètres pour le nombre d'époques, la taille de batch, la profondeur du modèle YOLOX.

Concernant le monitoring de l'entraînement, nous avons d'abord utilisé Weight&Biases (<https://wandb.ai/home>), puis Tensorboard (<https://www.tensorflow.org/tensorboard>).

Afin de conserver une trace de chaque entraînement, le pipeline permet de transferer automatiquement les informations suivantes vers une destination pérenne : nombre d'images et de détections utilisées ainsi que leurs sources (différents datasets), dimensions des images, classes, durée d'entraînement, carte graphique utilisées, filtres et seuils.

4.6 Prétraitement

Nous présentons dans cette section l'ensemble du travail réalisé concernant le prétraitement des images et la préparation pour les entraînements.

4.6.1 Analyse des datasets

Après avoir réunis un grand nombre d'images, nous les avons explorées pour avoir une idée de ce qu'elles contenait.

4.6.1.1 Labels

Pour commencer, chaque source d'image contenant des classes différentes. Les bateaux (notre classe d'intérêt) était parfois appelée "vessel", "ship", ou encore des noms plus précis comme "sailboat" ou "warship".

6. Le logger est le système permettant de monitorer les performances pendant l'entraînement.

7. Les embeddings sont une représentation numériques des images.

8. Un cluster est un groupe d'objets similaires ou pertinents qui ont été identifiés et regroupés en fonction de caractéristiques communes.

Pour pouvoir rassembler ces images et les utiliser, nous avons dans un premier temps créé des dictionnaires de conversion (*voir exemple en annexe B*) de classes pour obtenir un seul label "boat".

4.6.1.2 Similarité

Lors de la visualisation des datasets, nous avons remarqué que plusieurs d'entre eux utilisaient les même images. Aussi, certains étaient issus de vidéo de surveillance, ce qui produit des images très similaires entre elles. Le risque d'avoir des images très ressemblante est le surapprentissage⁹; pour éviter ce problème, nous avons utilisé un modèle de vision (ResNet101) pour calculer les embeddings de nos images, et créer une matrice de similarité. Ceci nous a permis d'appliquer un seuil pour régler l'hétérogénéité de nos datasets.

4.6.1.3 Exploration

Fiftyone permet d'utiliser les embeddings pour afficher une carte (*voir figure 4.6*) dans laquelle chaque point représente une image, et la proximité des points traduit la similarité des images. Ceci permet de sélectionner facilement des sous-ensemble d'un dataset, et rend l'exploration beaucoup plus facile.

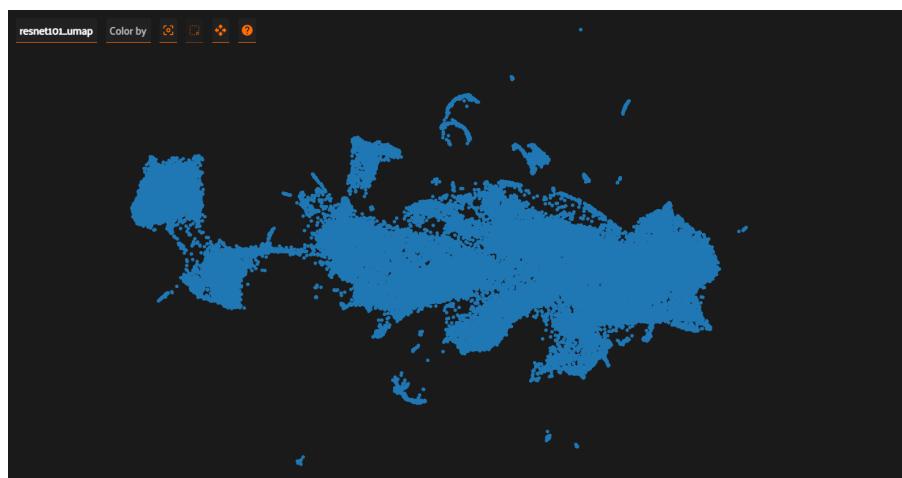


FIGURE 4.6 – Exemple de carte des embeddings dans FiftyOne

9. Le surapprentissage (ou overfitting en anglais) est un phénomène qui se produit lorsqu'un modèle de machine learning devient trop spécifique à l'ensemble de données d'apprentissage utilisé pour sa formation, et qu'il n'est plus capable de généraliser ses connaissances pour prédire correctement les résultats sur des données inconnues.

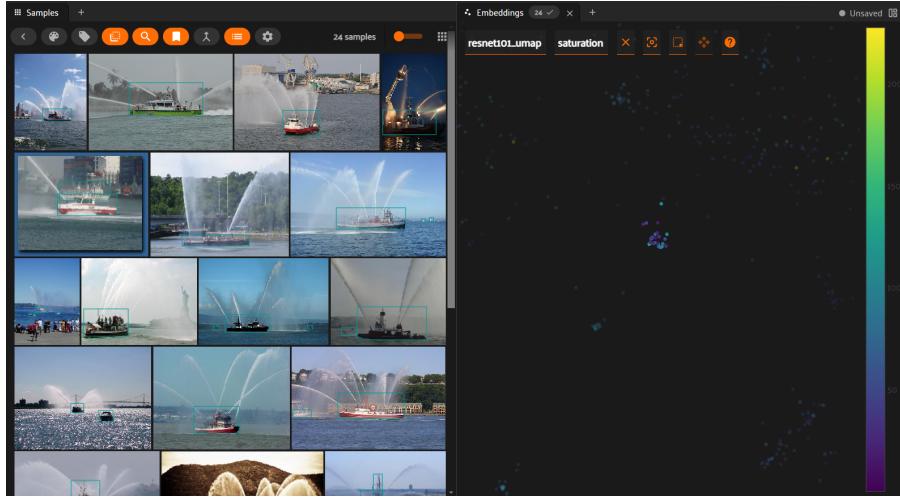


FIGURE 4.7 – Exemple de sélection d'un ensemble de points

Cet partie du travail nous a mené à identifier des éléments indésirables tels que des sous-marins ou des bouées labelisées comme étant des bateaux par exemple. Nous avons aussi remarqué que certains bateaux, selon le modèle de vision utilisé, se ressemblaient moins que d'autres : les cargos et les ferrys étaient plus proches sur la carte que les voiliers. Cela nous a apporté une meilleure intuition.

Enfin, nous avons observé des erreurs d'annotation : boîte englobante décalée ou trop petite, jouets et photos considérés comme des bateaux... Ces erreurs étant très ponctuelles et difficilement détectables, nous avons décidé de les ignorer.

4.6.1.4 Annotation

Après avoir travaillé sur une seule classe de bateaux, nous nous sommes servi des outils mentionnés précédemment pour créer nos propres annotations (*voir interface de FiftyOne en annexe E*). Cette action était guidée par plusieurs reflexions. La première est évidente : prédirer plusieurs classes de bateaux est un avantage indéniable du point de vue du client. Le seconde est que, les performances variant selon les classes (certains types de bateaux sont plus faciles à détecter que d'autres), on peut envisager d'être plus exigeant lorsque le modèle détecte une classe difficile, c'est à dire augmenter le seuil de confiance. Cette dernière méthode pourrait permettre de réduire le nombre de faux positifs, et donc la satisfaction des utilisateurs.

Comme mentionné précédemment, nous avons utilisé ResNet101 pour le calcul des embeddings, avec cette fois une subtilité : plutôt que d'utiliser le modèle sur tout l'image, nous l'avons utilisé uniquement sur les bateaux. En effet, deux images d'une même rivière peuvent être considérées comme similaires, bien que les bateaux présents sur ces images soient différents. Le calcul des embeddings sur les détections permet d'éviter ce biais.

Une fois les embeddings calculés et la carte de similarité créée, nous avons utilisé un algorithme de clustering pour accélérer le regroupement des bateaux par types. Après plusieurs essais et une analyse qualitative, nous avons choisi K-Means. Il est aussi précis que d'autres algorithmes plus complexes (DBSCAN, OPTICS, Agglomerative...), mais surtout beaucoup plus rapide (*voir documentation de scikit-learn <https://scikit-learn.org/stable/modules/clustering.html>*).



FIGURE 4.8 – Exemples de clusters

Il faut garder à l'esprit que les exemples ci-dessus sont des cas particuliers : la plupart des clusters contenaient parfois plusieurs types de bateaux, et deux semaines de travail ont été nécessaires pour obtenir un résultat satisfaisant. En tout, 117 744 objets ont été annotés, dont 29% sont restés dans la classe "**boat**" car trop difficile à classifier (en général des images floues ou trop petites).

Voici un tableau de la répartition des objets dans les différentes classes :

Les définitions des différentes classes ont été documentées pour permettre des annotations futures (*voir annexe C*). Après annotation, nous avons utilisé FiftyOne pour exporter ce dataset.

4.6.1.5 Filtres

Nous avons remarqué que la majorité des objets de nos datasets sont petits : dans le dataset COCO par exemple, environ 41% des objets sont petits ($\text{aire} < 322$), 34% sont moyens ($322 < \text{aire} < 962$), et 24% sont grands ($\text{aire} > 962$), selon la définition du site de COCO.

Nous avons donc mis en place des filtres pour contrôler la qualité des images utilisées pour l'entraînement. Pour cela, nous avons calculé différentes métriques pour mesurer l'entropie, le flou, la luminosité, le contraste, l'exposition, le bruit et la saturation.

Nous avons également mis en place un filtre qui permet d'appliquer un seuil de similarité (calculé au préalable, *voir section 4.6.1.2*).

Tous ces filtres sont appliqués au moment d'exporter les datasets dans le dossier servant à l'entraînement.

4.7 Entrainements

Durant le stage, l'amélioration de l'entraînement de YOLOX pour augmenter sa précision a nécessité plusieurs expérimentations. Au total, environ 60 entraînements ont été réalisés, d'une durée moyenne d'environ 2 jours.

Chaque entraînement a donné lieu à un rapport rédigé sur OneNote et une présentation des résultats à l'équipe. Les résultats de ces entraînements sont exposés au chapitre 5.

TABLE 4.2 – Nombre d'objets par classes

Type	Count
Cargo	35033
Boat	23648
Sailboat	13153
Buoy	9282
Recreational	8827
Ferry	8272
Warship	4593
Fishing	3374
Cruise	3172
Lighthouse	2705
Luxury	2145
Breakwater	2136
Service	1080
Platform	273
Bulker	51

4.8 Inférence

4.8.1 Conversion au format ONNX

À la fin d'un entraînement de YOLOX, on obtient un fichier au format pth (PyTorch), qui correspond à un dictionnaire contenant la structure et les poids du réseau de neurones. Ce fichier doit être converti au format onnx¹⁰ pour être utilisé pour l'inférence.

4.8.2 Quantization

En plus de la conversion en onnx, nous avons entrepris d'appliquer une "quantization"¹¹ sur nos modèles pour les rendre plus rapides lors de l'exécution.

Nous avons utilisé le framework OpenVINO, créé par Intel, car il permet l'optimisation pour

10. Le format ONNX (Open Neural Network Exchange) est un format de fichier qui permet d'échanger et de partager des modèles de réseaux neuronaux entre les différentes plateformes et langages de programmation, sans perte de précision ou de performances.

11. La quantization est une méthode qui transforme un modèle en une représentation optimisée pour un certain matériel. La méthode la plus populaire est la quantization 8-bit post entraînement, car elle a un impact contenu sur la précision du modèle et permet une accélération importante (*source :* https://docs.openvino.ai/2023.3/ptq_introduction.html)

les processeurs de la marque, qui occupent la majorité des machines visées par notre logiciel.

La quantization ayant un impact sur les performances, nous avons mesuré ces dernières pour éviter une trop grande perte de précision. Selon nos tests, cette perte est d'environ 2% maximum, ce que nous avons jugé acceptable vis à vis du gain en performances.

Nous avons observé, sur une machine équipée d'un processeur Intel i7-8400, une augmentation d'environ 41% en moyenne de la vitesse d'inférence :

TABLE 4.3 – Vitesse d'inférence (images par seconde)

Modèle	fp32	int8
yolox_s	11.12	17.65
yolox_s	10.78	18.41
yolox_s	11.33	18.97
yolox_s	11.34	18.72

4.8.3 Mise en production

Après avoir produit et optimisé des modèles, nous avons commencé, vers la fin du stage, à discuter de l'intégration aux outils existants. Ce travail a été fait par mon maître de stage, qui a intégré les réseaux à l'outil DebuggerTool (*voir annexe E*), créé l'année dernière.

Cet outil a fait l'objet de plusieurs optimisations. La principale est que l'inférence est effectuée par tuiles, ce qui permet de mieux détecter les petits bateaux. Aussi, il contient un tracker, ByteTrack [?], qui permet de suivre les détections.

Les meilleurs modèles seront prochainement sélectionnés, et potentiellement intégrés à TimeZero Coastal Monitoring.

Chapitre 5

Résultats

Ce chapitre présente les résultats des différentes expérimentation réalisées, généralement sous forme de couple d'entraînements. Sauf indications contraires, les précision et recall mentionnés sont issus de l'évaluation du modèle sur le dataset de test constitué en début de stage (5918 images).

5.1 Tests

5.2 Tuilage

D'après nos recherches, le tuilage est un moyen efficace d'augmenter les performances d'un modèle après entraînement. Nous avons tenté d'appliquer le tuilage avant l'entraînement, pour réduire le nombre de petits objets (quelques pixels) dans le dataset. Nous avons pour cela utilisé SAHI [?].

Nous avons fait trois expérimentations comprenant deux entraînements chacunes. Les datasets d'entraînement contenaient uniquement 1000 images, tirées au hasard parmi tous les datasets, afin de connaître rapidement le résultat.

Pour chaque expérimentation, un des entraînements était fait sur des tuiles de 640 pixels de côté, provenant des images originales.

Voici les précisions et rappels mesurés sur le dataset de test :

	Images tuilées	Images originales
mAP	34.001	35.276
mAR	50.249	50.654

TABLE 5.1 – Tuilage pré-entraînement, test avec petits datasets (moyenne des résultats).

Le tuilage pré-entraînement a un impact sensiblement négatif sur les performances du modèle.

À la demande de notre maître de stage, nous avons réitéré cette expérimentation sur un dataset plus conséquent, composé de toutes les images à notre disposition, filtrées avec les seuils de similarité que nous avons déterminé. Ceci représente 30 000 images.

La conclusion est similaire aux entraînements de test effectués précédemment : le tuilage avant entraînement réduit de manière significative la précision et le rappel du modèle.

	Images tuilées	Images originales
mAP	55.003	60.536
mAR	63.896	67.477

TABLE 5.2 – Tuilage pré-entraînement.

5.3 Entraînement avec images vides

Ces entraînements ont pour but de déterminer si la présence d'images sans bateaux pendant l'entraînement permet de réduire le nombre de faux positifs durant l'inférence.

Pour créer un dataset contenant des images vides, nous avons utilisé le tuilage, car SAHI permet de facilement conserver ou non les tuiles issue de parties d'image sans détections.

Voici les résultats obtenus avec 3000 images (avant tuilage) :

	Tuilage simple	Tuilage avec suppression des tuiles vides
mAP	43.875	43.765
mAR	56.028	55.201

TABLE 5.3 – Tuilage avec ou sans conservation des images vides.

Nous tirons comme conclusion de cet entraînement qu'ajouter des images vides n'a pas d'impact sur les performances du modèle.

5.4 Scinder la classe boat

Ces entraînements ont pour but de savoir si diviser les bateaux en plusieurs catégories améliore la précision générale du modèle. Les entraînements ont été fait sur le datasets **ABOships-PLUS**, car il contient les classes "powerboat", "sailboat" et "ship" et un nombre d'images suffisant (5866 images).

Le dataset de test constitué au début du stage ne permet pas l'évaluation du modèle car les différentes classes utilisées pendant l'entraînement ne sont pas présentes. Nous avons donc retenu les résultats issus de l'évaluation sur le dataset de validation à la dernière époque, puis nous avons calculé une moyenne pondérée des précision et rappels pour les classes powerboat, sailboat et ship.

	Classe unique	Classes multiples (moyenne)
mAP	41.828	40.856
mAR	53.520	53.475

TABLE 5.4 – Entraînement avec plusieurs classes (1 dataset).

On observe une légère diminution de la précision et du rappel lorsqu'on tente de prédire plusieurs classes.

Pour aller plus loin, nous avons créé une seconde expérimentation, cette fois-ci à partir de trois datasets différents : `dataset_GLSD`, `SMD_plus`, `vessel_detection_v21`. Nous avons choisi ces derniers car ils comportaient des classes similaires que nous avons pu réunir : "cargo", "ferry", "sailing boat", "fishing boat".

	cargo	ferry	fishing boat	sailing boat	moyenne
mAP	53.895	88.126	67.593	73.628	71.923
mAR	59.922	90.230	71.718	76.653	75.692

TABLE 5.5 – Entraînement avec plusieurs classes (3 datasets).

	boat
mAP	75.859
mAR	78.416

TABLE 5.6 – Entraînement avec une classe (3 datasets).

On observe encore une fois une diminution de la précision au profit de la capacité à classifier les bateaux.

Nous concluons des résultats précédents que diviser les bateaux en plusieurs classes plus précises ne permet pas d'améliorer les scores globaux lors de l'évaluation du modèle. Cependant, l'intérêt du point de vue commercial est indéniable. Nous avons donc entrepris de faire des entraînements à partir de datasets annotés par nos soins, selon une liste de classes la plus précise possible (*voir annexe C*).

Les scores du modèles en fonction des classes sont les suivants :

	boat	breakwater	bulker	buoy	cargo	cruise	ferry	fishing
mAP	39.839	44.762	79.066	42.374	79.203	90.311	81.185	72.556
mAR	54.493	58.744	85.000	49.788	82.153	92.163	85.623	77.036
	lighthouse	luxury	platform	recreational	sailboat	service	warship	moyenne
mAP	50.190	80.628	33.638	60.410	60.314	53.784	85.930	66.532
mAR	57.046	84.431	68.276	67.511	67.125	64.032	88.831	73.235

TABLE 5.7 – Entraînement avec plusieurs classes (3 datasets).

On remarque que certains classes sont plus difficiles à prédire que d'autres. Puisque nous connaissons la précision du modèle pour chaque classe, nous avons suggéré d'adapter les seuils

de confiance (appliqués lors de l'inférence) en fonction de la précision de chaque classe. Par exemple, la classe "platform" étant la plus difficile à prédire, on appliquerait un seuil plus élevé afin que seules les détections avec une confiance très élevée soient affichées.

Cependant, en conduisant des analyses plus poussées, ceci ne semble pas être une bonne solution. En effet, bien que le score de corrélation (Pearson) entre la taille des détections et la précision soit de 0.60025 (peu significatif), il faut faire attention à ne pas confondre les classes difficile à prédire et les classes ne contenant que des petits objets.

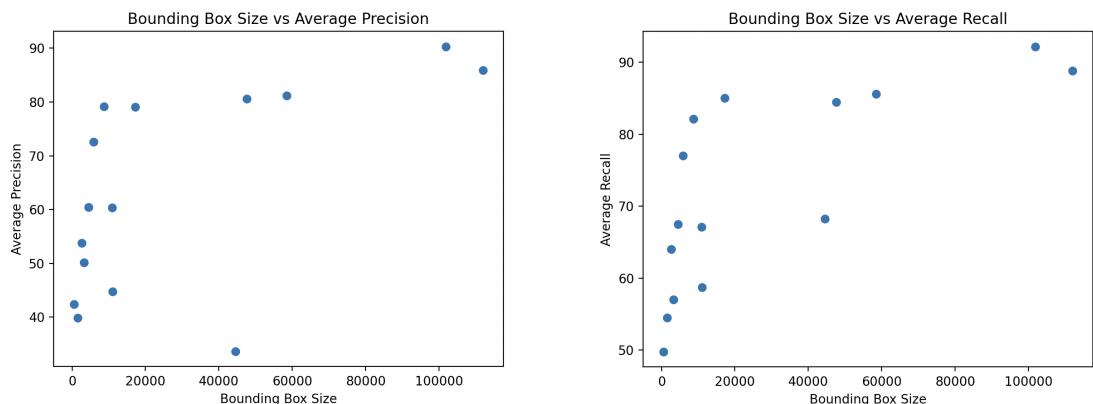


FIGURE 5.1 – Précision et rappel en fonction de la taille des objets.

5.5 Image facile à identifier

Nos expériences nous ont conduit à nous demander si des images plus faciles à identifier, c'est à dire en haute définition, contrastées et lumineuses, permettait au réseau de neurones de mieux apprendre les traits caractéristiques des bateaux, et donc d'obtenir de meilleurs scores.

Pour répondre à cette question, nous avons utilisé le dataset `marvel` dont les bateaux occupaient toutes l'image et étaient en général situés au centre. Le nombre d'image était de 11502 pour cet entraînement.

	marvel	images tirées au hasard
mAP	19.252	47.858
mAR	27.436	57.285

TABLE 5.8 – Entraînement avec image facile à identifier.

Ces scores montrent qu'il vaut mieux entraîner YOLOX avec des images représentatives de l'environnement d'inférence.

5.6 Résultats précédents

Le modèle entraîné l'année dernière a été évalué sur notre dataset de test :

	YOLOX-S base	Meilleur modèle 2023	Meilleur modèle 2024
mAP	25.361	32.903	60.907
mAR	43.585	47.276	66.471

TABLE 5.9 – Comparaison avec notre meilleur modèle.

Chapitre 6

Conclusions

6.1 Solutions retenues

6.2 Développement durable

Mes travaux sur l'algorithme de détection de navires nécessitent l'utilisation de cartes graphiques, qui consomme beaucoup d'énergie pour fonctionner. Dans le cadre de ce projet, j'ai effectué environ 60 entraînements du modèle, ce qui a consommé une quantité importante d'énergie électrique. Selon les calculs effectués (*source : <http://calculator.green-algorithms.org>*), cela représente un impact environnemental notable, avec un total de 762,83 kg CO₂e émis au cours de ces entraînements. Cette constatation soulève des questions sur l'impact global de mes travaux et le besoin de trouver des solutions pour réduire l'empreinte écologique de ce type d'applications. Cependant, il est également important de noter que cette technologie a le potentiel d'améliorer la sécurité et la surveillance dans les ports et les zones côtières, réduisant ainsi le risque d'accidents et de dommages causés par les navires. Il s'agit donc d'un exemple de compromis entre des enjeux concurrents qui nécessite une attention particulière pour minimiser l'impact négatif et maximiser les avantages positifs.

Chapitre 7

Perspectives

test [?]

Table des figures

1.1	Exemple de vue radar, AIS et camera dans TimeZero Coastal Monitoring	2
4.1	Exemple de mixup	9
4.2	Exemple de mosaïque	9
4.3	Évolution de la précision en fonction de la taille du dataset	13
4.4	Comparaison entre notre premier modèle entraîné (<i>à gauche</i>) et ResNet50 (<i>à droite</i>).	15
4.5	Comparaison entre notre premier modèle entraîné (<i>à gauche</i>) et YOLOv8 (<i>à droite</i>).	15
4.6	Exemple de carte des embeddings dans FiftyOne	17
4.7	Exemple de sélection d'un ensemble de points	18
4.8	Exemples de clusters	19
5.1	Précision et rappel en fonction de la taille des objets.	26

Liste des tableaux

4.1	YOLOX entraîné par notre équipe comparé à YOLOX entraîné par les auteurs	14
4.2	Nombre d'objets par classes	20
4.3	Vitesse d'inférence (images par seconde)	21
5.1	Tuilage pré-entraînement, test avec petits datasets (moyenne des résultats).	23
5.2	Tuilage pré-entraînement.	24
5.3	Tuilage avec ou sans conservation des images vides.	24
5.4	Entraînement avec plusieurs classes (1 dataset).	24
5.5	Entraînement avec plusieurs classes (3 datasets).	25
5.6	Entraînement avec une classe (3 datasets).	25
5.7	Entraînement avec plusieurs classes (3 datasets).	25
5.8	Entraînement avec image facile à identifier.	26
5.9	Comparaison avec notre meilleur modèle.	27

Appendices

Annexe A

Métriques

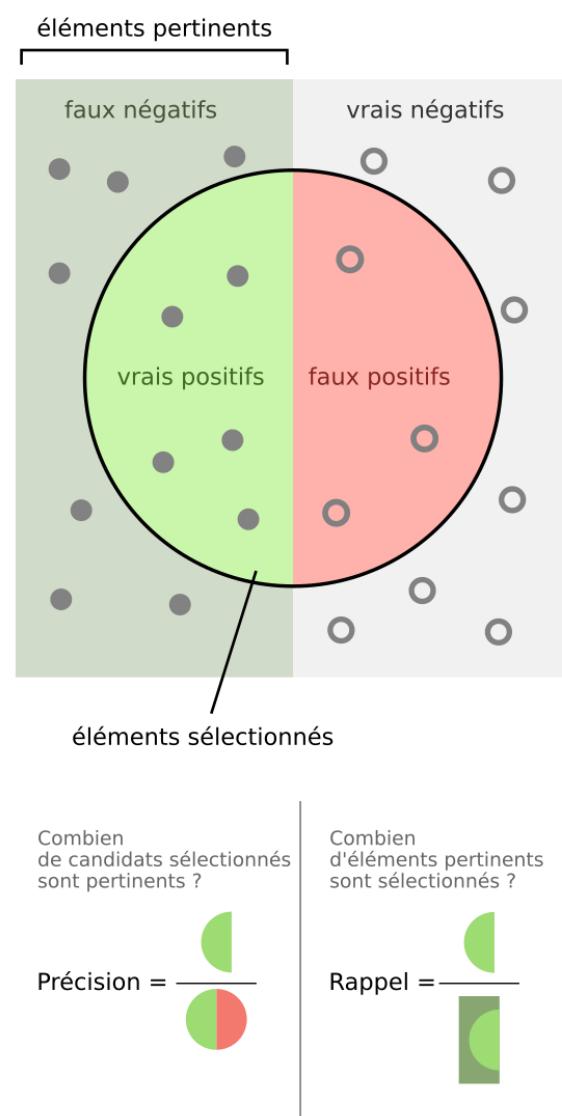


FIGURE A.1 – Précision et rappel (*source : https://fr.wikipedia.org/wiki/Précision_et_rappel*)

Annexe B

Exemple de dictionnaire de conversion de labels (format json)

```
{  
    "vessel": "boat",  
    "unknown": "boat",  
    "commercial cargo-distant": "boat",  
    "tug": "boat",  
    "non-commercial small-human powered-blurry": "boat",  
    "non-commercial medium-distant": "boat",  
    "commercial cargo-blurry": "boat",  
    "non-commercial sailing": "boat",  
    "non-commercial small-backlit": "boat",  
    "non-commercial small-dinghy": "boat",  
    "non-commercial large-backlit": "boat",  
    "commercial cargo": "boat",  
    "other-backlit": "boat",  
    "other-blurry": "boat",  
    "non-commercial large": "boat",  
    "non-commercial small-distant": "boat",  
    "non-commercial sailing-blurry": "boat",  
    "non-commercial small-dinghy-distant": "boat",  
    [...]  
    "non-commercial small-dinghy-blurry": "boat",  
    "tug-blurry": "boat",  
    "commercial small-blurry": "boat",  
    "non-commercial small-blurry": "boat",  
    "non-commercial large-distant": "boat",  
    "commercial small": "boat",  
    "commercial small-backlit": "boat",  
    "other-tow-backlit": "boat",  
    "non-commercial medium-backlit": "boat",  
    "commercial large passenger-blurry": "boat",  
    "non-commercial sailing-distant": "boat",  
    "other-tow": "boat",  
    "tug-backlit": "boat",  
    "non-commercial medium-blurry": "boat",  
    "tug-distant": "boat",  
}
```

```
"non-commercial large-blurry": "boat",
"non-commercial sailing-backlit": "boat",
"other": "boat",
"non-commercial small-human powered-distant": "boat",
"commercial large fishing-blurry": "boat",
"unknown-distant": "boat"
}
```

Annexe C

Définition des classes utilisées pour annoter les datasets

- **ferry** : Un navire utilisé pour le transport de personnes ou de marchandises.
- **warship** : Des navires militaires, des patrouilleurs ou d'autres navires utilisés par les forces militaires.
- **submarine** : Sous marin.
- **recreational** : Des petits bateaux légers fabriqués en matériau souple, des kayaks et des bateaux à pédale, ou des embarcations personnelles conçues pour le loisir ou la course.
- **luxury** : Des grands yachts luxueux conçus pour le plaisir.
- **sailboat** : Des petits voiliers utilisés pour le loisir ou la course.
- **service** : Des navires utilisés pour remorquer ou manœuvrer d'autres navires, les bateaux-pilotes ou les embarcations de sauvetage et de garde-côtes.
- **platform** : Des structures utilisées pour l'extraction pétrolière ou gazière.
- **fishing** : Un type de bateau utilisé pour la pêche ou le loisir.
- **bulker** : Des bateaux utilisés pour le transport de marchandise en vrac (vraquiers).
- **cargo** : Des grands navires conçus pour transporter des marchandises dans des conteneurs standardisés.
- **cruise** : bateaux de croisière.
- **buoy** : bouées.
- **breakwater** : des jetées ou structure destinées à réduire l'impact des vagues.
- **lighthouse** : phares et structure de localisation.

Annexe D

Seuils de similarité choisis

Les seuils correpondent à la similarité maximum acceptée entre deux images.

```
"similarity_threshold": {  
    "coco_boats": 1,  
    "coco_boats_val": 1,  
    "ABOships-PLUS-test": 1,  
    "ABOships-PLUS-train": 1,  
    "ABOships-PLUS-val": 1,  
    "bayonne": 0.95,  
    "dataset_GLSD_train": 0.98,  
    "dataset_GLSD_test": 0.98,  
    "lajolla_v5_test": 1,  
    "lajolla_v5_train": 1,  
    "lajolla_v5_valid": 1,  
    "marine_object_detection_v4_train": 0.98,  
    "marine_object_detection_v4_valid": 0.98,  
    "marvel_single_v1_train": 1,  
    "marvel_single_v1_test": 1,  
    "mcships_test": 1,  
    "mcships_train": 1,  
    "mcships_valid": 1,  
    "mods_train": 0.96,  
    "mods_test": 0.96,  
    "SeaDronesSee_train": 0.99,  
    "SeaDronesSee_test": 0.99,  
    "SeaShips_train": 0.985,  
    "SeaShips_test": 0.985,  
    "singapore_maritime_test": 0.995,  
    "singapore_maritime_train": 0.995,  
    "singapore_maritime_valid": 0.995,  
    "vais_old_train": 0.89,  
    "vais_old_test": 0.89,  
    "vessel_detection_v21_test": 1,  
    "vessel_detection_v21_train": 1,  
    "vessel_detection_v21_valid": 1  
}
```

Annexe E

Interfaces

Interface de FiftyOne pour l'annotation de datasets :

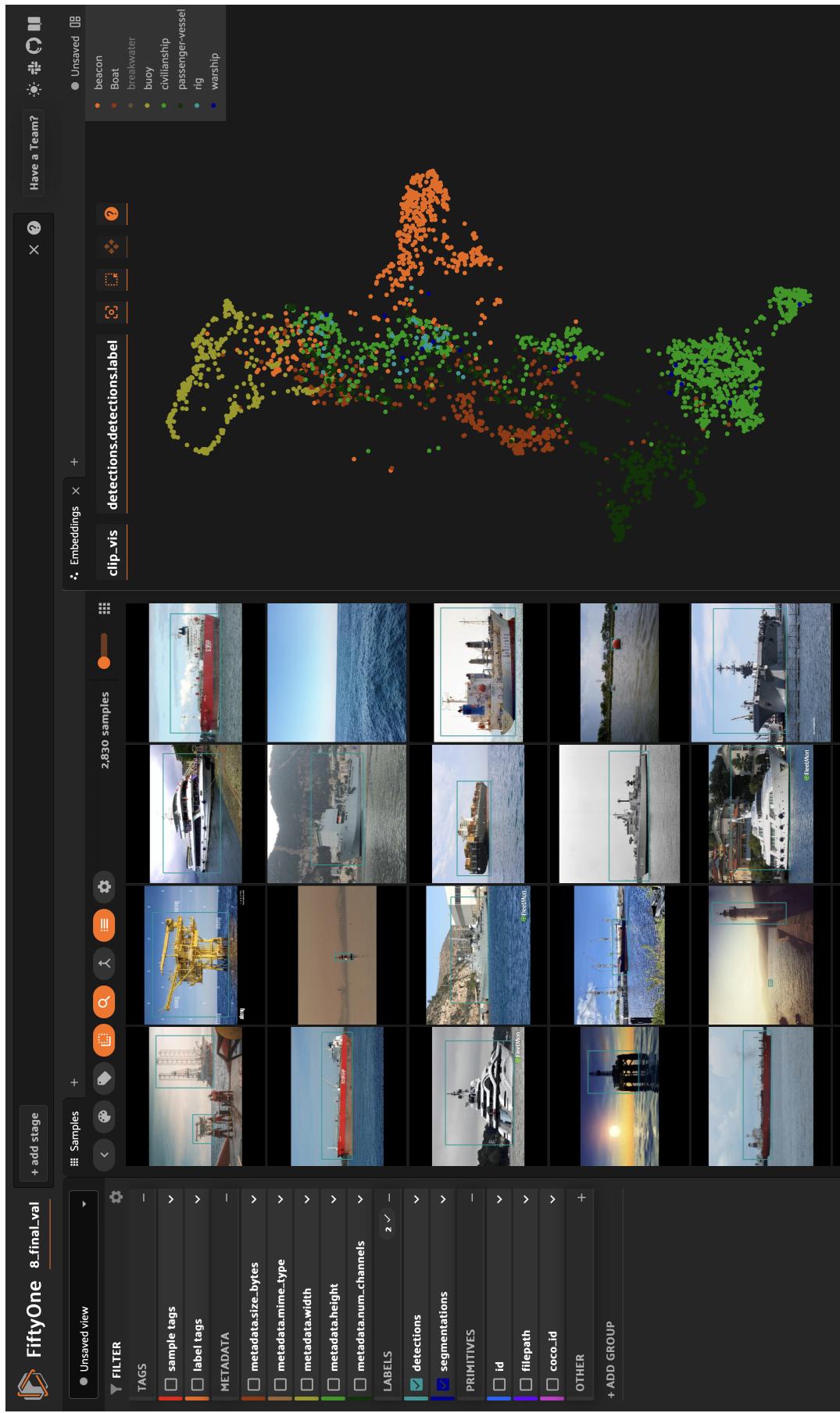


FIGURE E.1 – Interface pour l'annotation de datasets (carte de similarité à droite, une couleur par cluster K-Means)

Debugger Tool :

