

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, dont 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 météorologiques. 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, l'entreprise avons décidé de construire leur propre solution. 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. Ce modèle, YOLOX, fait partie des modèles open source les plus performants à ce jour. Les solutions apportées par le stage sont une base de données de plus de 120 000 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	Début du projet	2
2	Cahier des charges	3
2.1	Précision de la détection	3
2.2	Vitesse de traitement	3
2.3	Visualisation et supervision	3
2.4	Facilité d'utilisation	3
3	Méthodes et outils	5
3.1	Matériel	5
3.2	Planning du stage	5
3.3	Organisation d'équipe	8
4	Travail réalisé	9
4.1	Environnement virtuel	9
4.2	Documentation initiale	9
4.3	Datasets	12
4.4	Prise en main	12
4.5	Pipeline d'entraînement	15
4.6	Prétraitement	17
4.7	Entraînements	20
4.8	Inférence	20
5	Résultats	23
5.1	Tests	23
5.2	Tuillage	24
5.3	Images vides	25
5.4	Scinder la classe boat	25
5.5	Images faciles à identifier	27
5.6	Résultats précédents	28
6	Conclusions	29
6.1	Solutions retenues	29
6.2	Bénéfices du stage	29
6.3	Développement durable	29
7	Perspectives	31
Appendices		36

Chapitre 1

Introduction et contexte

1.1 Présentation de l'entreprise

Il y a 35 ans, Brice Pryszo a fondé MaxSea International en créant 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 compte des plaisanciers que des pêcheurs ou encore 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 de leur service météo, qui donne accès à des précisions météorologiques très complètes, fournies par les modèles météo les plus fiables.

Ces technologies 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 en particulier à TimeZero Coastal Monitoring, qui permet de surveiller les zones côtières.

C'est pour ce dernier produit que nous avons effectué notre stage de fin d'études, au sein de l'équipe de recherche et développement. Plus précisemment, nous avons implémenté un modèle de détection de navires en temps réel, 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.

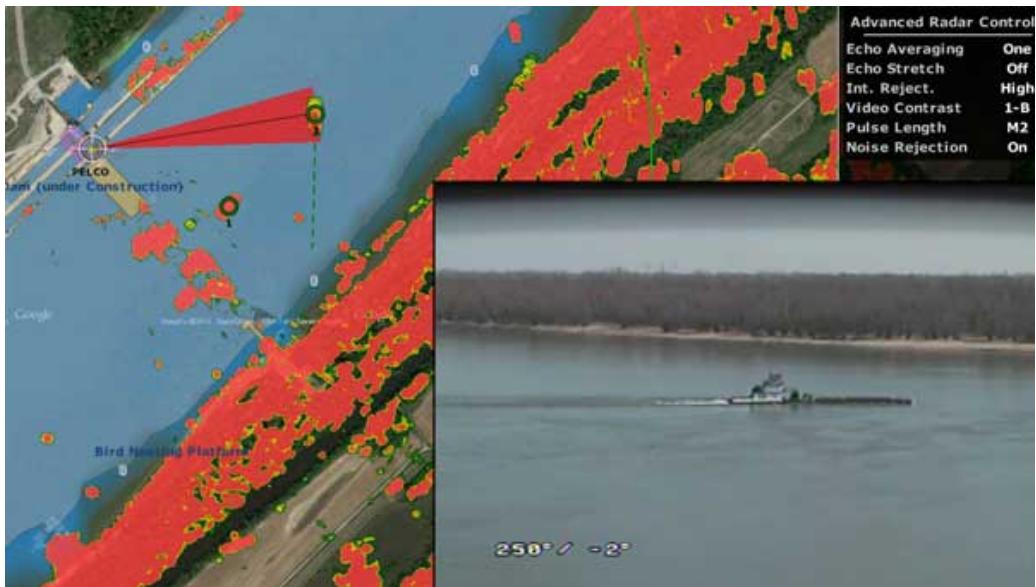


FIGURE 1.1 – Exemple de vue radar, AIS et camera dans TimeZero Coastal Monitoring

L'équipe que nous avons rejoint était composée de 2 personnes : Ronan Golhen, mon tuteur de stage et directeur technique, et Victor Opter, développeur logiciel. L'entreprise ne comptant parmi ses effectifs aucun développeur en apprentissage automatique, nous avons 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 de surcroit une grande expérience en navigation.

1.2 Début du projet

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

Ses travaux comptaient tout d'abord le choix d'un modèle de machine learning nommé YO-LOX (*voir* 4.4.2). Ce choix a été guidé par la recherche de performances, aussi bien du côté de la précision que de celui de la rapidité, et 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 récolté des datasets, et réuni environ 8000 images de navires. Trois entraînements ont été réalisés en utilisant des machines distantes via Google Collab.

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

Après avoir pris connaissance de ces travaux, nous avons donc entrepris de développer un nouvel outil, en utilisant les même librairies que l'année précédente, et en écrivant des scripts plus modulaires et en documentant précisément les étapes de l'entraînement et le code.

Chapitre 2

Cahier des charges

L'objectif principal du stage était d'entraîner un modèle de détection de navires pour enrichir la fonction de vidéo-surveillance de Coastal Monitoring. Cet objectif peut être décomposé en plusieurs éléments, détaillés ci-dessous.

2.1 Précision de la détection

La précision est le point clef 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 entravent 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'étant pas nécessairement de dernière génération.

2.3 Visualisation et supervision

Afin de partager nos résultats à l'équipe, d'explorer les datasets et d'observer les résultats d'inférence, nous avons choisi d'utiliser FiftyOne, un outil open source.

Pour le suivi des performances pendant l'entraînement, nous avons utilisé Weight&Biases. Cet outil présente l'avantage d'être en ligne, ce qui permet de suivre les entraînements à distance et facilement partager les résultats. Après discussion, nous avons décidé de changer au profit de TensorBoard, qui permet une meilleure protection des données.

2.4 Facilité d'utilisation

Il est important que le système soit facile à utiliser et à maintenir. Nous avons 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 Windows, qui permettait de se connecter à deux machines distantes servant aux entraînements : 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 (SAN).

Nous avons choisi l'éditeur VS Code pour le développement, et Git comme gestionnaire de versions. Git m'a permis de synchroniser toutes les machines, et de créer des branches pour l'ajout de nouvelles fonctionnalités. Cela a eu pour avantage de toujours conserver une version stable pour être en capacité de démarrer des 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

Après la présentation du projet au début du stage, nous avons mis en place une stratégie visant à explorer l'espace des solutions que nous pouvions envisager. Elle comprend principalement la familiarisation avec des logiciels utiles, la recherche d'images et les entraînements.

3.2.1 Prise en main

Pour éviter les problèmes de dépendance rencontrés lors de la reprise du travail de l'ancien stagiaire, nous avons décidé de créer un environnement virtuel.

La création de cet environnement de développement a commencé par l'installation de WSL¹ puis des drivers CUDA², et enfin des librairies nécessaires à YOLOX et FiftyOne4.4.1. 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,

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

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

notamment les paramètres et les méthodes de data augmentation³ 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.

Nous avons documenté toutes ces étapes afin que notre travail puisse être repris par un autre développeur.

3.2.2 Datasets

La première étape pour améliorer le système était d'entraîner le réseau de neurones YOLOX sur un jeu de données plus important. Le début du stage était donc consacré à la recherche d'images de bateaux. N'ayant pas le temps ni les ressources pour annoter les images, nos recherches portaient exclusivement sur des datasets de détection existants.

L'entraînement de ce modèle nécessite un dataset au format COCO⁴; ces recherches sont donc accompagnées du développement de scripts de conversion pour les datasets qui ne sont pas au bon format.

3.2.3 Recherche d'optimisations

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, nous nous sommes basés sur les connaissances acquises lors de notre 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, nous avons proposé à l'équipe de procéder de la manière suivante : pour tester une hypothèse (par exemple, l'effet de la data augmentation), nous mettons en place deux entraînements. 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. Pour obtenir rapidement les réponses à nos questionnements, nous avons suggéré que ces expérimentations soient faites avec une sous-partie du dataset d'entraînement.

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

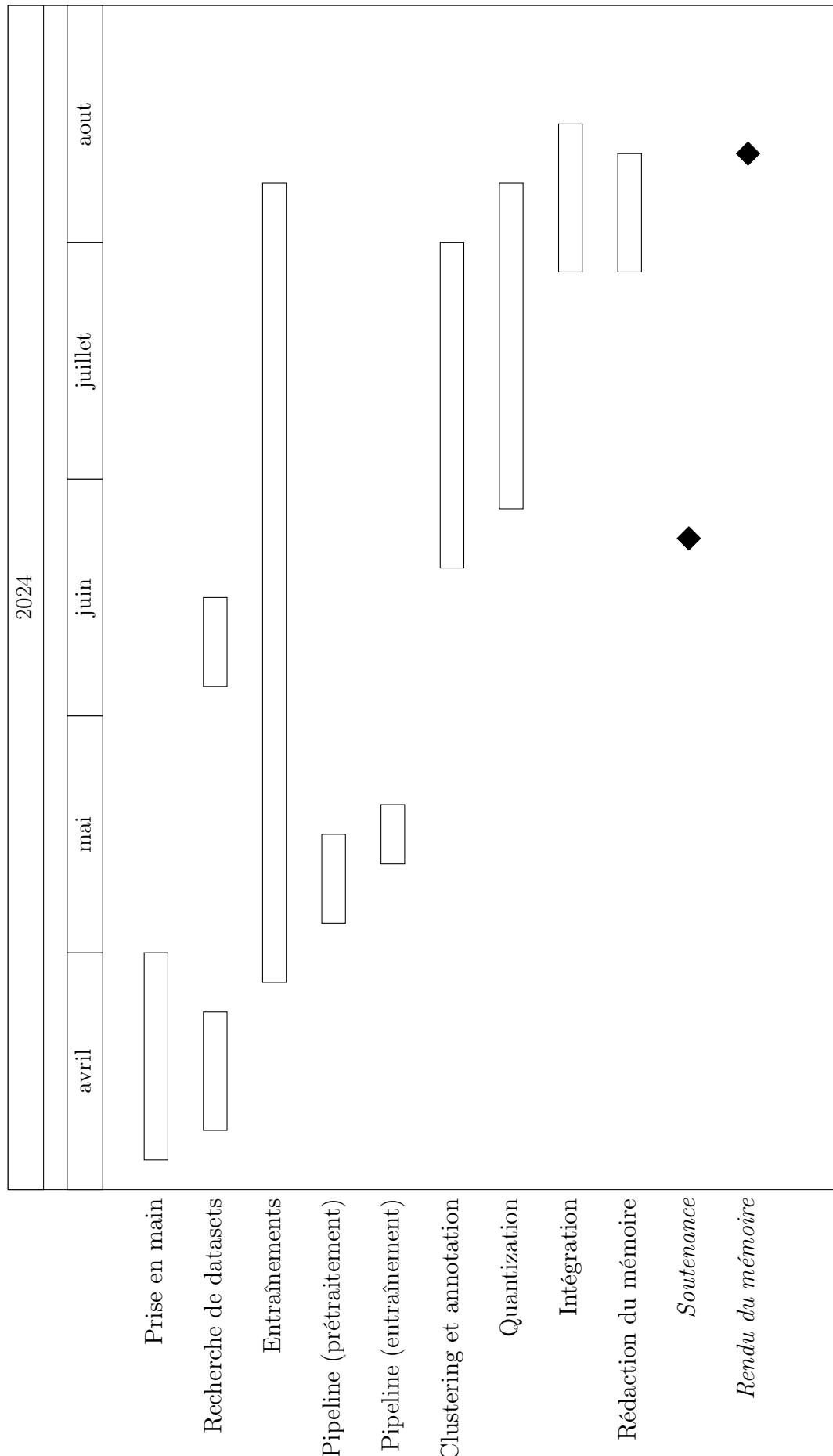
Après avoir optimisé les entraînements et donc le modèle, nous avons exploré d'autres méthodes, applicables en production, pour rendre la détection plus efficace.

3. La data augmentation est un ensemble de techniques qui consistent à dupliquer puis modifier les images du dataset pour augmenter sa taille et améliorer la généralisation du modèle.

4. 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.

3.2.4 Répartition du temps de travail

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



3.3 Organisation d'équipe

Le projet dont nous étions chargé étant à un stade de recherche, nous n'avons pas été soumis par l'entreprise aux tests unitaires, ni à des conventions de nommage de variables ou autres contraintes de génie logiciel.

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

Pour organiser notre temps de travail, une réunion SCRUM a lieu tous les matins à 9h30 avec tous les membres de l'équipe. Nous en profitons pour partager les tâches réalisées la veille, et les objectifs de la journée. Pour appuyer cette réunion, nous utilisons l'outil Trello⁵.

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 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`. Nous avons ainsi bénéficié de la résolution automatique de dépendances de `pip`.

Ce fichier a permis de porter l'environnement de développement sur toutes les machines à notre disposition, et permettra aux prochains développeurs de reprendre le projet.

4.2 Documentation initiale

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

4.2.1 COCO (Common Objects in COnText)

Le format COCO[1] 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 classes d'objets présentes dans le dataset ;
- les informations sur les images (nom, taille, etc.) ;
- les boîtes englobantes 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 paramètre 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 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 A. Mumuni and F. Mumuni[2].

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 mouvement.

4.2.2.2 Data augmentation dans YOLOX

Pour éviter de créer de la redondance, nous avons vérifié quelles méthodes étaient utilisées dans YOLOX.

La documentation (<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, dont mixup, hsv et mosaïc.

Mixup est une technique proposée par Zhang et al. [3] qui consiste à combiner deux images en réduisant leur opacité. La mosaïque est une technique qui consiste à assembler plusieurs images pour en former une seule.

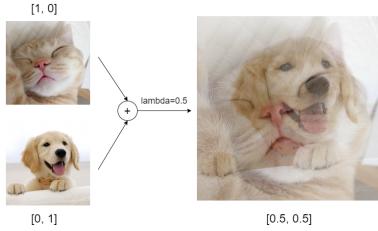


FIGURE 4.1 – Exemple de mixup

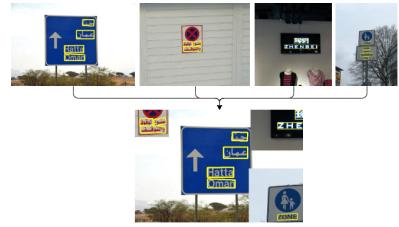


FIGURE 4.2 – Exemple de mosaïque

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. Il s'agissait d'une transformation par les canaux HSV¹.

Dans le code source, il est possible de spécifier un nombre d'époques sans augmentation. Nous n'avons pas trouvé d'informations ni 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 est 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 (*voir illustration en annexe A.1*) :

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

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

avec VP les vrai positifs, FP les faux positifs (erreurs de type 1) et FN les faux négatifs (erreurs de type 2).

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 prédiction est vraie ou fausse. Pour cela on utilise un seuil IoU (Intersection over Union, *voir annexe A.2*) :

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

Si l'IoU est supérieure à un certain seuil, la prédiction 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.

1. HSV (Hue, Saturation, Value) est une notation colorimétrique utilisée en informatique graphique, photographie et graphisme pour représenter les couleurs.

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 selon le modèle. Cette métrique n’est pas forcément cohérente avec notre perception de la scène : parfois, une bouée obtient une confiance élevée, tandis qu’un bateau obtient une confiance faible. Appliquer un seuil sur ce score permet de régler le compromis entre précision et rappel, mais ne suffit pas pour éliminer les faux positifs.

4.3 Datasets

Après avoir terminé la documentation, nous avons commencé la collecte d’images. L’entraînement d’un modèle d’apprentissage automatique nécessitant un grand nombre d’exemples, nous avons consacré plusieurs semaines à la recherche d’images de bateaux annotées. Le résultat de ces recherches est une combinaison de plusieurs datasets existants : ABOships-PLUS (*source* : <https://research.abo.fi/fi/datasets/aboships-plus-2>), boat_computer_vision_project [4], coco_boats (*source* : <https://cocodataset.org/#overview>, dataset_GLSD [5], lajolla_v5 (*source* : <https://universe.roboflow.com/roboflow-100>), marvel_single_v1 [6], mcships [7], mods [8], official_buoy_detection (*source* : <https://universe.roboflow.com/roboflow-100>), open_images [9], open_images_lighthouse [9], orda (*source* : <https://universe.roboflow.com/roboflow-100>), SeaDronesSee [10], SeaShips [11], singapore_maritime [12], SMD_Plus [13], vais_old [14], vessel_detection_v21 [15].

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

4.4.1 FiftyOne

FiftyOne est un outil open source permettant de visualiser des datasets, les filtrer, détecter les doublons, les erreurs de détection et bien d’autres fonctionnalités. Il a été d’une grande utilité lors de nos travaux, en particulier pour explorer facilement des datasets conséquents, 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 formats 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[16] 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 [17] : head, neck et backboneB.1. La partie backbone permet d'extraire les features² 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 [16] introduisent une nouvelle méthode pour améliorer la détection d'objets : la décorrélation 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.

Ces versions correspondent à différents paramètres de largeur et profondeur de certaines couches de neurones. Ces paramètres sont modifiables directement dans le fichier de configuration, et on peut observer leur effet sur la structureB.2 du modèle grâce à l'outil Netron (<https://netron.app>).

Les modèles les plus légers sont les plus rapides, mais ont une précision plus faible. L'année précédents, 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 exécuté les scripts de démonstration pour tester l'environnement et fait une première inférence³, nous avons entrepris d'entraîner le modèle sur nos datasets spécialisés pour les bateaux. Cette étape a été complexe, car certaines fonctions n'étaient 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 entraîner 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 :

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

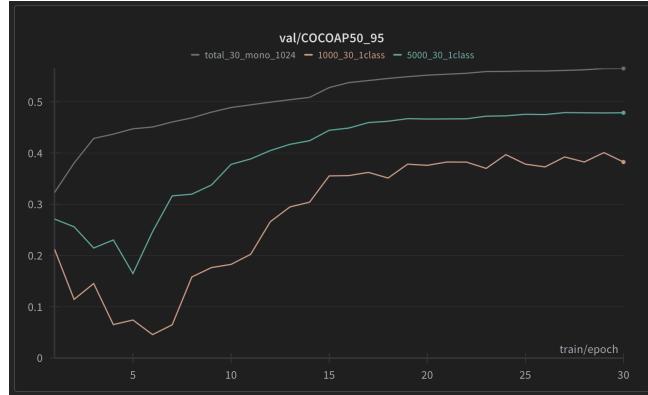


FIGURE 4.3 – Évolution de la précision en fonction de la taille du dataset

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

Naturellement, les scores sont généralement plus faibles lorsque les objets sont petits.

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 :

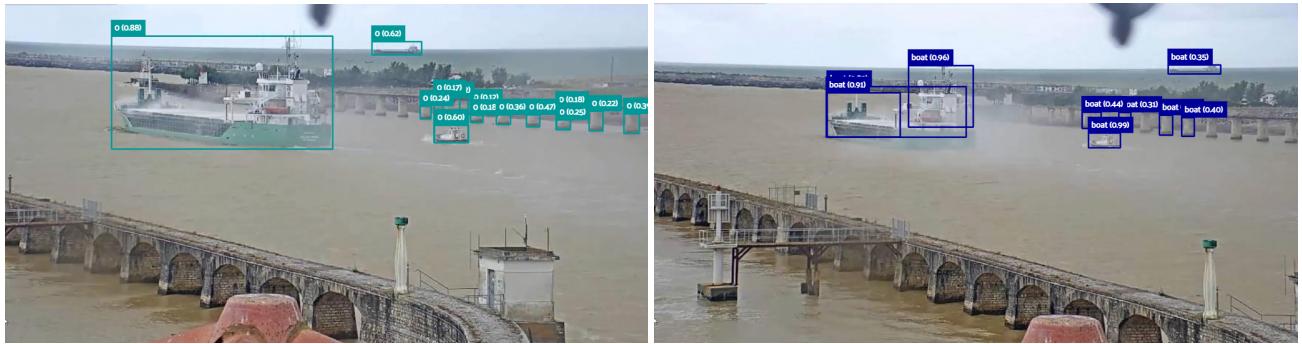


FIGURE 4.4 – Comparaison entre notre premier modèle entraîné (*à gauche*) et ResNet50 (*à droite*).

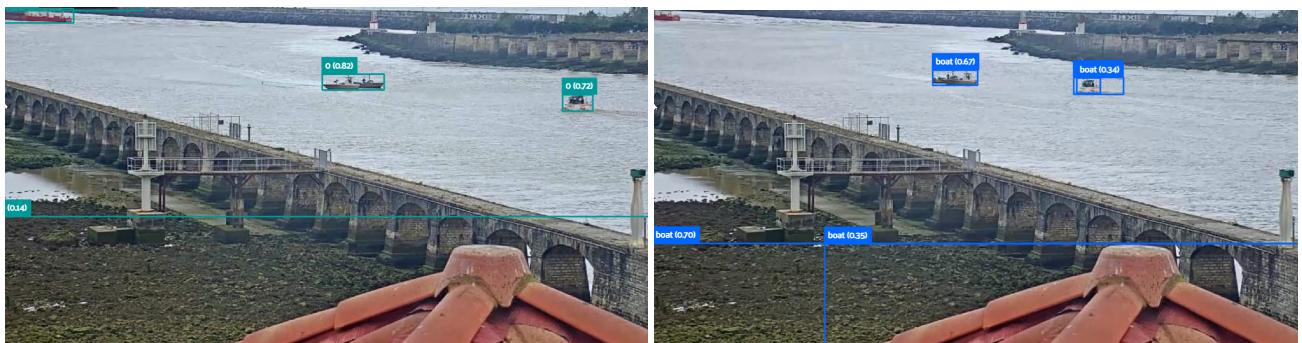


FIGURE 4.5 – Comparaison entre notre premier modèle entraîné (*à gauche*) et YOLOv8 (*à droite*).

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 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 prétraitement⁵ 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 convertis puis stockés au format COCO ;
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 ;
5. YOLOX est entraîné et Tensorboard permet d'observer les progrès ;
6. les logs d'entraînement et les poids⁶ sont exportés.

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 suppression 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 logger⁷.

L'entièreté du pipeline est contrôlée par un fichier de configuration 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 embeddings⁸ et régler le nombre de clusters⁹ pour la visualisation.

Pour l'entraînement, on trouve entre autres les paramètres pour le nombre d'époques¹⁰, la taille de batch¹¹, 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

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 stabilité des modèles de machine learning.

5. Le prétraitement, également appelé "preprocessing" 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.

6. Les poids du modèle représentent les valeurs des paramètres apprises lors de l'entraînement.

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

8. Les embeddings sont une représentation numérique simplifiée des images.

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

10. En général, une époque est définie comme le nombre de fois où le modèle passe sur l'ensemble des données d'entraînement.

11. Dans le contexte de l'apprentissage automatique, un batch désigne une sélection partielle des données d'entraînement utilisées pour former ou entraîner un modèle de ML.

En général, les batches sont utilisés pour réduire la quantité de mémoire nécessaire pour stocker et manipuler les données pendant l'apprentissage.

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 réuni un grand nombre d’images, nous avons exploré les datasets pour avoir une idée de ce qu’ils contenaient.

4.6.1.1 Labels

Pour commencer, chaque source d’image contenant des classes différentes. Les bateaux (notre classe d’intérêt) étaient parfois appelée "vessel", "ship", ou encore des noms plus précis, comme "sailboat" ou "warship". Ces classes sont souvent incompatibles entre elles, et concernent parfois des objets qui ne sont pas des bateaux.

Pour pouvoir rassembler ces images et les utiliser, nous avons dans un premier temps créé des dictionnaires de conversion (*voir exemple en annexe C*) 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 mêmes 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 surapprentissage¹². En effet, après avoir testé un entraînement sur ce genre de dataset, nous avons eu un résultat confirmant ce risque. Toutes les prédictions du modèles correspondaient presque exactement aux annotations d’origines du dataset. En particulier, nous avons retrouvé dans ces prédictions l’absence ou les erreurs d’annotations spécifiques aux images d’entraînement.

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 ajuster 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-ensembles d’un dataset, et rend l’exploration beaucoup plus facile.

12. 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 nouvelles.



FIGURE 4.6 – Exemple de carte des embeddings dans FiftyOne

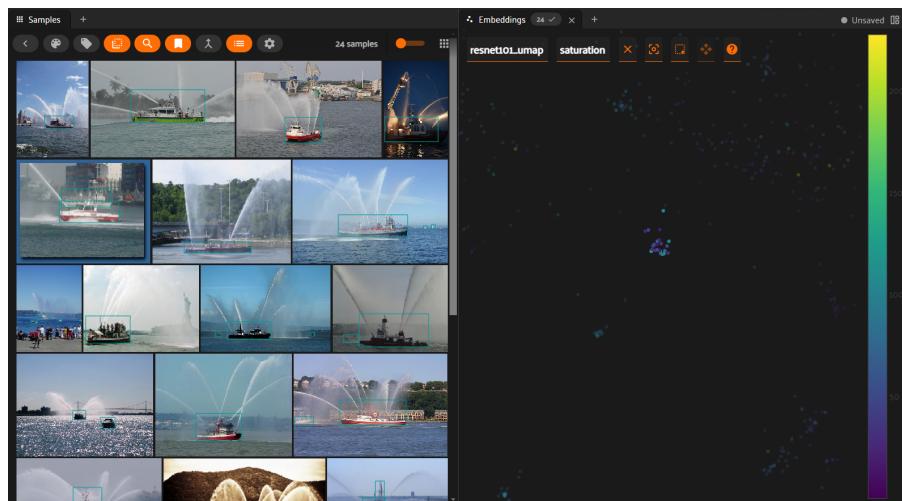


FIGURE 4.7 – Exemple de sélection d'un ensemble de points

Cette 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, et nous a permis de nous rappeler que notre perception est différente de celle d'un modèle de vision.

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 réflexions. La première est évidente : prédir 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 toute 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 quelques recherches[18][19], 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>*).

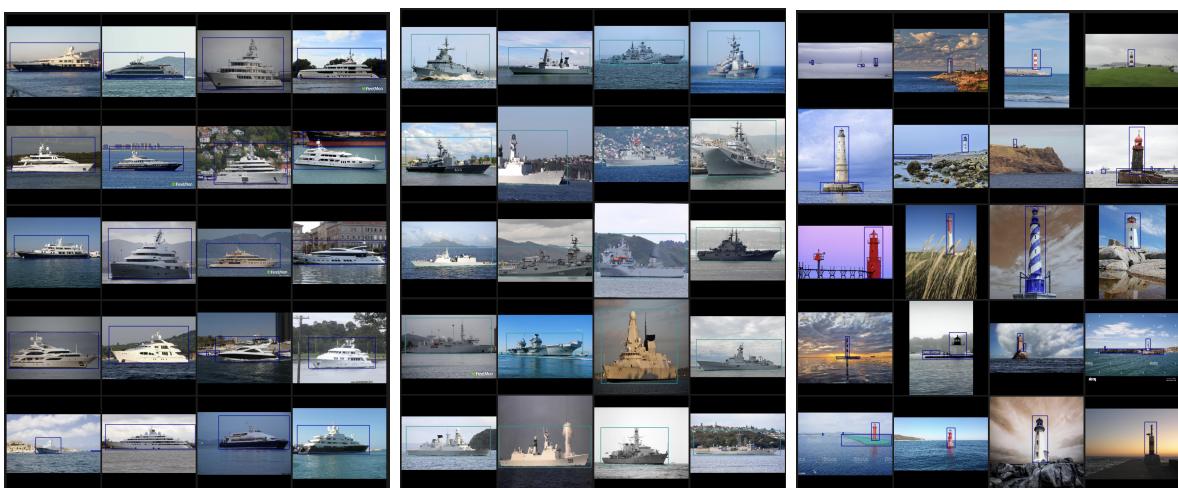


FIGURE 4.8 – Exemples de clusters

Il faut garder à l'esprit que les exemples ci-dessus sont des cas particuliers : la plupart des clusters contenaient 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 :

TABLE 4.1 – Nombre d'objets par classes

cargo	boat	sailboat	buoy	recreational	ferry	warship	fishing
35033	23648	13153	9282	8827	8272	4593	3374
cruise	lighthouse	luxury	breakwater	service	platform	bulker	
3172	2705	2145	2136	1080	273	51	

La liste des classes a été créée en collaboration avec mon maître de stage, qui est la personne ayant le plus de connaissances en termes d'architecture navale. Les définitions des différentes classes ont été documentées pour permettre des annotations futures (*voir annexe D*). Après l'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 (aire < 322), 34% sont moyens (322 < aire < 962), et 24% sont grands (aire > 962), selon les définitions 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.

Pour résumer cette section, nous avons constitué un dataset d'entraînement conséquent, créé différents filtres applicables aux images, et annoté un grand nombre de bateaux.

4.7 Entraînements

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, détaillant les méthodes de preprocessing et les paramètres utilisés, et une présentation des résultats à l'équipe. Les résultats de ces entraînements sont exposés au chapitre 5.

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 (PyTorch¹³), qui correspond à un dictionnaire contenant la structure et les poids du réseau de neurones. Ce fichier doit être converti au format onnx¹⁴ pour être utilisé pour l'inférence.

13. PyTorch est une librairie d'apprentissage automatique très populaire, disposant d'une API en Python.

14. 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.

4.8.2 Quantization

En plus de la conversion en onnx, nous avons entrepris d'appliquer une "quantization"¹⁵ 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 les processeurs de la marque, qui occupent la majorité des machines visées par les logiciels TimeZero. La participation à une conférence d'Intel sur le sujet nous a permis de rapidement comprendre les outils disponibles.

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 4.2 – Vitesse d'inférence (images par seconde)

Modèle	fp32 (précision originale)	int8
yolox_s	11.34	18.72

15. 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*).

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 [20], 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 principales expérimentations réalisées, généralement sous forme de couple d’entraînements. Sauf indications contraires (scores en italique), 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

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 5.1 – 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 [21] 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"¹, et une taille de batch trop grande saturera 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, 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 de l'entraînement.

5.2 Tuilage

D'après nos recherches, le tuilage est un moyen efficace d'augmenter les performances d'un modèle après entraînement. Il consiste à diviser l'image en plusieurs parties, afin d'éviter d'avoir à la compresser (et donc perdre en résolution). YOLOX contient des couches de convolution², et, selon notre raisonnement, un objet trop petit (contenant peu de pixels) risque d'être effacé par ces couches, et n'apporter aucune information utile à l'apprentissage du modèle. Afin de tester notre hypothèse, avons tenté d'appliquer le tuilage avant l'entraînement, et ainsi de réduire le nombre de petits objets dans le dataset. Nous avons pour cela utilisé SAHI [22].

Nous avons fait trois expérimentations comprenant deux entraînements chacune. Les datasets d'entraînement contenait 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 5.2 – Tuilage pré-entraînement, 3 tests 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.

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

2. La convolution est un opérateur mathématique qui permet de mettre en correspondance des signaux spatiaux avec un filtre ou un noyau, pour détecter des patterns ou des caractéristiques spécifiques.

	Images tuilées	Images originales
mAP	55.003	60.536
mAR	63.896	67.477

TABLE 5.3 – Tuilage pré-entraînement.

5.3 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 issues 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 5.4 – Tuilage avec ou sans conservation des images vides.

Nous tirons comme conclusion de cet entraînement qu’ajouter des images videss 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 dataset AB0ships-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)
<i>mAP</i>	41.828	40.856
<i>mAR</i>	53.520	53.475

TABLE 5.5 – Entraînement avec plusieurs classes.

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
<i>mAP</i>	53.895	88.126	67.593	73.628	71.923
<i>mAR</i>	59.922	90.230	71.718	76.653	75.692

TABLE 5.6 – Entraînement avec plusieurs classes (3 datasets).

La moyenne ci-dessus a été pondérée par le nombre de bateaux dans chaque classe.

boat
<i>mAP</i> 75.859
<i>mAR</i> 78.416

TABLE 5.7 – 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 D*).

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

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

TABLE 5.8 – Entraînement avec plusieurs classes (3 datasets).

On remarque que certaines 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. Nous avons supposé que les classes obtenant le plus faible rappel sont celles qui contiennent les objets les plus lointains ou difficiles à détecter. Le score de corrélation (Pearson) entre la taille des détections et la précision est de 0.60025, donc peu significatif. Malgré une faible corrélation, il faut faire attention à ne pas confondre les classes difficiles à prédire et les classes ne contenant que des petits objets.

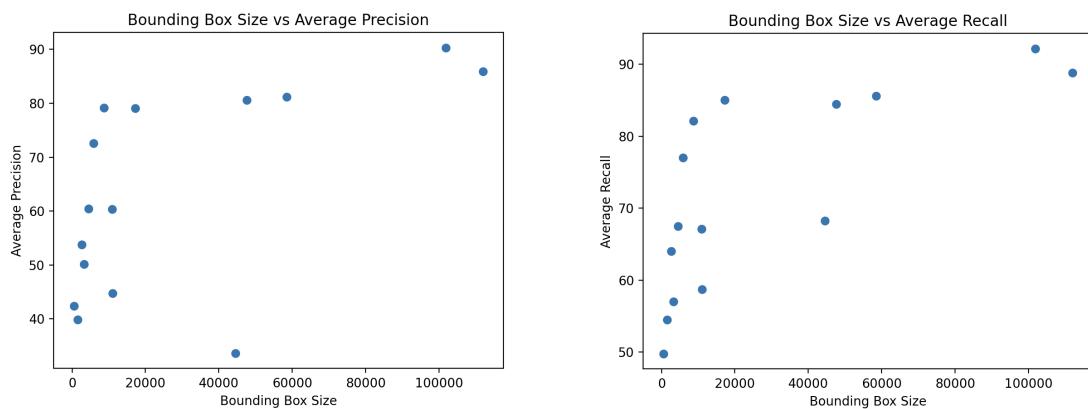


FIGURE 5.1 – Précision et rappel en fonction de la taille des objets.

5.5 Images faciles 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'images était de 11502 pour cet entraînement.

	marvel	images tirées au hasard
mAP	19.252	47.858
mAR	27.436	57.285

TABLE 5.9 – 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 5.10 – Comparaison avec notre meilleur modèle.

Ce tableau montre les progrès que nous avons fait dans la détection de navires, et permettent de valider l'efficacité de nos travaux durant le stage.

Chapitre 6

Conclusions

6.1 Solutions retenues

Parmi tous les essais que nous avons faits, augmenter le nombre d'images dans le dataset d'entraînement semble être le moyen le plus efficace d'améliorer la précision et le rappel du modèle.

En revanche, il est important que ces images soient le plus hétérogène possible, pour éviter le surapprentissage. Le seul filtre qui nous a permis d'obtenir de meilleurs scores en diminuant la taille du dataset est le filtre de similarité.

Les autres filtres, basés sur les tailles d'objets ou la qualité d'image, n'ont produit que des résultats négatifs.

Des recherches sont encore en cours pour trouver le modèle qui apporte le meilleur compromis entre précision et vitesse d'exécution : lors de l'intégration dans le logiciel TimeZero Coastal Monitoring, il sera couplé à un logiciel de suivi de cibles qui dépend du temps d'inférence.

6.2 Bénéfices du stage

Les bénéfices du stage se mesurent principalement grâce aux scores obtenus par YOLOX : la précision et le rappel ont respectivement été augmentés de 85% et 40%.

La documentation produite a également été d'une grande aide, car elle a apporté à l'équipe une compréhension beaucoup plus fine de tout l'environnement d'apprentissage automatique.

Enfin, ce projet pourra être repris au-delà du stage, pour permettre des entraînements avec de nouvelles images.

De notre côté, ce stage a eu un impact très positif sur nos compétences. En étant responsable de toute la chaîne de création d'un modèle d'apprentissage automatique, au sein d'une entreprise produisant des algorithmes plus "classiques", nous avons eu la chance de maîtriser toutes les étapes, et de grandement affiner nos capacités de recherches.

6.3 Développement durable

Mes travaux sur l'algorithme de détection de navires ont nécessité l'utilisation de cartes graphiques, qui consomment 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.com/>).

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

L'ajout d'images au dataset d'entraînement est un des enjeux principaux pour continuer le développement de ce projet. La famille de modèles YOLO était en constante augmentation, il est possible que de nouveaux modèles voient le jour, ce qui offre également des perspectives d'amélioration.

Le reste du travail comportera probablement des optimisations logicielles inhérentes au métier maritime, comme la possibilité pour les utilisateurs de régler, dans le champ de la caméra, des zones exemptées de détections (berges, parkings, arbres...).

Enfin, l'augmentation des performances du matériel embarqué permettra peut-être l'utilisation de modèle plus complexe achevant de meilleures performances.

Bibliographie

- [1] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, “Microsoft coco : Common objects in context,” Feb. 2015. arXiv :1405.0312 [cs].
- [2] A. Mumuni and F. Mumuni, “Data augmentation : A comprehensive survey of modern approaches,” *Array*, vol. 16, p. 100258, Dec. 2022.
- [3] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, “mixup : Beyond empirical risk minimization,” Apr. 2018. arXiv :1710.09412 [cs, stat].
- [4] brac university, “boat detection dataset.” <https://universe.roboflow.com/brac-university-w4ehn/boat-detection-4tmst>, apr 2023. visited on 2024-08-09.
- [5] Z. Shao, J. Wang, L. Deng, X. Huang, T. Lu, F. Luo, R. Zhang, X. Lv, C. Dang, Q. Ding, and Z. Wang, “Glsd : The global large-scale ship database and baseline evaluations,” 2021.
- [6] Asian Conference on Computer Vision (ACCV), *Marvel : A Large-Scale Image Dataset for Maritime Vessels*, 2016.
- [7] Y. Zheng and S. Zhang, “Mcships : A large-scale ship dataset for detection and fine-grained categorization in the wild,” in *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, 2020.
- [8] B. Bovcon, J. Muhovič, D. Vranac, D. Mozetič, J. Perš, and M. Kristan, “Mods – a usv-oriented object detection and obstacle segmentation benchmark,” 2021.
- [9] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, S. Kamali, M. Malloci, J. Pont-Tuset, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy, “Openimages : A public dataset for large-scale multi-label and multi-class image classification.,” *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.
- [10] L. A. Varga, B. Kiefer, M. Messmer, and A. Zell, “Seadronesee : A maritime benchmark for detecting humans in open water,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2260–2270, 2022.
- [11] Z. Shao, W. Wu, Z. Wang, W. Du, and C. Li, “Seaships : A large-scale precisely annotated dataset for ship detection,” *IEEE transactions on multimedia*, vol. 20, no. 10, pp. 2593–2604, 2018.
- [12] D. K. Prasad, D. Rajan, L. Rachmawati, E. Rajabaly, and C. Quek, “Video processing from electro-optical sensors for object detection and tracking in maritime environment : A survey,” Nov. 2016. arXiv :1611.05842 [cs].
- [13] C. Lee and S. Lee, “Vulnerability of clean-label poisoning attack for object detection in maritime autonomous surface ships,” *Journal of Marine Science and Engineering*, vol. 11, p. 1179, 06 2023.
- [14] *2015 IEEE Conference on Computer Vision and Pattern Recognition workshops (CVPRW 2015) : Boston, Massachusetts, USA, 7 - 12 June 2015*. Piscataway, NJ : IEEE, 2015.

- [15] G. Matasci, J. Plante, K. Kasa, P. Mousavi, A. Stewart, A. Macdonald, A. Webster, and J. Busler, “Deep learning for vessel detection and identification from spaceborne optical imagery,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. V-3–2021, p. 303–310, June 2021.
- [16] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, “Yolox : Exceeding yolo series in 2021,” Aug. 2021. arXiv :2107.08430 [cs].
- [17] J. Redmon and A. Farhadi, “Yolov3 : An incremental improvement,” Apr. 2018. arXiv :1804.02767 [cs].
- [18] D. Xu and Y. Tian, “A comprehensive survey of clustering algorithms,” *Annals of Data Science*, vol. 2, p. 165–193, June 2015.
- [19] H. Yin, A. Aryani, S. Petrie, A. Nambissan, A. Astudillo, and S. Cao, “A rapid review of clustering algorithms,” Jan. 2024. arXiv :2401.07389 [cs].
- [20] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, “Bytetrack : Multi-object tracking by associating every detection box,” Apr. 2022. arXiv :2110.06864 [cs].
- [21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [22] F. C. Akyon, S. O. Altinuc, and A. Temizel, “Slicing aided hyper inference and fine-tuning for small object detection,” in *2022 IEEE International Conference on Image Processing (ICIP)*, p. 966–970, Oct. 2022. arXiv :2202.06934 [cs].
- [23] J. Terven, D. M. Cordova-Esparza, A. Ramirez-Pedraza, and E. A. Chavez-Urbiola, “Loss functions and metrics in deep learning,” Sept. 2023. arXiv :2307.02694 [cs].

Table des figures

1.1	Exemple de vue radar, AIS et camera dans TimeZero Coastal Monitoring	2
4.1	Exemple de mixup	11
4.2	Exemple de mosaïque	11
4.3	Évolution de la précision en fonction de la taille du dataset	14
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	18
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.	27

Liste des tableaux

4.1	Nombre d'objets par classes	19
4.2	Vitesse d'inférence (images par seconde)	21
5.1	YOLOX entraîné par notre équipe comparé à YOLOX entraîné par les auteurs.	23
5.2	Tuilage pré-entraînement, 3 tests avec petits datasets (moyenne des résultats).	24
5.3	Tuilage pré-entraînement.	25
5.4	Tuilage avec ou sans conservation des images vides.	25
5.5	Entraînement avec plusieurs classes.	25
5.6	Entraînement avec plusieurs classes (3 datasets).	26
5.7	Entraînement avec une classe (3 datasets).	26
5.8	Entraînement avec plusieurs classes (3 datasets).	26
5.9	Entraînement avec image facile à identifier.	27
5.10	Comparaison avec notre meilleur modèle.	28

Appendices

Annexe A

Métriques

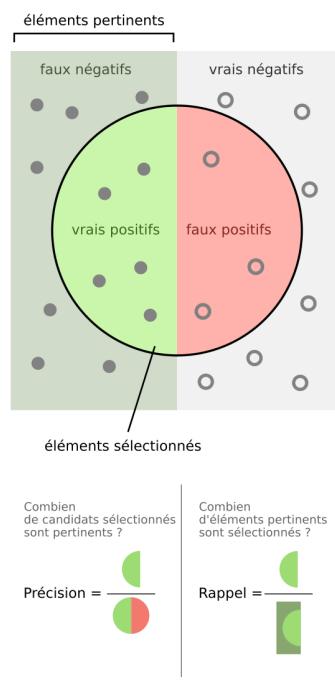


FIGURE A.1 – Précision et rappel (*source* : https://fr.wikipedia.org/wiki/Précision_et_rappel)

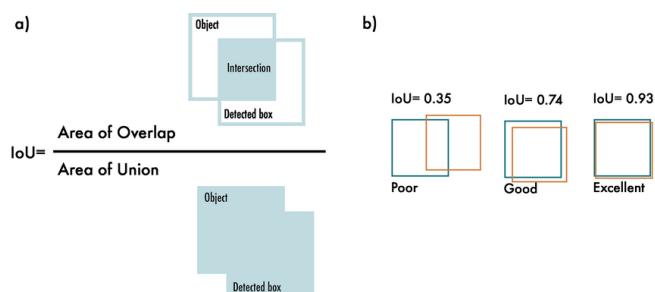


FIGURE A.2 – Intersection over Union (*source* : [23])

Annexe B

Structure de YOLOX

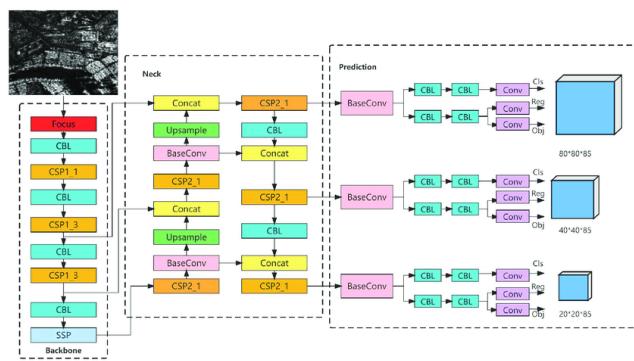


FIGURE B.1 – Structure de YOLOX.

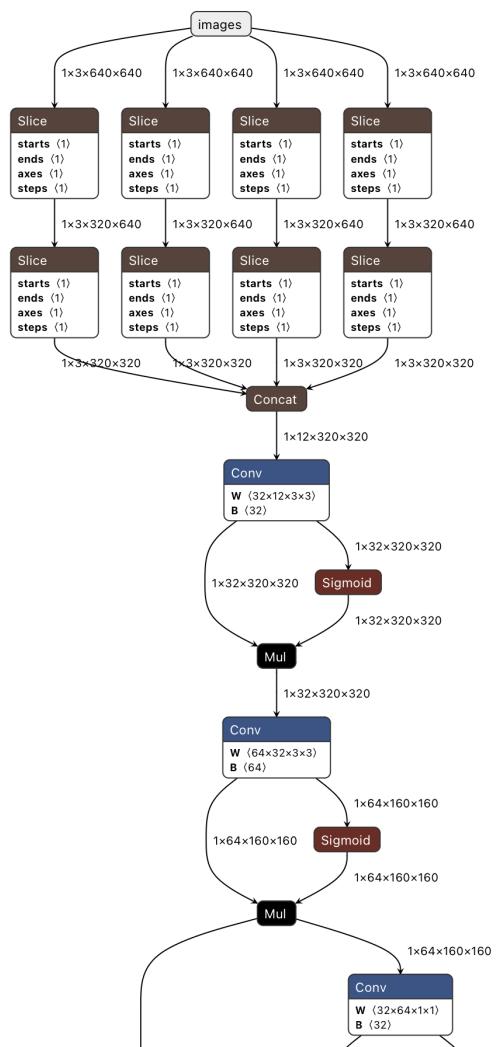


FIGURE B.2 – Structure de YOLOX dans Netron.

Annexe C

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 D

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 E

Interfaces

Interface de FiftyOne pour l'annotation de datasets :

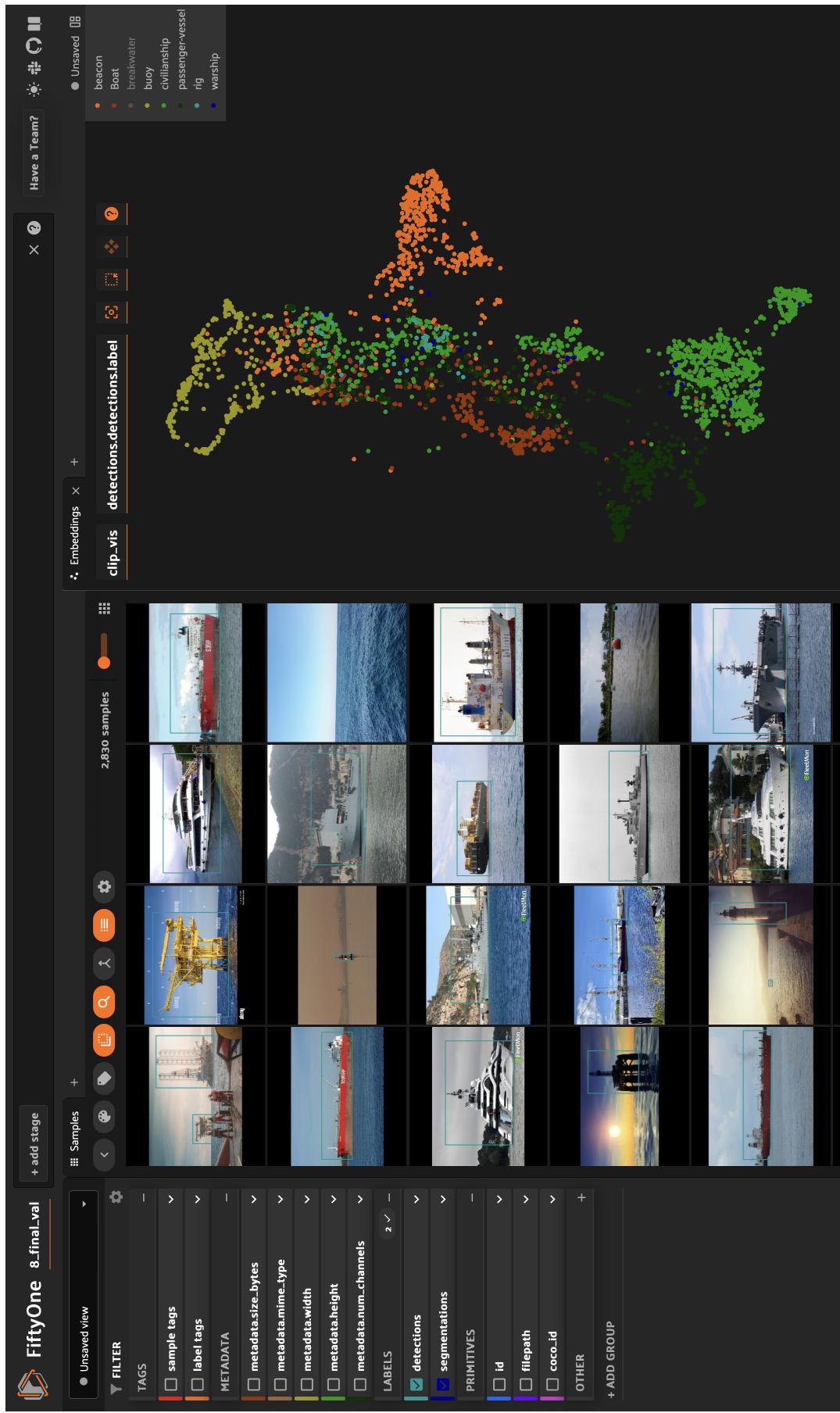


FIGURE E.1 – Interface pour l'annotation de datasets (carte de similarité à droite, une couleur par cluster K-Means)

Debugger Tool :

