



MASTER 1 INFORMATIQUE - TER

# Système de reconnaissance d'individus pour une espèce de baleines noires en danger d'extinction

*Hugo Maitre  
Adrien Linares*

Encadrant :  
Pr. William PUECH  
Rapporteur(s) :  
Pr. Pascal PONCELET

23 mai 2022

# Table des matières

1	Préambule . . . . .	3
1.1	Contexte général . . . . .	3
1.1.1	Données . . . . .	4
1.1.2	Résultat attendu par Kaggle . . . . .	7
1.2	Organisation du travail . . . . .	7
1.2.1	Gestion de projet . . . . .	7
1.2.2	Communication Travail de Groupe . . . . .	8
1.2.3	Répartition des tâches et planning . . . . .	8
1.3	Motivations . . . . .	9
1.4	Difficultés majeures . . . . .	10
2	Réalisation du projet . . . . .	12
2.1	Introduction sur la stratégie effectuée . . . . .	12
2.2	Choix techniques . . . . .	12
2.2.1	Python et ses librairies . . . . .	12
2.2.2	Stockage des images . . . . .	12
2.2.3	Boostage des calculs GPU . . . . .	13
2.3	Développement des modèles de deep learning . . . . .	13
2.3.1	Object Localization Yolo v3 . . . . .	13
2.3.1.1	Premiers prétraitements . . . . .	16
2.3.1.2	Développement du premier modèle . . . . .	17
2.3.1.3	Évaluation des résultats . . . . .	20
2.3.1.4	Problèmes et seconds prétraitements . . . . .	21
2.3.2	Head Aligner Yolo v4 et Algo d'Alignement . . . . .	24
2.3.2.1	Premiers prétraitements . . . . .	24
2.3.2.2	Développement du second modèle . . . . .	25
2.3.2.3	Résultats . . . . .	26
2.3.2.4	Alignment des images . . . . .	29
2.3.3	Système de reconnaissance faciale . . . . .	31
2.3.3.1	Premiers prétraitements et Data Augmentation . . . . .	34
2.3.3.2	Flipping . . . . .	35
2.3.3.3	Rescaling . . . . .	35

2.3.3.4	Translation . . . . .	36
2.3.3.5	Channel shift . . . . .	36
2.3.3.6	Regard sur notre Data Augmentation . . .	37
2.3.4	Développement du second modèle . . . . .	39
2.3.4.1	Résultats . . . . .	42
3	Conclusion . . . . .	45
3.1	Perspectives d'amélioration . . . . .	45
3.2	Ce qu'on a appris . . . . .	45
	<b>Bibliographie</b>	<b>46</b>

# 1 Préambule

## 1.1 Contexte général

Les baleines noires de l'Atlantique Nord sont une espèce de cétacés qui se distinguent des autres espèces de baleines entre autres par des callosités blanches se trouvant sur leurs têtes, qui est en fait de la peau kératinisée. Cette espèce de baleine, comme son nom l'indique, vit dans l'Océan Atlantique et à plus forte raison entre les degrés 20 et 60 de latitude Nord avec des déplacements variant selon les saisons.

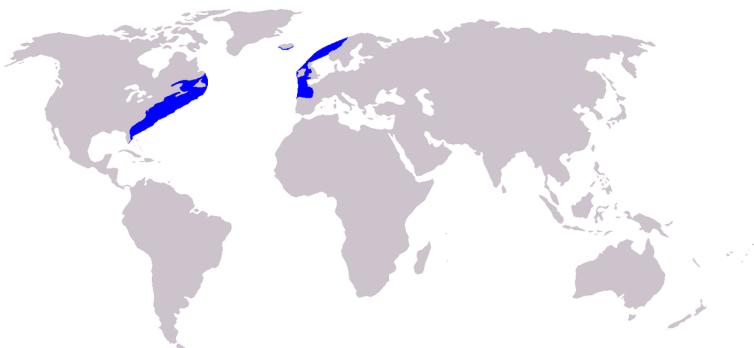


FIGURE 1 – Répartition géographique de l'espèce

Elles sont connues pour leurs comportements sociable vis à vis de l'humain. Elle se déplace lentement et souvent à la surface de l'eau. On estime qu'entre 1991 et 2007, 50% des mortalités seraient dues à des collisions avec les bateaux[2]. Elle est également chassée principalement pour son huile, ce qui fait d'elle l'une des grandes baleines les plus menacées au monde. Aujourd'hui, sa population est estimée à moins de 450 individus, elle est classifiée par l'IUCN<sup>1</sup> comme en danger critique d'extinction.

---

1. Union internationale pour la conservation de la nature



FIGURE 2 – Statut de conservation établi par l'IUCN

C'est dans ce contexte que la NOAA (National Oceanic and Atmospheric Administration) a proposé un challenge d'une valeur de \$10.000 en 2016, afin de construire un système de reconnaissance automatique de ces baleines à partir de méthodes de machine learning. En effet, seulement quelques personnes dans le monde savent aujourd’hui distinguer ces baleines à l’œil nu, la majorité étant des chercheurs expérimentés dans le domaine. C'est de plus un processus qui prend beaucoup de temps. Cet outil sera donc nécessaire pour surveiller bien plus aisément la population de baleines noires de l'Atlantique Nord et de même avoir accès à son historique de santé. Le challenge a été publié sur Kaggle, site web rassemblant une communauté de data scientists et professionnels de l'apprentissage automatique, leur permettant de participer à des concours sur des problèmes réels, à partir de données réelles. L'abréviation AN sera utilisée dans la suite de ce rapport pour « Atlantique Nord ».

### 1.1.1 Données

Nous avons à notre disposition 11468 images aériennes de baleines noires de l'A.N. La qualité des images est très aléatoire, on constate par exemple des images 200\*200 pixels côtoyant des images de très haute définition, images de 4000\*3000 pixels. L'exposition des images est aussi très aléatoire même si l'on constate que 90% des images sont correctement exposées. Certaines images souffrent de contraste trop fort ou inversement apparaissent trop “lisses”. On peut voir ci-dessous ces

écart sur un échantillon non aléatoire du dataset.

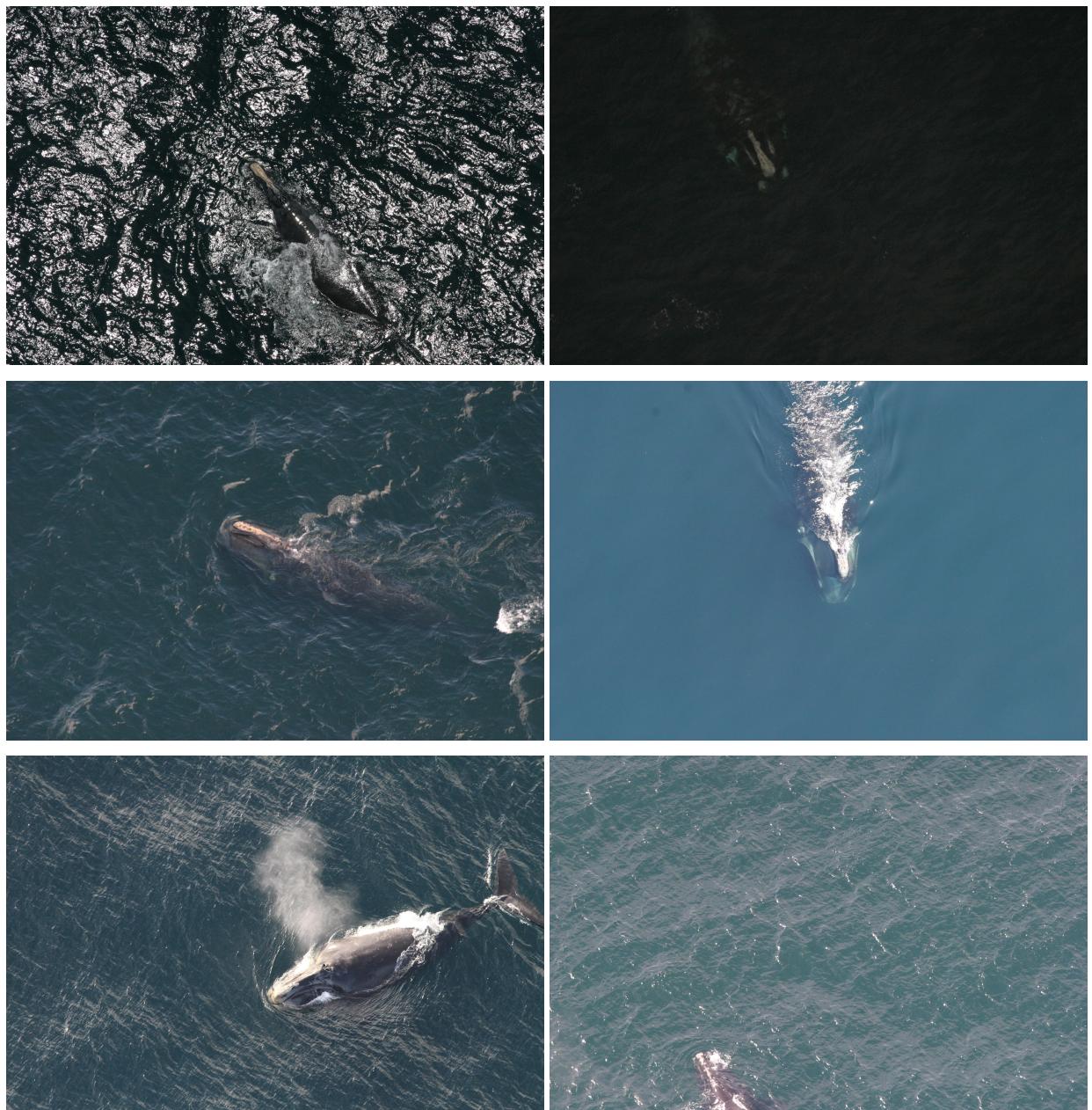


FIGURE 3 – Echantillon non aléatoire du Dataset

Parmi le dataset nous avons seulement 4544 images qui sont labellisées, c'est-à-dire environ 40% du dataset. Nous n'avons bien évidemment pas accès aux labels des images restantes qui serviront à évaluer les performances de notre classifieur lors de la soumission de nos prédictions en ligne sur kaggle. On comprend déjà la difficulté qui nous attend : le manque de données. On compte 447 individus de baleines noires répertoriées ce qui est probablement très proche du nombre de baleines noires de l'A.N existantes. Parmi les images labellisées on constate un nombre très hétérogène d'images attribuées à nos baleines. En effet certaines d'entre elles sont des "célébrités" ayant une cinquantaine d'images rien que pour elle mais la majorité des individus ont seulement entre 1 et 10 images.

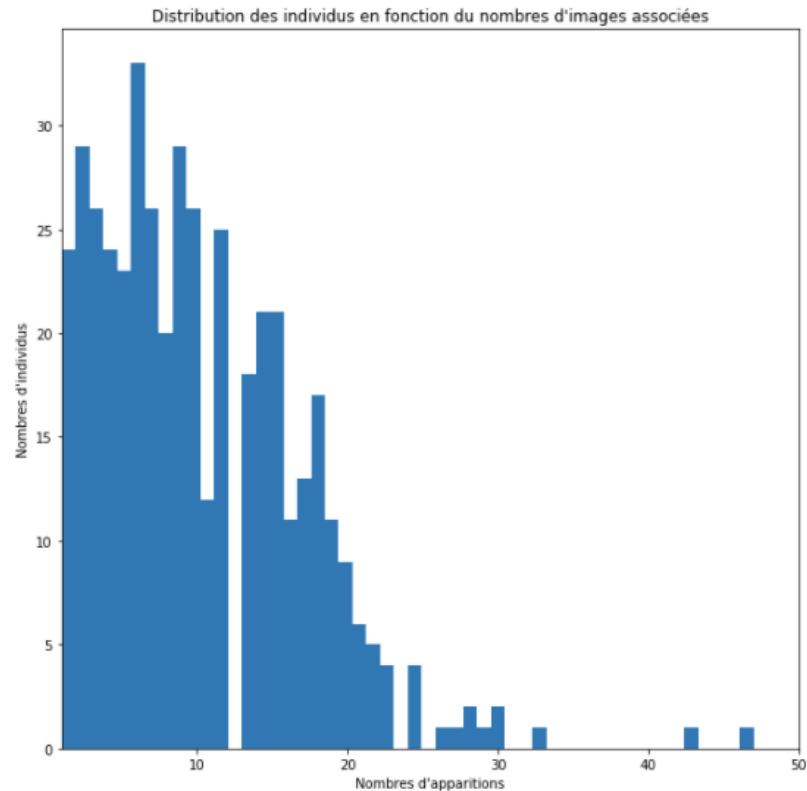


FIGURE 4 – Visualisation de l'hétérogénéité des occurrences

### 1.1.2 Résultat attendu par Kaggle

La solution s'imposant assez naturellement pour ce type de problème est d'utiliser une série de réseaux de neurones convolutifs. La plateforme met à notre disposition une "vérité terrain" c'est-à-dire l'identité de 4544 images, et nous évaluera sur les prédictions effectuées sur les 6924 images restantes. La plateforme de Kaggle nous évaluera à la fin sur la précision, f1-score et rappel de notre modèle. À titre d'exemple, le modèle ayant le mieux performé a obtenu un f1-score d'environ 80%[3].

## 1.2 Organisation du travail

### 1.2.1 Gestion de projet

Nous avons eu l'idée de nous servir de Miro, outil collaboratif de travail mais qui a été au final très peu utilisé. En ce qui concerne la gestion de projet, nous avions initialement choisi d'adopter la méthode agile qui nous a semblé être la plus adaptée.

Nous n'avions pas fixés de rôle à chacun au départ même si durant le projet les rôles se sont définis comme suit :

- **Hugo Maître** : chef de projet et développement des modèles, data preparation, redaction rapport.
- **Adrien Linares** : data augmentation/visualisation, annotations, aide à la rédaction du rapport.
- **Thomas Lecampion** :

### **1.2.2 Communication Travail de Groupe**

La communication a été assez difficile dans le groupe sachant qu'une personne nous a quittés assez vite (Thomas Lecampion), nous nous sommes donc retrouver à deux sur le projet ce qui nous a poussé à revoir un peu nos objectifs finaux même si le travail fournit n'est pas loin de nos attentes initiales.

### **1.2.3 Répartition des tâches et planning**

Afin de mener à bien ce projet, nous avons utilisé différents outils nécessaires à l'organisation des tâches et de l'avancement dans le temps du projet. Nous avons premièrement utilisé Gantt, ci dessous on peut voir le planning prévisionnel de début de parcours :

Je (Hugo Maître) me suis occupé de mettre en place les réseaux de neurones et de la rédaction du rapport. Ainsi que la préparation des données et le développement des programmes permettant l'alignement, le crop et autres manipulations sur les images. J'ai aussi pris le rôle de chef de projet en essayant au maximum de donner une direction viable à notre projet et fixer des deadlines.

Adrien Linares s'est occupé de constituer nos jeux d'apprentissage. Il a donc réalisé la grande majorité des annotations de nos images. Il s'est aussi chargé de faire de la "data augmentation" sur le dernier jeux d'apprentissage donnée à "faceNet". Mais également de l'aide à la rédaction du rapport.

Thomas lecampion nous a un peu aidé lors de la première session d'annotations ( sur les box englobant la baleine entière) avant de quitter le projet.

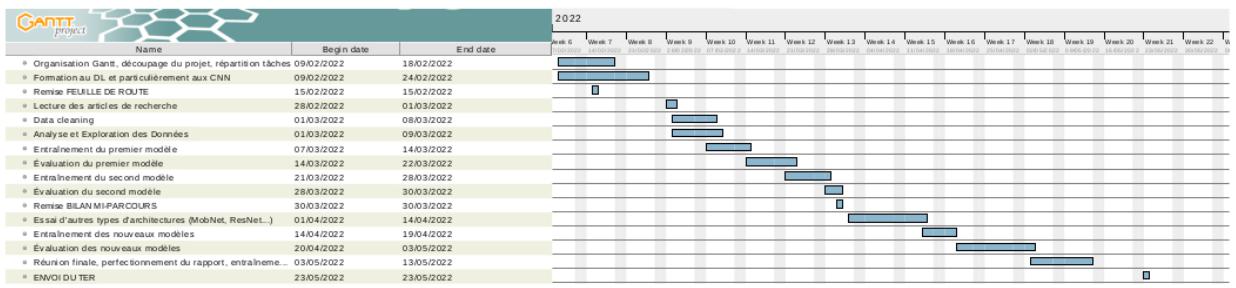


FIGURE 5 – Planning prévisionnel

### 1.3 Motivations

Nous avons recueilli les témoignages au début du TER de chacun d'entre nous afin de savoir pourquoi nous avons choisi un tel projet.

- **Thomas Lecampion :** J'ai choisi de rejoindre mes collègues sur ce TER pour m'initier au machine learning.
- **Adrien Linares :** Ce projet m'a semblé le compromis idéal sur des choses que j'affectionne particulièrement, le domaine de la biodiversité, du deep learning que je ne connaissais pas et du traitement d'images. Il me servira en effet de tremplin pour de futurs stages et/ou emploi en m'apprenant davantage sur ces domaines.
- **Hugo Maître :** Partant de 0 en deep learning, je me suis intéressé très vite au sujet au début de cette année. J'ai donc eu envie au travers d'un projet conséquent comme celui-ci d'en apprendre plus. En particulier puisque celui-ci porte sur le domaine de la vision par ordinateur qui est le domaine de l'intelligence artificielle sur lequel je souhaite me spécialiser.

## 1.4 Difficultés majeures

Dès le commencement, nous avons pu distinguer sur le projet trois difficultés majeures :

Premièrement, il est nécessaire d'uniquement récupérer les informations pertinentes sur chacune des images de notre dataset, isoler les baleines. Autrement dit, il nous faut un moyen de localiser et récupérer la zone autour d'une baleine présente sur une image. La difficulté à première vue réside dans le fait que les baleines sont prises sous différents angles de vues plus ou moins loin et n'occupent donc pas du tout le même espace sur l'image. Pour cela nous envisageons d'entraîner un réseau de neurones (certainement un des modèles YOLO) capable de détecter automatiquement des objets sur une image. Une fois la baleine détectée et isolée il faudra certainement établir un deuxième réseau de neurones capable de reconnaître les différentes caractéristiques de la baleine, probablement la tête et l'évent. Là aussi, en regardant un petit peu nos images, on s'aperçoit que l'écume blanche sur celles-ci va grandement compliquer notre tâche puisque l'évent<sup>2</sup> ou le bout de la tête sont des zones qui apparaissent aussi blanches comme l'écume, aussi bien qu'il est parfois difficile même pour un humain de les repérer correctement sur l'image. De plus, les baleines peuvent se présenter sous différents profils. Ce qui rend leur identification difficile même pour un expert. \*images de baleines : (image où le dos est net, gueule ouverte, etc...)\*

Ensuite, nous devrons effectuer quelques traitements sur les images réduites obtenues afin d'en faire de véritables photos d'identités, standardisées à la manière d'image d'identité humaine où les caractéristiques des baleines devront matcher une certaine zone de l'image. Il s'agira de corriger l'orientation des têtes. La difficulté n'est pas tant l'opération en elle-même mais plutôt de savoir de quel angle

---

2. narine double des cétacés

l'image doit-elle être orientée. Pointer l'emplacement exact du bout de la tête et de l'évent nous donne l'angle correspondant par de simples calculs. Enfin, il y a de fortes différences d'intensité lumineuse, de contrastes d'une image à l'autre. Un rehaussement de contraste par égalisation d'histogramme ainsi qu'une correction de l'exposition des images doit sans doute considérablement réduire les écarts.

Enfin, il faut implementer un dernier réseau bien différent des 2 autres pour la tâche de reconnaissance faciale. Ce réseau doit être capable de détecter quel est l'individu donné en entrée. Le principal problème de cette partie provient du grand déséquilibre de la quantité de nos données comme expliqué, on peut d'ailleurs le voir sur le diagramme (partie 1.1.1). De plus, les baleines apparaissent souvent dans des conditions bien différentes. Certaines ouvrent la gueule, sont entièrement sous l'eau ou ne possèdent qu'une photo en basse résolution, trop sombre avec un gros problème de mise au point...

## 2 Réalisation du projet

### 2.1 Introduction sur la stratégie effectuée

### 2.2 Choix techniques

#### 2.2.1 Python et ses librairies

Parmi les différentes options qui s'offraient à nous, le langage de programmation Python semblait de loin être le plus adapté pour les taches de Computer Vision. Le langage dispose d'une immense communauté surtout dans le domaine du deep learning appliquée à l'image, il propose des solutions rapides et faciles à mettre en place.

Parmi les librairies souvent utilisées j'en citerai quelques-unes. Opencv : pour toutes les tâches de manipulations d'images, crop rotation ainsi que des opérations plus complexes (homographie, corrections d'histogramme).

Numpy : Librairie permettant la manipulation plus rapide et performante de tableaux (et donc d'images) par le processus de "vectorisation".

Matplotlib : une des plus grandes librairie de la communauté, très utilisée pour la visualisation de données, pour tracer des graphiques etc...

Enfin tensorflow, librairie utilisée pour la mise en place de réseau de neurones.

#### 2.2.2 Stockage des images

L'ensemble des données représente environ 10 GB d'images ce qui n'est pas si conséquent que cela mais le problème étant qu'aucun de nous n'avait un ordinateur assez performant pour entraîner les modèles. Le passage par un outil tiers s'est donc imposé. J'ai donc dû charger sur un Google Drive plusieurs GB d'images en ligne pour permettre un entraînement via des GPU plus puissants.

### **2.2.3 Boostage des calculs GPU**

Les algorithmes de deep learning utilisés ont tous demandé à la fois beaucoup de puissance de calcul GPU mais aussi beaucoup de RAM GPU. J'ai donc souscrit à un abonnement Google Colab Pro qui met à disposition ce type de ressources (sans être trop limité). Tous les entraînements des modèles se sont donc effectués sur Google Colab.

## **2.3 Développement des modèles de deep learning**

### **2.3.1 Object Localization Yolo v3**

Le premier réseau implémenté va servir à détecter la baleine sur l'image, comme les baleines sont photographiées de plus ou moins loin, cette étape est cruciale pour isoler seulement l'information qui nous est nécessaire sur l'image.

Nous nous intéressons dans un premier temps à l'algorithme Yolo "You Only Look Once". Cet algorithme est notamment très utilisé pour des applications de détection d'objets sur des images ou vidéos. Sa réputation est en partie due à sa rapidité et sa précision. Il est aujourd'hui copieusement utilisé dans de nombreux domaines comme celui des voitures autonomes ou du tracking d'objet sur vidéo.

Les créateurs mettent à disposition de nombreux datasets pour entraîner un modèle sur des classes assez courantes (voitures, humains, ...) mais comme vous vous en doutez certainement, le réseau n'a pas été entraîné sur des baleines noires de l'A.N. Nous allons donc utiliser ce réseau qui est pré-entraîné sur des datasets classiques ImageNet pour le premier algorithme présenté dans le papier de recherche, puis l'entraîner avec nos propres objets.

L'algorithme fonctionne à l'aide d' "anchor boxes" c'est à dire de petites boites rectangulaire qui vont aider à la détection de features qui vont aider à la détection de l'objet. Elles sont définies comme suit :

$$y = \begin{pmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ . \\ . \\ c_n \end{pmatrix}$$

$p_c$  = 1 s'il y a un objet, 0 sinon

$b_x, b_y$  les coordonnées du centre du rectangle

$b_w, b_h$  la largeur et hauteur

$n = \#\text{classes}$

Par exemple Yolov3 utilise 9 formes possibles pour les anchor boxes. Il s'avère que la plupart de ces boîtes de délimitation ont certains rapports entre leur hauteur et leur largeur. Donc, au lieu de prédire directement une boîte englobante (bounding box), YOLOv3 prédit des ensembles de boîtes avec des rapports hauteur sur largeur particuliers, ces ensembles de boîtes pré-déterminés sont les "anchor box". YOLOv3 dispose de trois couches finales, la première a une dimension divisée par 31 par rapport à l'image initiale, la deuxième par 16 et la troisième par 8. Ainsi en partant d'une image de taille  $416 \times 416$  pixels, les trois features maps en sortie du réseau auront des tailles respectives de  $13 \times 13$ ,  $26 \times 26$  et  $52 \times 52$  pixels. C'est en

ce sens que YOLOv3 prédit trois niveaux de détails, pour détecter respectivement les gros, moyens et petits objets.

Partant d'une image de taille  $416 \times 416$  pixels, chaque entrée (pixel) est « conduit » à travers le réseau jusqu'à trois cellules (première sortie, 2ème, 3ème). Pour chaque cellule trois bounding box sont prédites, cela en fait un total de 9 qui sont issues des 9 "anchor box". Pour chaque bounding box, un score d'objectness et des scores d'appartenances aux classes sont prédits. Au total, le réseau propose  $52 \times 52 \times 3 + 26 \times 26 \times 3 + 13 \times 13 \times 3 = 10647$  bounding boxes.

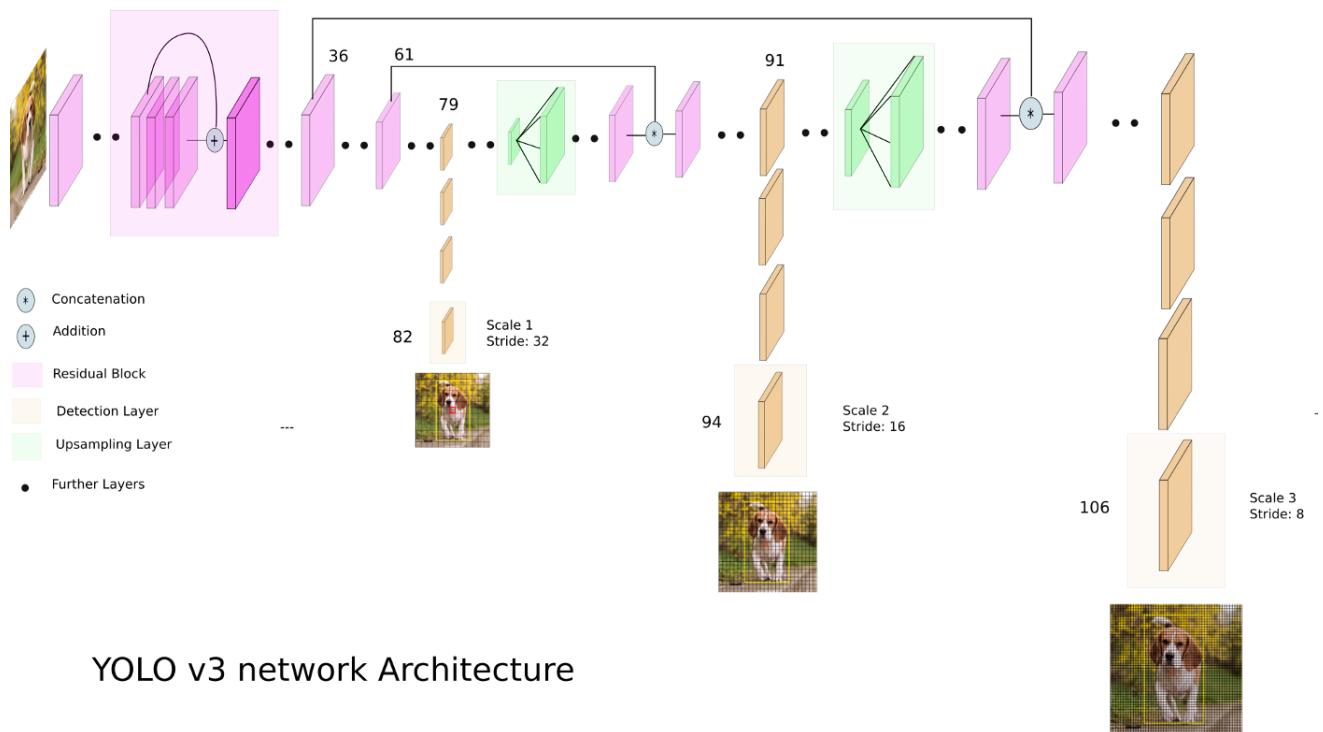


FIGURE 6 – Architecture Yolov3

image libre de droit src : <https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>  
L'IoU – ou Intersection over Union – est un indicateur de la qualité de la détection

d'un objet en comparant, dans un dataset d'entraînement, la position connue de l'objet dans l'image avec la prédiction faite par l'algorithme. L'IoU est le rapport entre la surface de l'intersection de 2 bounding box (celle prédite et la véritable) considérées et la surface de l'union des 2.

L'IoU peut valoir entre 0 (pour une détection totalement ratée) et 1 (pour une détection parfaite). De façon générale l'objet est considéré comme étant détecté à partir d'un IoU supérieur à un certain nombre (0.24 dans la littérature).

Nous utiliserons abondamment une métrique très similaire à l'IOU comme mesure pour notre validation lors de l'entraînement. La métrique mAP basé sur la moyenne des précisions. Pour cette première tâche nous optons pour la version3 de l'algorithme.

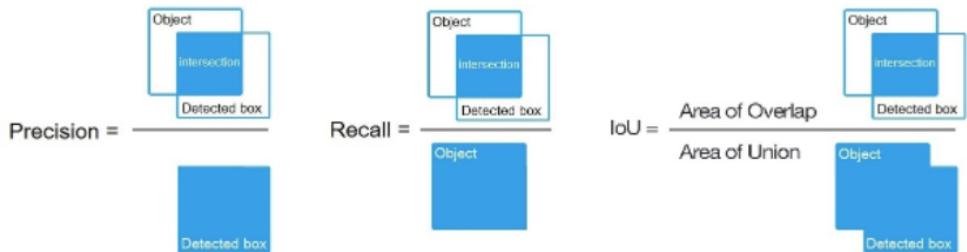


FIGURE 7 – Precision - Recall - IOU src :[1]

Plus de détail sur le papier de recherche de YOLO.[4]

**2.3.1.1 Premiers prétraitements** Nous avons choisi LabelImg pour annoter nos données. L'étape de l'annotation est une étape cruciale et a un impact immense sur la qualité des résultats de prédiction. Cette étape consiste à tracer des rectangles autour de l'objet que nous souhaitons détecter et d'annoter un nom de classe à chaque rectangle tracés sur l'image. Pour constituer notre jeu d'entraîne-

ment nous avons donc choisi dans un premier temps de tracer les box autour de la baleine entière sur environ 500 images. Nous verrons plus tard que c'était d'une mauvaise idée.

Le logiciel génère pour chaque image un fichier txt correspondant où chaque ligne correspond à un rectangle avec comme paramètre respectivement :

$\text{classe}_{id}, x_{center}, y_{center}, h_{box}, w_{box}$

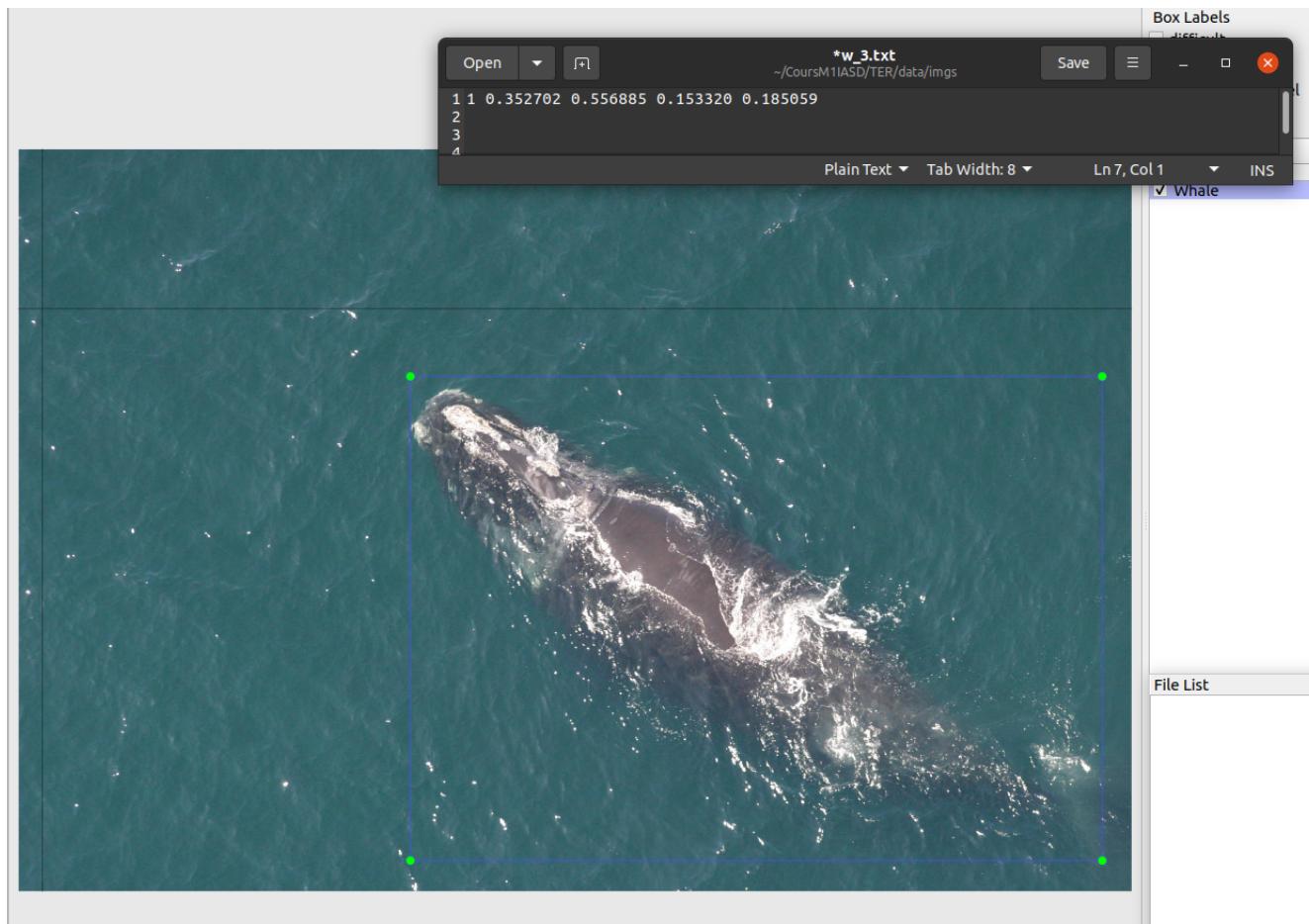


FIGURE 8 – LabelImg - Whole whale Box

**2.3.1.2 Développement du premier modèle** Pour mettre en place un entraînement Yolo il est nécessaire de faire quelques réglages avant. Pour toutes les

implémentations de l'algorithme Yolo nous avons utilisé le repo suivant<sup>3</sup>.

Ci-dessous sont décrites les étapes effectuées dans les grandes lignes même s'il y a eu beaucoup d'étapes intermédiaires :

1. **Cloner le repository**

2. **Faire quelques changement dans le Makefile** Dans les lignes suivantes nous précisons que nous utiliserons les fonctionnalités pour le GPU et OpenCV. CUDNN et GPU = 1 pour du calcul accéléré sur GPU.

```
sed -i 's/OPENCV=0/OPENCV=1/' Makefile  
sed -i 's/GPU=0/GPU=1/' Makefile  
sed -i 's/CUDNN=0/CUDNN=1/' Makefile
```

Puis compiler.

3. **Ajout des fichiers de configurations** Ajout des fichiers obj.names précisant le nom de la ou des classes. Et obj.names indiquant le chemin du fichier train.txt, le chemin d'accès au fichier d'accès au test.txt, le chemin d'accès au fichier.

```
mkdir training  
echo "Whale" > data/obj.names  
echo -e 'classes= 1\ntrain = data/train.txt\  
nvalid = data/test.txt\nnames = data/obj.  
names\nbackup = training/' > data/obj.data  
mkdir data/obj
```

4. **Décompresser les images** Décompresser les images et les mettre dans le répertoire data.

5. **Créer un jeu de données d'entraînement et de validation** à partir de toutes les images, créer un script bash (le script est disponible sur notre

---

3. <https://github.com/AlexeyAB/darknet>

github) qui va diviser le jeu de manière à avoir 10% de validation et le reste pour l'entraînement. Créer les fichier train.txt et test.txt avec la liste des chemins d'accès des images des jeux respectifs.

6. **Modifier le fichier yolov3-custom.cfg** Modifier le fichier du paramétrage du réseau : Régler la batch size (nombre d'images envoyées par paquet durant l'entraînement). Régler la subdivision assez haute mais pas supérieure à la batch size. Attention pour ce dernier, un nombre trop faible va inévitablement faire crasher le programme car il n'y aura plus assez de RAM disponible. L'entraînement de Yolo est très coûteux en ressources donc un bon conseil serait de garder le nombre de subdivisions supérieur à 32. Vient ensuite la modification du nombres de classes puis la modification de la taille des filtres à modifier sur chaque couches de sorties du réseau.

```
sed -i 's/batch=1/batch=64/' cfg/  
yolov3_training.cfg  
sed -i 's/subdivisions=1/subdivisions=16/' cfg/  
yolov3_training.cfg  
sed -i 's/max_batches = 500200/max_batches =  
4000/' cfg/yolov3_training.cfg  
sed -i '610 s@classes=80@classes=10' cfg/  
yolov3_training.cfg  
sed -i '696 s@classes=80@classes=10' cfg/  
yolov3_training.cfg  
sed -i '783 s@classes=80@classes=10' cfg/  
yolov3_training.cfg  
sed -i '603 s@filters=255@filters=180' cfg/  
yolov3_training.cfg  
sed -i '689 s@filters=255@filters=180' cfg/
```

```
yolov3_training.cfg  
sed -i '776 s@filters=255@filters=180' cfg/  
yolov3_training.cfg
```

## 7. Récuperer les poids

```
wget https://pjreddie.com/media/files/darknet53  
.conv.74
```

## 8. Lancer l'entraînement

```
./darknet detector train data/obj.data cfg/  
yolov3_training.cfg darknet53.conv.74 -  
dont_show -map
```

Plus de détails sur les réglages à effectuer dans le README du projet.

**2.3.1.3 Évaluation des résultats** Malheureusement pour ce premier réseau, la trace de la métrique mAP durant l'entraînement est absente pour une raison très simple : C'était pour nous la première implémentation de l'algorithme YOLO, nous n'avons donc pas correctement appliqué le tuto précédent et nous avons oublié le set de validation (grosse erreur) durant l'entraînement du modèle, donc impossible de garder trace des performances de notre algorithme durant l'entraînement. Pour l'évaluation de nos résultats nous avons donc simplement regardé les sorties données par l'algorithme. Les images croppées sont plutôt bonnes même si le nombre de 500 images semble un peu léger pour une détection optimale. ex de sortie après crop de l'image sur les box prédites.

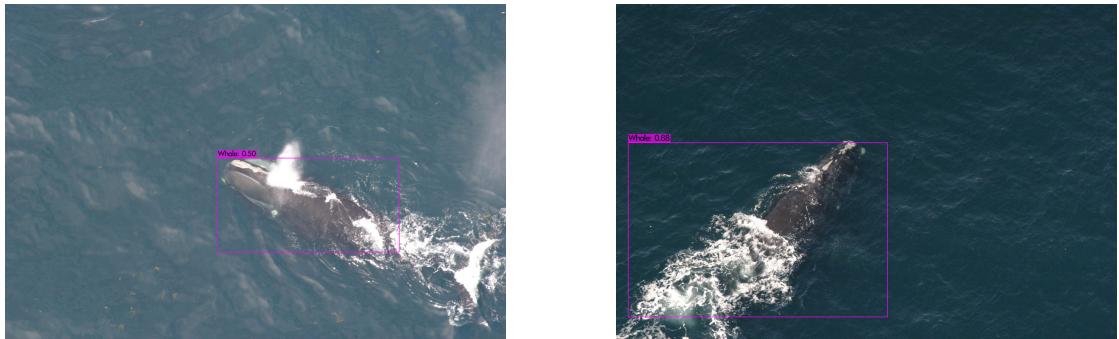


FIGURE 9 – exemple de sortie sur Yolov3

#### 2.3.1.4 Problèmes et seconds prétraitements

Après, quelques recherches sur Internet sur les baleines noires on s'aperçoit que la caractéristique la plus distinctive d'une baleine noire se situe au niveau de sa tête. Les plaques rugueuses de la peau sur sa tête qui apparaissent blanches en raison du parasitisme par les poux de baleine. Il s'avère que cela les distingue non seulement des autres espèces de baleines, mais constitue également un bon moyen de différencier les individus. La solution présentée précédemment ne se concentre pas assez sur la tête mais considère la baleine entière. On décide donc d'utiliser une autre configuration : On va répéter le processus précédent mais en se concentrant uniquement sur la zone autour de la tête et de la partie haute du dos. Nous constituons cette fois-ci un jeu de données d'entraînement de 3000 images.

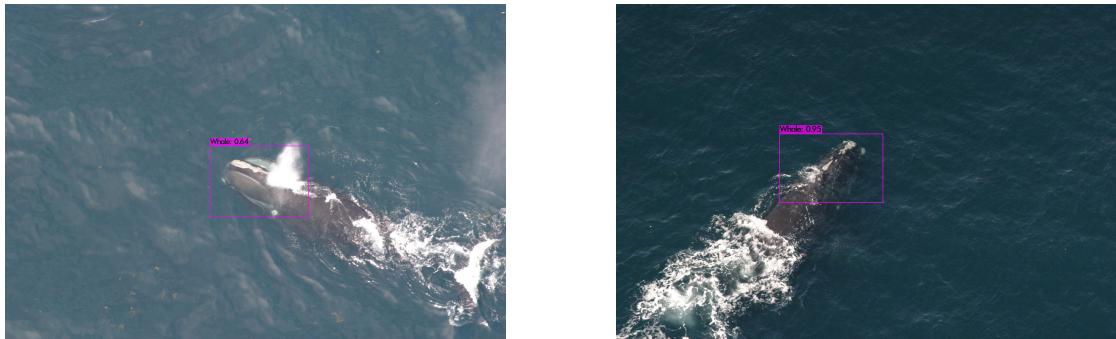


FIGURE 10 – exemple de sortie sur Yolov3 avec seulement la tête

Après entraînement en appliquant cette fois ci correctement toutes les étapes, les résultats sont très bons, ci-dessous le graphique de l’entraînement n’indique pas de signe d’overfitting y compris après les 28h d’entraînement. On remarque une très bonne précision d’environ 99% (d’indice mAP ce qui n’est pas rare avec YOLO pour les gros objets), du modèle ce que nous avons pu aussi constater sur les photos. Nous avons utilisé 10% du dataset pour la validation.

Une fois l’entraînement et la validation de nos résultats effectué, nous avons croppé chaque image au niveau de la “box” prédite par Yolo. Le script crop\_x12.py disponible sur github récupère les coordonnées des bounding box de chaque image, “croppé” l’image w\_i.jpg et enregistre l’image croppée wC\_i.jpg

Pour les quelques images où Yolo n’a pas détecté la partie avant de la baleine sur l’image (moins de 0.5% des images) nous avons tracé les rectangles à la main.

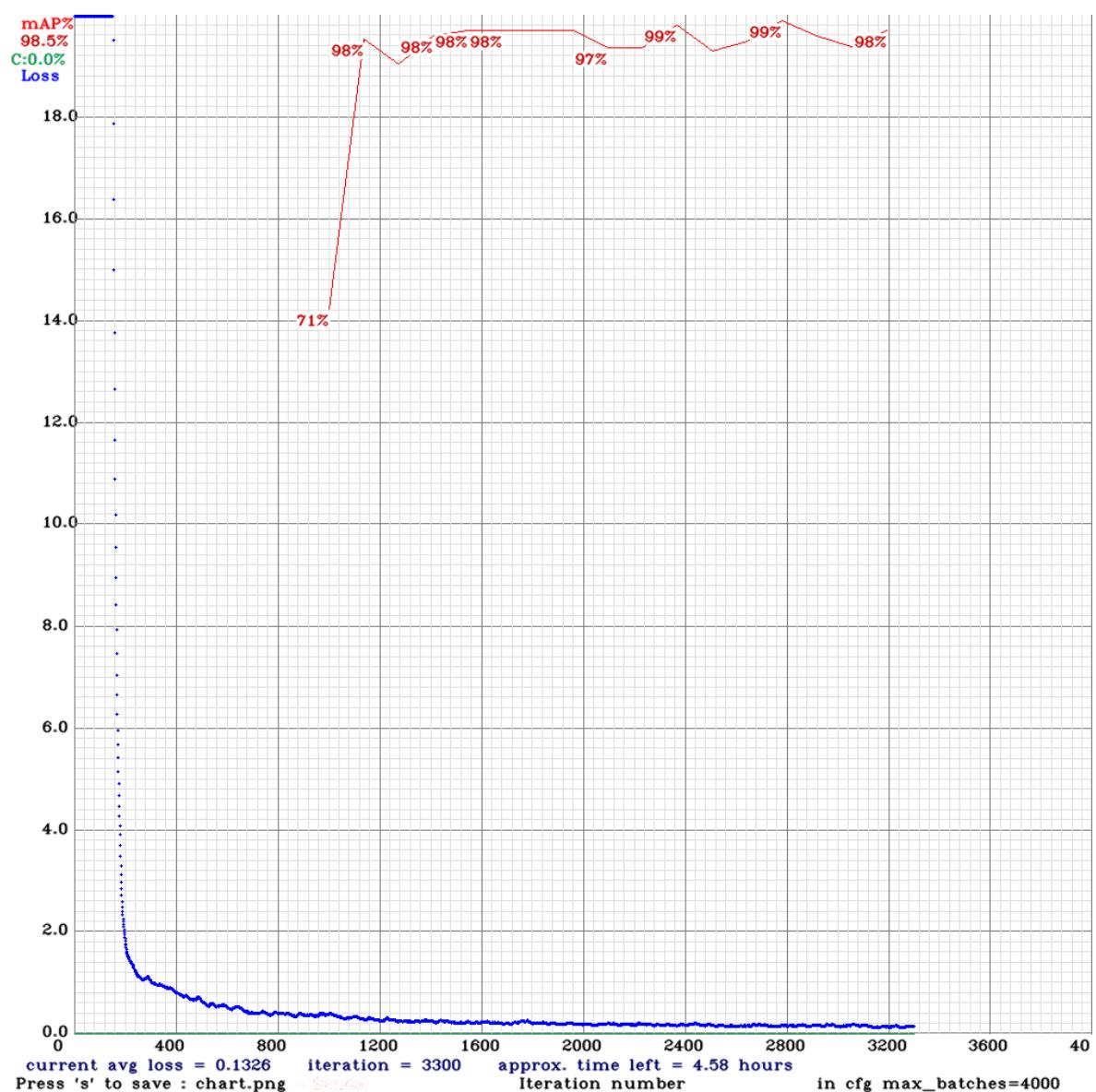


FIGURE 11 – Loss et mAP pendant l'entraînement de Yolov4

### **2.3.2 Head Aligner Yolo v4 et Algo d'Alignment**

Une fois la partie avant de la baleine détectée, la tâche suivante consiste maintenant à aligner nos images sur un même standard de manière à constituer des sortes de photos d'identité de baleines. Cette partie est absolument primordiale à l'obtention de bons résultats lors du déroulement du 3ème algorithme qui porte sur la détection de face.

Pour mettre en place cet alignement nous avons d'abord envisagé d'utiliser deux approches une première en essayant d'appliquer directement des opérations comme des homographies par exemple. Cette approche marche très bien dans certaines tâches de computer vision lorsqu'une sorte de template de référence sert de base pour les images à aligner. Or ici ce n'est pas du tout le cas, les baleines ne se ressemblent pas assez et les prises de vues comportent trop d'écart pour espérer qu'une approche comme celle-ci fonctionne.

Nous allons donc réutiliser l'algorithme Yolo mais cette fois-ci dans sa version 4 qui est plus performante dans la détection de petits objets. Nous utiliserons 2 classes : une classe Head(rectangle tracé autour de la tête) et une classe Air (rectangle tracé autour de l'évent).

#### **2.3.2.1 Premiers prétraitements**

Pour les annotations nous avons choisi de labéliser environ 2000 images avec 2 choix particuliers. Tout d'abord nous labélisions toujours nos images avec 2 classes pour ne pas avoir un jeu de données asymétriques. De cette manière nous avions 2000 “boundings box“ pour la classe head et 2000 pour la classe air.

De plus, nous avons décidé de tracer les bounding box uniquement lorsque les attributs que l'algorithme essaie de distinguer sont clairement visibles par un humain. En fait, on a appliqué une règle simple lorsque l'on constitue un jeu d'apprentissage : on trace des “box” uniquement autour des objets que l'on souhaite que

l'algorithme distingue. Si dans une situation (comme cela arrive souvent) l'évent est sous l'eau par exemple, nous sommes capables nous humain de le placer à peu près sur l'image mais l'algorithme va voir uniquement un carré d'eau et d'écumes. Prendre en considération cette image et tracer une "box" approximative est le meilleur moyen pour "pourrir" nos données d'apprentissage.

### 2.3.2.2 Développement du second modèle

Ajout de deux options pour accélérer encore plus le calcul GPU.

```
sed -i 's/OPENCV=0/OPENCV=1/' Makefile
sed -i 's/GPU=0/GPU=1/' Makefile
sed -i 's/CUDNN=0/CUDNN=1/' Makefile
sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

Nous avons augmenté encore le nombre de subdivisions car le programme s'est retrouvé de nombreuses fois à court de RAM ce qui peut être assez frustrant après plusieurs heures d'entraînement. Nous avons aussi augmenté le nombre d'itérations pour l'entraînement, fixé désormais à 6000, ce qui est en fait le nombre minimal recommandé.

```
sed -i 's/batch=1/batch=64/' cfg/yolov3_training.cfg
sed -i 's/subdivisions=1/subdivisions=32/' cfg/
    yolov3_training.cfg
sed -i 's/max_batches = 500200/max_batches = 6000/' cfg/
    yolov3_training.cfg
sed -i '610 s@classes=80@classes=2@' cfg/yolov3_training
    .cfg
```

```

sed -i '696 s@classes=80@classes=2@' cfg/yolov3_training
.cfg
sed -i '783 s@classes=80@classes=2@' cfg/yolov3_training
.cfg
sed -i '603 s@filters=255@filters=21@' cfg/
yolov3_training.cfg
sed -i '689 s@filters=255@filters=21@' cfg/
yolov3_training.cfg
sed -i '776 s@filters=255@filters=21@' cfg/
yolov3_training.cfg

```

Au final l’entraînement à été plus court d’environ de moitié que la deuxième implémentation de Yolov3 (pour détecter la partie avant du corps), ceci étant dû à la taille plus légère du dataset.

### 2.3.2.3 Résultats

Pour évaluer les performances de l’algo avec réglages d’un seuil on tape la commande suivante :

```
!./darknet detector test data/obj.data cfg/yolov4-
testing.cfg training/yolov4-custom_best.weights
path_image -thresh 0.24
```

On voit sur les images ci-dessous que la détection se déroule comme prévu sur des images de qualité correcte même sur des images vraiment moyennes (image du bas à gauche). Par contre lorsque l’évent est partiellement sous l’eau la détection ne se fait pas (image du bas à droite).

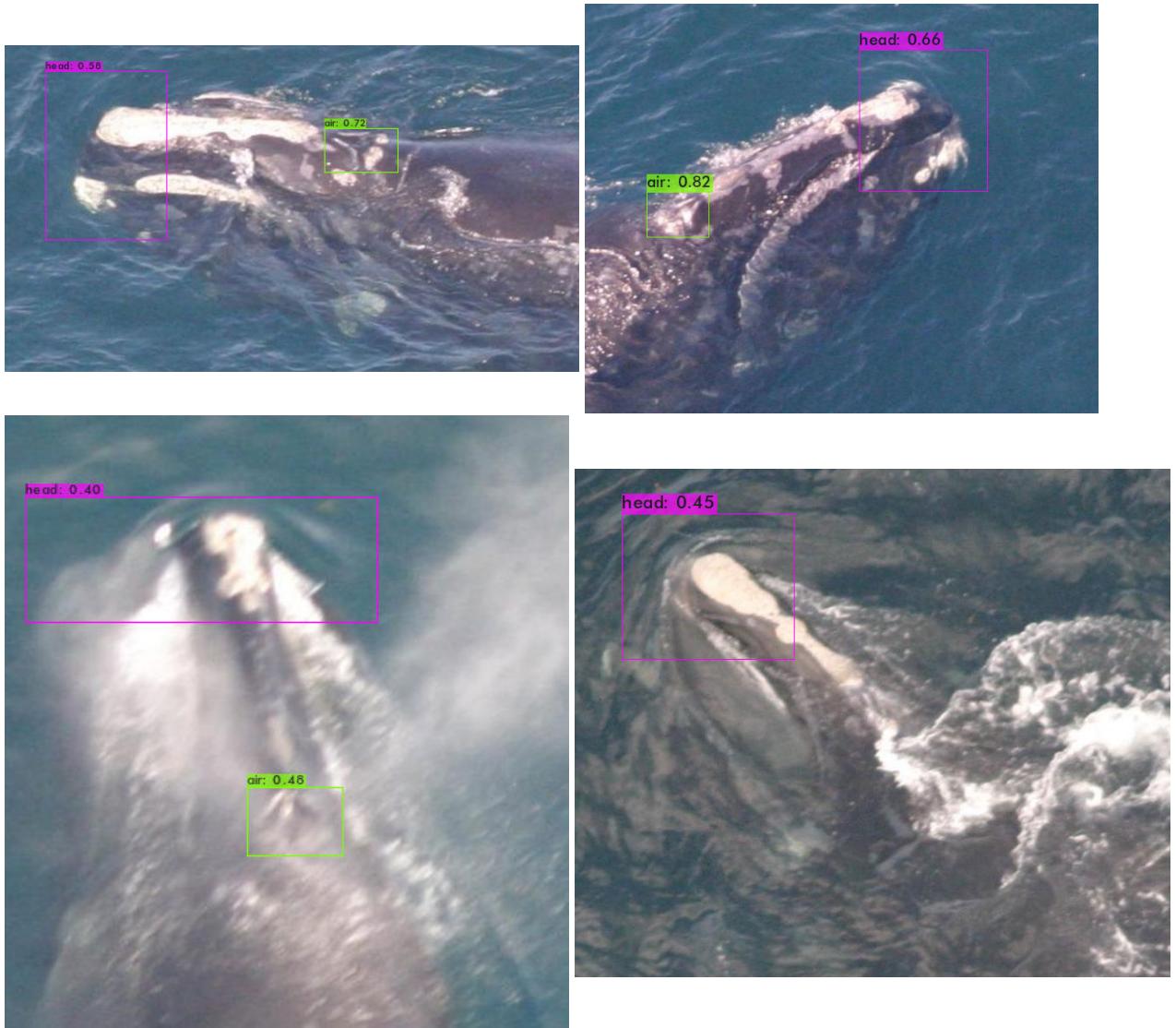


FIGURE 12 – Résultat Yolov4 sur les attributs

On observe une précision moyenne (mAP) beaucoup moins bonnes que précédemment. (cf graphique ci dessous)

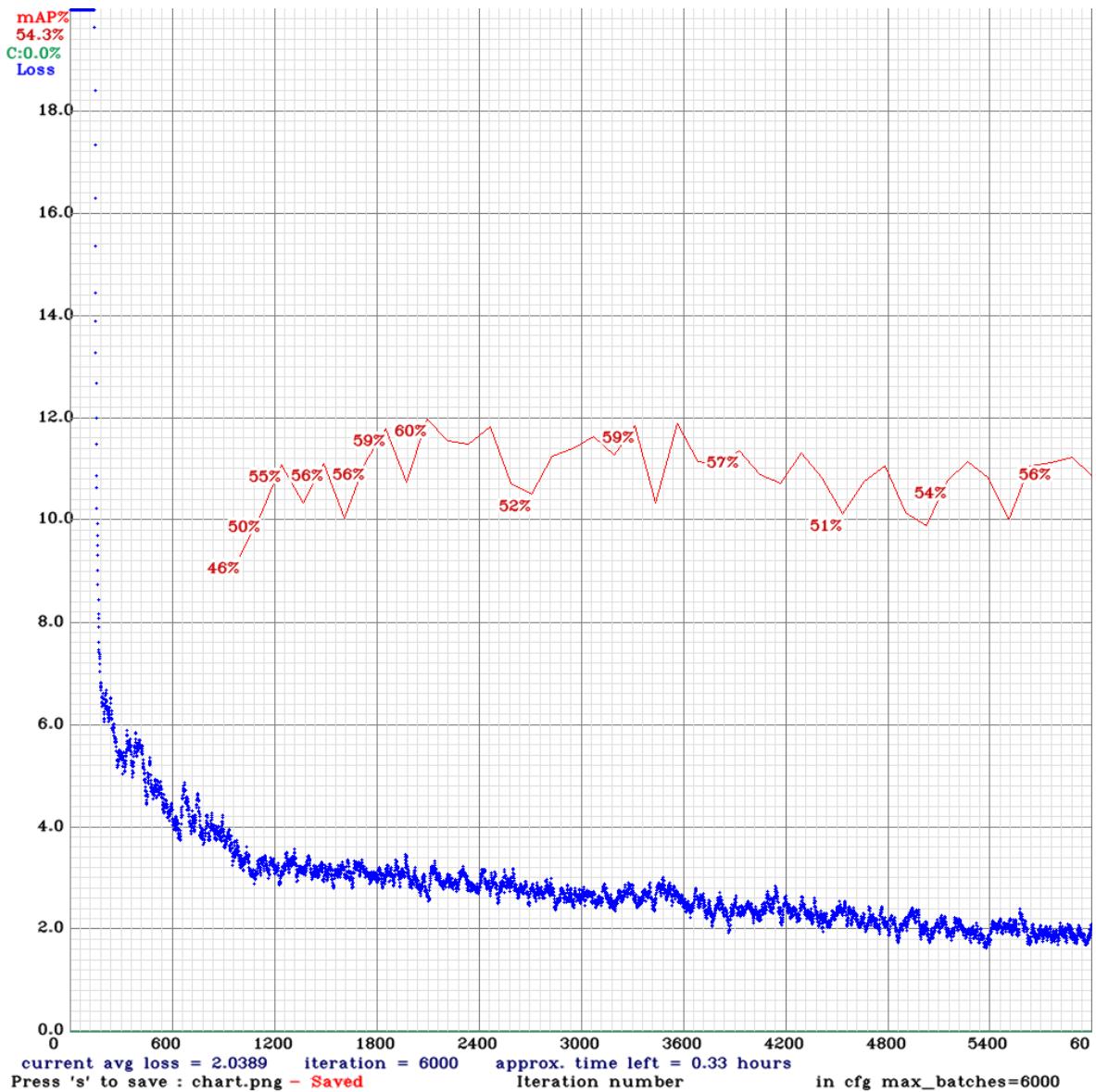


FIGURE 13 – Loss et mAP pendant l’entraînement de Yolov3

Cela peut s’expliquer en partie par deux choses. Premièrement dans environ 20% des images du jeu l’évent n’est pas visible ou très difficilement ce qui explique qu’il ne soit pas détecté dans ces cas. Deuxièmement les box head sont détectés mais avec des valeurs assez faibles ce qui doit s’expliquer en partie par notre annotation qui n’a pas été très correcte.

Notre algorithme semble correctement détecter les objets mais avec des chiffres de précision assez faibles. Nous fixerons donc un seuil à 0.2 pour essayer de garder au maximum les détections et ne pas avoir à aligner à la main trop d'images.

Ci-dessous la commande permet de sauvegarder toutes les coordonnées des bounding box prédites par Yolo dans un fichier json avec un threshold réduit au maximum( dans la limite de l'acceptable en matière de précision).

```
./darknet detector test data/obj.data yolov4-testing.cfg  
training/yolov4-custom_best.weights -ext_output -out  
train.json < imgs_path.txt -dont_show -thresh 0.2
```

Parmi les 114000 images, notre modèle à réussi à détecter à la fois la tête et l'évent sur environ 9000 images. Dans les cas restants la baleine sur l'image avait soit l'évent sous l'eau ou alors noyé dans l'écume ce qui nous a poussé à ne pas réentraîner le modèle sur les images restantes, car de toute façon nous aurions eu du mal à bien localiser la position de l'évent lors de la labelisation.

#### 2.3.2.4 Alignement des images

Une fois les coordonnées de l'évent et de la tête récupérées, nous avons écrit un script qui, via des formules mathématiques, (à coup de trigo  $\arccos$ , etc.) va trouver l'angle de rotation qu'il faut appliquer pour aligner l'image. Une fois la rotation effectuée on coupe l'image en conservant le même ratio hauteur, largeur de manière à supprimer toutes les parties noires sur l'image créée par la rotation.

Ci-dessous un échantillon non aléatoire de baleines alignées :

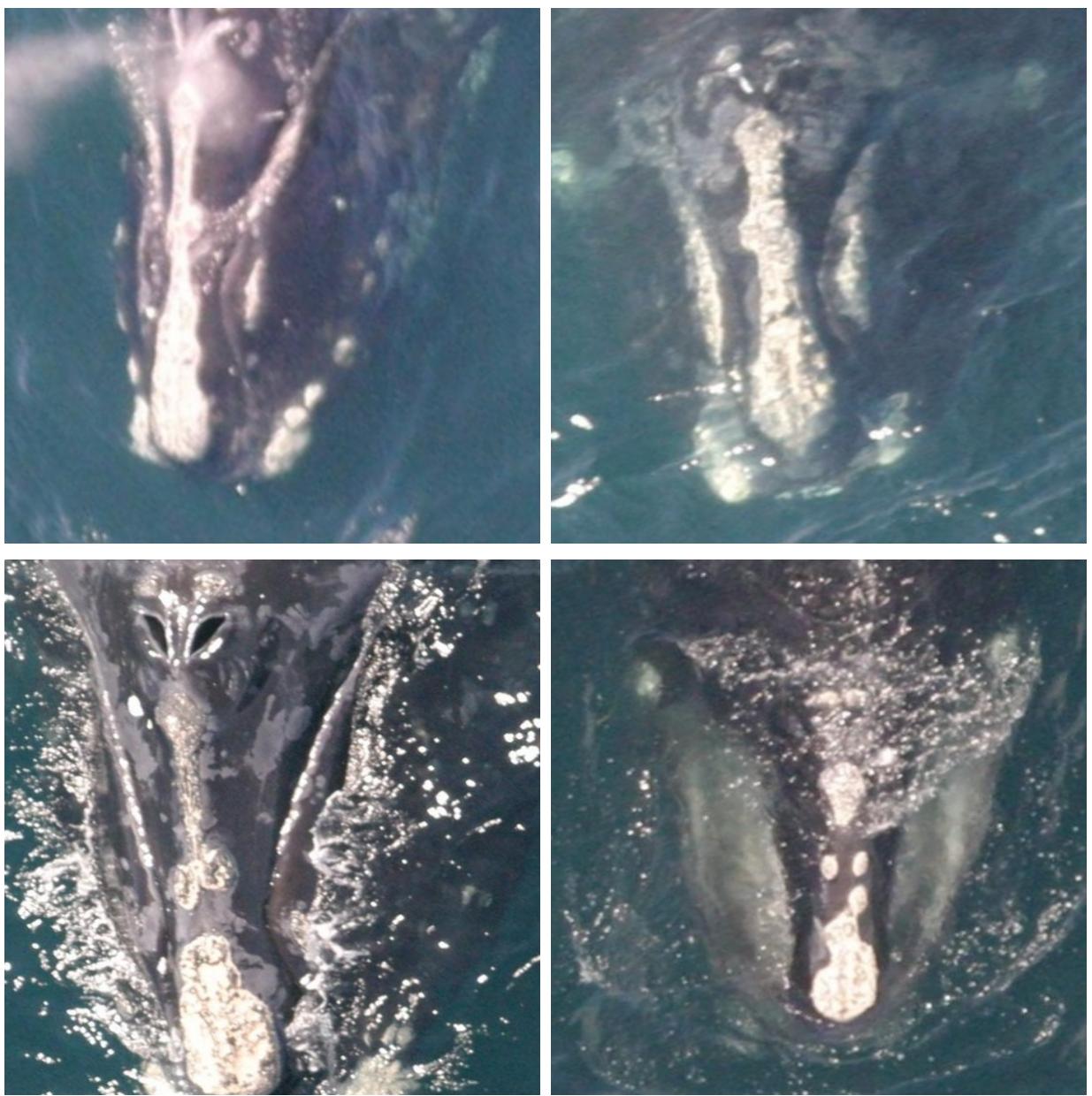


FIGURE 14 – Echantillon de baleines alignées

### 2.3.3 Système de reconnaissance faciale

La dernière partie à été de loin la plus complexe à mettre en place, il existe pléthore de ressources et d'implémentation du réseau Facenet mais pré-entraîné sur des bases d'images d'humains et c'est ce qui fait toute la différence...

En quelques mots, FaceNet, est un réseau qui apprend et génère un mappage de visage de haute qualité à partir des images en utilisant des architectures classiques de CNN. Il produit donc un espace euclidien "des vecteurs embeddings" pour chaque image, ou pour faire simple, la distance entre ces deux vecteurs représente le taux de similarité des 2 images correspondantes. Le réseau apprend donc les caractéristiques principales d'un visage, ce qui permet donc de les distinguer entre eux.

Une fois que cet espace de vecteurs embeddings est produit, des tâches comme la vérification<sup>4</sup> et le regroupement d'individus peuvent être facilement mises en œuvre. C'est d'ailleurs la première tâche que nous ferons pour évaluer notre modèle. La production du mappage est indispensable pour faire la face recognition<sup>5</sup> ce qui, je le rappelle, est le but final de notre projet.

Nous avions donc 2 possibilités : une assez simple et une plus complexe. La plus simple consistait à faire du transfert learning<sup>6</sup> c'est à dire partir du réseau FaceNet qui à été pré-entraîné et à appris un mappage de visage "humains" et espérer ensuite, en faisant du fine tuning<sup>7</sup> avec nos images de "faces de baleines" obtenir de bons résultats.

Cette solution je pense aurait très mal performé sur les baleines (ne trouvant

---

4. déterminer à partir d'une paire d'images si oui ou non il s'agit du même individus sur cette paire d'images

5. à parir d'une image déterminer l'individu sur l'image

6. Le transfer learning, désigne les techniques permettant de transférer les connaissances de plusieurs sources vers d'autres problèmes à régler

7. Le fine tuning permet des réglages spécifiques lors de la méthode du transfer learning, permettant de faire des ajustements progressifs et de geler certains poids du modèle pré-entraîné

pas les yeux, le nez, la bouche, . . .), une baleine ne comportant a priori que très peu de caractéristiques avec un “visage humain”, cette solution était sans doute vouée à l’échec.

Nous avons à la place décidé de réimplémenter le réseau sans pré-entraînement et de réutiliser les idées développées dans le papier de recherche de FaceNet<sup>8</sup>. Le réseau a été donc entraîné pour produire un mappage sur des images de faces de baleines, nous avons par contre réutilisé la même architecture que le réseau FaceNet.

L’architecture se compose d’un réseau convolutif suivi d’une normalisation et en sortie un vecteur-embedding<sup>9</sup> normalisé.

La nouveauté de ce réseau provient en partie de sa fonction de perte. Au lieu d’évaluer la perte directement en comparant la donnée en sortie et la donnée attendue en sortie (ce qui est fait traditionnellement). Ici on récupère un triplet d’images, l’image actuelle dite “ancre”, une image dite “positive” de la même classe et une image dite “négative” d’une autre classe. On cherche de cette manière à assurer la condition suivante.

(1)

$$\|f(x_i^a) - f(x_i^p)\| + \alpha < \|f(x_i^a) - f(x_i^n)\|$$

où  $x^a$  représente l’image anchor  $x^p$  représente l’image positive  $x^n$  représente l’image négative c’est-à-dire on veut qu’il y ait une marge  $\alpha$  entre la distance séparant notre image à l’image négative et la distance entre notre image et l’image positive. Ceci est explicité sur le schéma ci dessous :

---

8. <https://arxiv.org/pdf/1503.03832.pdf>

9. un vecteur qui représente les données d’entrée c’est une manière de “mathématiser” nos données.

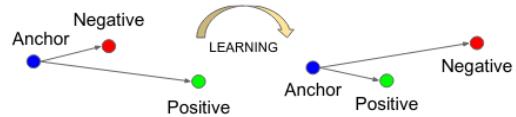


FIGURE 15 – Processus d’entraînement avec FaceNet

Ce type de calcul va permettre de "clusteriser" toutes les images qui appartiennent à la "même identité". On sera donc capables dans un premier temps de faire de la "face verification".

### 2.3.3.1 Premiers prétraitements et Data Augmentation

Il peut arriver que les données d'apprentissage manquent. C'est le cas du jeu de données que l'on dispose de Kaggle sur les baleines. Comme vu précédemment dans le rapport, il existe une grande hétérogénéité dans nos données. Par exemple, il peut y avoir une certaine baleine d'id n° $i$  n'admettant qu'une image, une autre en admettant une vingtaine et plus. Cela se fera clairement ressentir lors de la phase d'apprentissage de FaceNet. Pour pallier à ceci, nous avons recouru à une méthode commune en Machine Learning : l'augmentation des données - ou data augmentation. Celle-ci consiste en la génération de données "artificielles" par le biais de perturbations sur les données originelles. Cela nous permet d'augmenter la taille et la diversité de notre jeu d'entraînement. C'est une technique clé de régularisation, pour améliorer la performance de notre modèle et éviter l'overfitting. Dans le cas de nos images, ces "perturbations" évoquées précédemment seront en majorité des transformations géométriques mais également photométriques, c'est à dire sur l'espace de couleurs, RGB dans notre cas.

Nous avons choisi 4 techniques de data augmentation pour notre jeu d'entraînement, lesquelles seront détaillées par la suite :

- Flipping horizontal
- Rescaling
- Translation
- Channel shift

Nous avons isolé deux groupes de baleines préalablement, un groupe contenant entre 1 et 5 images par baleine, l'autre en contenant entre 6 et 10. C'est précisément sur ces baleines, donc sur ces deux groupes, que nous avons effectués de la data augmentation, les autres n'en ayant pas nécessairement besoin, ne souffrant pas

de ce manque de données. Le choix a été donc fait de faire entre 15 et 20 images augmentées supplémentaires par baleines pour le premier groupe, qui initialement en avait entre 1 et 5, et le double d'images augmentées pour chaque baleine du second groupe.

#### 2.3.3.2 Flipping :

Dû au fait que nos données soient alignées, nous n'avons utilisé que le flip vertical de sorte à n'avoir que l'effet miroir/symétrique sur l'image augmentée. Un flipping horizontal aurait changé la "carte d'identité" déjà standardisée, pour toutes nos baleines. Un exemple de cette transformation est donnée ci-dessous.

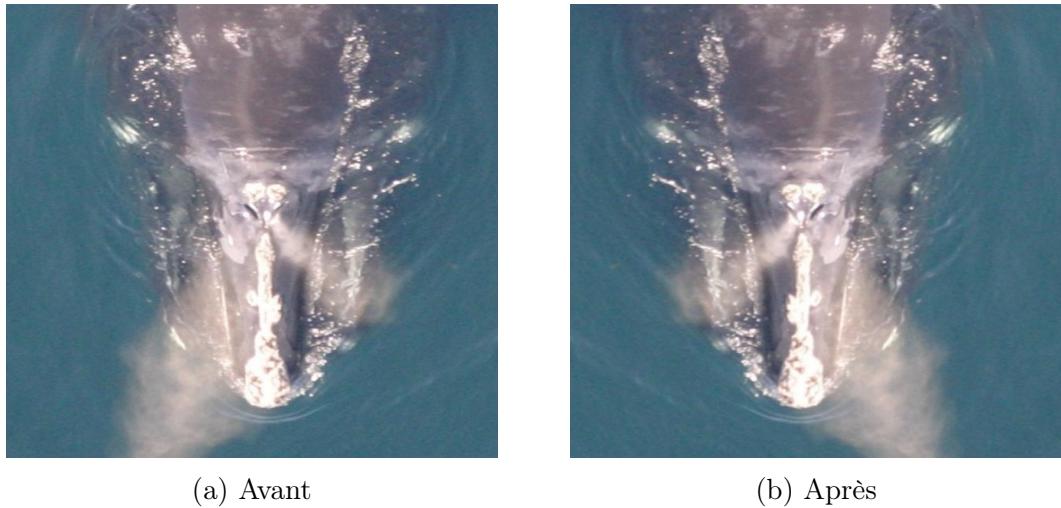


FIGURE 16 – Avant et après retournement [wCR\_3964.jpg]

#### 2.3.3.3 Rescaling :

Pour la mise à l'échelle, comme c'est une transformation d'image avec perte, nous devons choisir une méthode d'interpolation. L'interpolation, c'est, en traitement d'images, le calcul d'un pixel en fonction de ses pixels voisins. Nous avons choisi un facteur aléatoire entre 1 et 1.2 pour la mise à l'échelle de notre image, donc un élargissement, ou « sketch ». Étant donné ce facteur, nous devons donc “remplir”

les pixels manquants dans la “nouvelle matrice” que sera notre nouvelle image. Nous aurons donc besoin d’interpolation. C’est pourquoi avons choisi l’interpolation linéaire (ici fournie par OpenCV) : méthode plus lente que l’interpolation des plus proches voisins mais ne perd pas d’informations essentielles. L’interpolation cubique est quant à elle coûteuse en terme de calculs et donc plus lente, elle est de plus plus adaptée à la réduction d’images. [6]

#### **2.3.3.4 Translation :**

La translation a été faite dans l’intervalle  $[-4; 4]$  pixels de façon pseudo aléatoire. Donc des translations quasiment imperceptibles à l’oeil humain. Cependant, comme pour les rotations, nous faisons face à une aire noire inévitable, notre première solution qui n’en était pas une a été de croppé la zone concernée, mais l’image perdait alors en résolution. Nous avons donc finalement traité cette zone par interpolation faite "à la main", en isolant la zone originale, faisant la moyenne des pixels de la zone sur chaque canal RGB et en remplaçant harmonieusement cette dernière. La translation est définie par la matrice suivante :

$$M = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix}$$

Où  $tx$  est la translation sur l’axe des x, donc horizontale (gauche/droite, droite/gauche), et  $ty$  est la translation verticale (bas/haut, haut/bas).

#### **2.3.3.5 Channel shift :**

La perturbation des couleurs est une méthode couramment utilisée dans le domaine de la computer vision pour l’augmentation d’images. Notre méthode s’appuie sur l’approche traditionnelle consistant à ajouter une valeur de pixel sur l’espace de couleurs. Cette valeur dans notre cas est une valeur aléatoire. Nous avons premièrement utilisé l’approche avec les tableaux numpy, dont l’addition est basée sur

l'opération modulo, ce qui a posé problème, car étant basé sur cette opération, on se retrouve à avoir des pixels tourant au noir sur l'image, donc on a utilisé la fonction d'addition d'OpenCV qui est une opération arithmétique saturée et par conséquent évite cela. De plus, on sait que combiner une méthode avec d'autres algorithmes donnait de meilleurs résultats lors de la classification, le choix donc de combiner la perturbation de couleurs avec celles de transformations comme précédemment évoquées s'imposait.[5] Ci-dessous un exemple de distribution d'intensités sur les différents canaux pour une image aléatoire du jeu d'entraînement. On peut percevoir que l'image augmentée a un peu plus de contraste et plus saturée sur les tons rouges.

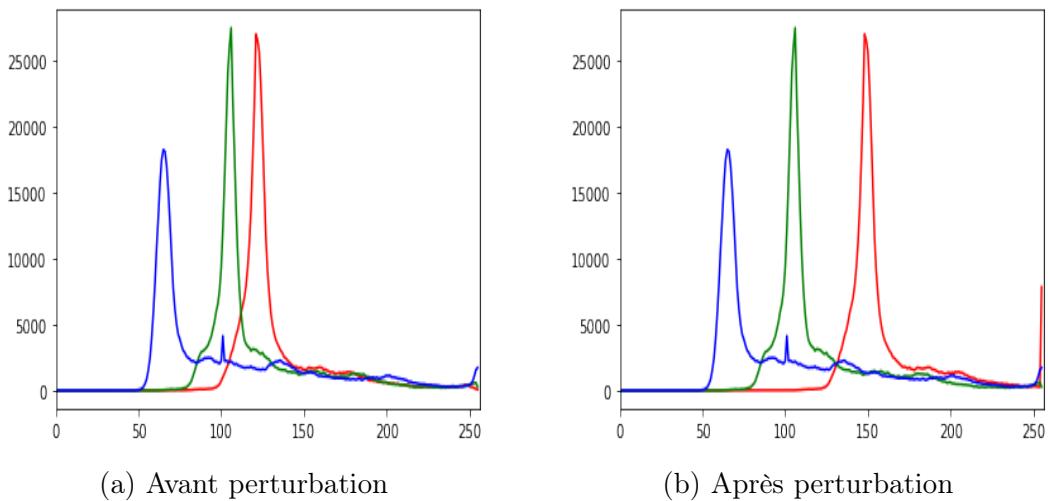


FIGURE 17 – Distribution des intensités d'une image de baleine [wCR\_3964.jpg]

#### 2.3.3.6 Regard sur notre Data Augmentation :

Par rétrospection, nous avons finalement fait un choix qui différait assez de celui présenté ci-dessus. Ce choix incluait uniquement des transformations géométriques, en l'occurrence le flipping, le rescaling et la translation. Nous avons donc exclu la perturbation des couleurs qui nous semblait compromettre assez les résultats

finaux. Nous avons également pris un seul groupe de baleines sur lequel créer des images augmentées : le groupe des baleines ayant entre 1 à 2 photos seulement, donc celles ayant le moins d'images sur tous le dataset. Ces baleines sont au nombre de 53 parmi les 447 individus répertoriés dans notre jeu de données. Bien loin donc des milliers d'images augmentées initialement pensées, mais nous reviendrons sur cela dans la suite de ce rapport.

### 2.3.4 Développement du second modèle

Malheureusement nous nous sommes heurtés à des difficultés conséquentes sur cette partie. Nous avons complètement sous-estimé le développement du 3ème modèle qui représentait un projet en lui même. Comme dit précédemment, un travail important sur l'implémentation du modèle quasiment "from scratch" a dû être réalisé. Le code développé inclut donc plusieurs parties distinctes :

Dans un premier temps nous chargeons le jeu de données préalablement "augmenté". On sépare ensuite le jeu de données en réservant 30% des classes (et non pas des images) pour la validation et le reste pour l'entraînement.

Dans un second temps on définit un ensemble de fonctions qui vont "pré-traiter" nos données : les normaliser, les resizer et c'est tout à priori. On définit ensuite des fonctions pour la génération des triplets. Pour ceux-ci, il existe deux modes de générations de triplets, un mode online et un mode offline développés dans le papier de recherche.

On définit ensuite notre fonction de perte qui a la même que dans le papier.

$$L = \sum_{n=1}^N [\|f(x_i^a) - f(x_i^p)\| - \|f(x_i^a) - f(x_i^n)\| + \alpha]$$

L'étape de la génération des triplets est cruciale si l'on veut assurer un apprentissage plus rapide et performant, nous devons prendre des triplets qui violent la condition (1) décrite précédemment. Il est cependant très coûteux de générer des triplets sur la base d'un ensemble d'entraînement complet.

Il existe donc deux modes de générations de triplets comme on a pu le voir précédemment et qui sont tous deux bien développés dans le papier de recherche. Nous avons choisi le second qui consiste à générer des triplets à chaque étape sur la base du dernier checkpoint du réseau et calculer le "hard negative" :  $\operatorname{argmin}_{x_i^n} (\|f(x_i^a) -$

$f(x_i^n)\|)$  et le "hard positive" :  $\text{argmax}_{x_i^p}(\|f(x_i^a) - f(x_i^p)\|)$  sur un sous-ensemble de données pris dans nbof\_subclasses que nous avons fixé à 40.

Pour limiter le nombre de calcul lors de la prediction, on ne va pas calculer nbof\_subclasses predictions pour la generation des hard triplets mais calculer un nombre directement impacté par la valeur de l'accuracy. Plus l'accuracy est élevée, plus on va chercher à générer de hard triplets. Ainsi on va en générer  $\text{int}(nbof\_subclasses * hard\_triplet\_ratio) + 2$ , où  $hard\_triplet\_ratio = \exp\left(\frac{-10*loss}{batch\_size}\right)$

Vient-ensuite la définition du modèle : Il suit l'architecture suivante avec pour taille d'embedding 32.

```

inputs = Input(shape=SIZE)
x = Conv2D(16, (7, 7), (2, 2), use_bias=False,
           activation='relu', padding='same')(inputs)
x = BatchNormalization()(x)
x = MaxPooling2D((3, 3))(x)

for layer in [16, 32, 64, 128, 512]:
    x = Conv2D(layer, (3, 3), strides=(2, 2), use_bias=
               False, activation='relu', padding='same')(x)
    r = BatchNormalization()(x)

    x = Conv2D(layer, (3, 3), use_bias=False, activation='
               relu', padding='same')(r)
    x = BatchNormalization()(x)
    r = Add()([r, x])

    x = Conv2D(layer, (3, 3), use_bias=False, activation='
               relu', padding='same')(x)
    x = BatchNormalization()(x)
    r = Add()([r, x])

```

```

    relu', padding='same')(r)
x = BatchNormalization()(x)
x = Add()([r,x])

x = GlobalAveragePooling2D()(x)
x = Flatten()
x = Dropout(0.5)(x)
x = Dense(emb_size, use_bias=False)(x)
outputs = Lambda(lambda x: tf.nn.l2_normalize(x, axis=-1))(x)

model = tf.keras.Model(inputs,outputs)

```

On lance ensuite l'entraînement où l'on sauvegarde à chaque epoch (car très souvent déconnecté des GPUs par Google colab), les poids du réseau, la val\_triplet\_accuracy et train\_triplet\_accuracy tout simplement en mesurant la proportion de triplet qui satisfait la condition (1). Nous faisons face à de nombreuses reprises avant de trouver les bons paramètres à de l'overfitting. D'où l'importance de regarder assez souvent ces deux métriques.

Mon abonnement à Google Colab Pro s'est terminé avant de pouvoir lancer l'entraînement des modèles, de plus nous étions aussi à court de temps pour bien terminer. Dans ce contexte nous voulions avoir tout de même quelques résultats à montrer, nous avons dû faire des concessions drastiques. Réduction de la taille des images en entrée, size fixée à (128,128,3). Taille du vecteur embedding réduite à 32 (normalement la valeur de 128 est préconisée). Un jeu de donnée pas "trop augmenté" pour ne pas avoir trop d'images.

#### 2.3.4.1 Résultats

Comme expliqué plus haut nous sommes arrivés à court de temps et de matériel pour terminer ce projet. Nous n'avons pas pu aller jusqu'à la partie détection d'individus qui était le but final du projet. Cependant comme vous avez pu le remarquer en lisant le rapport cette partie représentait seulement la partie émergée de l'iceberg. On peut voir sur la courbe suivante notre modèle qui est capable de satisfaire la condition (1) sur environ 80%, 85% des triplets du jeu de validation généré. On observe sur la courbe un léger sur-apprentissage sur la fin. Les meilleurs résultats s'observant autour de l'epoch 150-200.

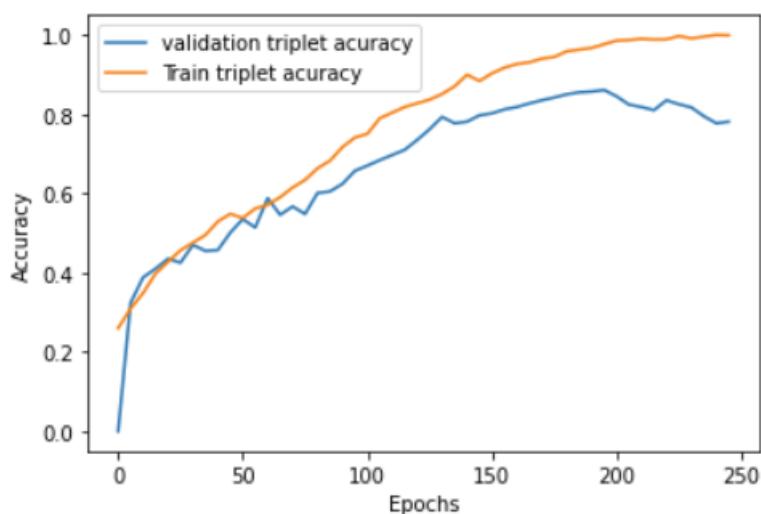


FIGURE 18 – Accuracy durant l'entraînement de FaceNet

On effectue ensuite dans notre évaluation une génération aléatoire de paires d'images de même classes et de paires de classes différentes en gardant trace si la paire est "same" (même classe) ou "not same" comme vérité terrain. On en génère le même nombre pour ces deux catégories. C'est-à-dire 2500 paires de classes différentes et 2500 paires d'images de même classes avec à chaque fois des images différentes dans chaque paire. On calcule ensuite les predictions (image transformé

en vecteur embedding) de chaque image et on compare tout simplement la distance séparant les 2 "vecteur embedding" produit en prédiction. Il faut ensuite déterminer une valeur de seuil sur la distance qui nous permettra de dire si oui ou non ces 2 vecteurs sont issus de la même classe. On détermine ensuite le meilleur seuil à fixer (celui qui donne la meilleure accuracy) à l'aide de notre vérité terrain. Une fois le seuil fixé on calcule une matrice de confusion sur toutes les paires générées.

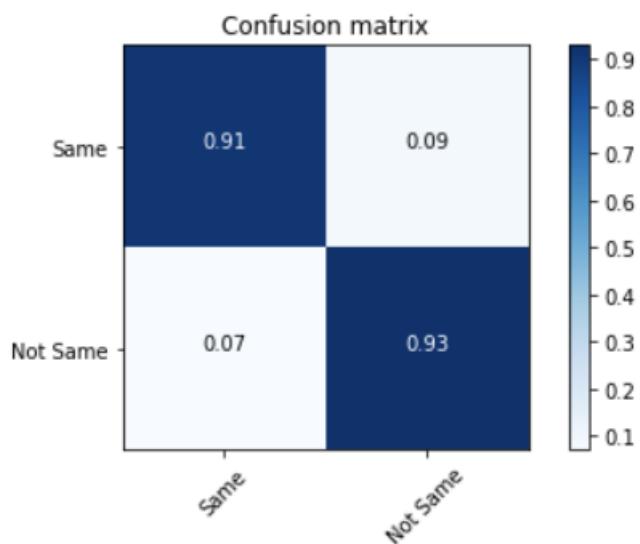


FIGURE 19 – Matrice de confusion

Il faut maintenant se servir de notre modèle pour prédire l'individu sur une image. N'ayant plus vraiment de temps nous avons pensé à une technique "simpliste" et gloutonne qui consiste à considérer toutes les images possibles du dataset et constituer des paires pour chacune avec l'image que nous essayons de reconnaître. On effectue ensuite un calcul de distance sur les embedding prédicts et l'on fait la moyenne pour chaque classe. La classe qui aura la moyenne la plus faible sera la classe de l'image. Cette méthode est très mauvaise en plus d'être extrêmement lente car elle oblige à considérer pour chaque image  $N-1$  paires où  $N$  est le

nombre d'images du jeu de données. Nous n'irons pas jusqu'au bout des calculs en se rendant vite compte qu'elle arrive à correctement prédire dans seulement un cas sur 10.

Il faut penser à une autre méthode d'évaluation... malheureusement le temps du projet est écoulé.

## 3 Conclusion

### 3.1 Perspectives d'amélioration

Améliorer les performances de Yolov4 en s'intéressant à tous les faux négatifs et essayer de gérer cela dans nos annotations leurs cas(Pourquoi yolo ne détecte pas la tête sur cette image ?). Terminer la dernière partie de FaceNet en réfléchissant à un modèle s'ajoutant à FaceNet permettant de faire de la face recognition. Pousser plus loin l'analyse de nos résultats notamment en observant à chaque fois les faux positifs faux négatifs et comprendre pourquoi ceux-ci n'ont pas été bien classé par l'algorithme. Soumettre nos résultats sur Kaggle. Faire de la recherche d'hyperparamètres sur nos algos avant de les lancer à nouveau. Améliorer la qualité de notre jeu de donnée, de notre data augmentation...

### 3.2 Ce qu'on a appris

Il y a à peine 5 mois nous ne connaissions absolument rien au deep learning hormis peut-être le fait qu'il résonnait en nous comme un buzz-word, et un domaine de l'IA qui semblait très attrayant. Nous avons donc grandement appris sur ce domaine en l'espace de ces 5 mois en passant de jeune novice ignorant à initié. Nous sommes désormais capable d'appliquer des modèles de deep sur des problèmes de computer vision plus ou moins classiques. Constituer nos propres jeux d'apprentissage, interpréter les résultats d'un modèle, lire et comprendre dans sa globalité des papiers scientifiques traitant le sujet, formaliser et expliquer assez clairement (je l'espère) les notions mathématiques et algorithmiques de ces modèles. On peut aussi se targuer d'en connaître un peu plus sur ces mammifères marins impressionants.

# Bibliographie

- [1] AlexeyAB. Yolov4 / scaled-yolov4 / yolo - neural networks for object detection (windows and linux version of darknet ).
- [2] GREMM (baleinesendirect.org). Baleines noires de l'atlantique nord.
- [3] Robert Bogucki, Marek Cygan, Christin Brangwynne Khan, Maciej Klimek, Jan Kantz Milczek, and Marcin Mucha. Applying deep learning to right whale photo identification. *Conservation Biology*, 33(3) :676–684, 2019.
- [4] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once : Unified, real-time object detection. 2016.
- [5] Eun Kyeong Kim, Hansoo Lee, Jin Yong Kim, and Sungshin Kim. Data augmentation method by applying color perturbation of inverse psnr and geometric transformations for object recognition based on deep learning. *Applied Sciences*, 10(11), 2020.
- [6] Pankaj Parsania and Dr Virparia. A comparative analysis of image interpolation algorithms. *IJARCCE*, 5 :29–34, 01 2016.