



UNIVERSITÉ DE  
**SHERBROOKE**

Élèves du deuxième cycle

Projet IFT-870

---

## **Identification de dauphins et de baleines à l'aide de réseaux convolutifs**

---

Firas ABOUDA *abof3001*

Gaétan REY *reyg3101*

Julien LEVARLET *levj2718*

Timothée WRIGHT *writ2801*

*Encadrant :*  
Mme. Nadia TAHIRI

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Rappel de la problématique . . . . .	3
1.2	Rappel du Contexte . . . . .	3
1.3	Conclusion de notre rapport précédent . . . . .	4
1.4	Ressources liées au projet . . . . .	4
<b>2</b>	<b>Pré-traitements des données</b>	<b>5</b>
2.1	Algorithme de localisation pour le rognage des images . . . . .	5
2.1.1	Notre implémentation . . . . .	5
2.1.2	Alternative . . . . .	6
2.2	Création de plusieurs jeux de données . . . . .	7
<b>3</b>	<b>Augmentation de données</b>	<b>8</b>
3.1	Résultats des transformations sur nos images . . . . .	9
3.2	Intégration dans le code . . . . .	10
<b>4</b>	<b>Entraînement</b>	<b>11</b>
4.1	Resnet . . . . .	11
4.1.1	Architecture . . . . .	11
4.1.2	Résultats . . . . .	12
4.2	EffNet . . . . .	13
4.2.1	Pourquoi utiliser EffNet . . . . .	13
4.2.2	Objectif du EffNet et scaling . . . . .	13
4.2.3	Architecture du EffNet . . . . .	14
4.2.4	Transfer Learning . . . . .	16
4.2.5	Résultats . . . . .	16
4.3	ArcFace . . . . .	17
4.3.1	Utilisation des embeddings . . . . .	17
4.3.2	Qu'est-ce qu'ArcFace . . . . .	17
4.3.3	Notre implémentation et nos résultats . . . . .	18

<b>5</b>	<b>Recherche d'hyperparamètre</b>	<b>19</b>
<b>6</b>	<b>Soumission des résultats sur Kaggle</b>	<b>20</b>
6.1	Mise en Place . . . . .	20
6.2	Résultats . . . . .	20
<b>7</b>	<b>Conclusion</b>	<b>22</b>

# 1 Introduction

## 1.1 Rappel de la problématique

Nous utilisons les empreintes digitales et la reconnaissance faciale pour identifier les personnes, mais pouvons-nous utiliser des approches similaires avec les mammifères marins ?

## 1.2 Rappel du Contexte

Les chercheurs suivent manuellement la vie marine par la forme et les marques sur leurs queues, leurs nageoires dorsales, leurs têtes et autres parties du corps des mammifères marins. L'identification par des marques naturelles via des photographies, appelée photo-identification, est un outil puissant pour la science des mammifères marins. Il permet de suivre les animaux individuels au fil du temps et il permet d'évaluer l'état et les tendances de la population.

L'objectif de ce projet est d'automatiser la photo-identification des baleines et des dauphins, les chercheurs pensent pouvoir réduire les temps d'identification des images de dauphins et de baleines de plus de 99% grâce au machine learning.

Actuellement, la plupart des instituts de recherche s'appuient sur une mise en correspondance manuelle, chronophage et parfois imprécise par l'œil humain. Des milliers d'heures sont consacrées à l'appariement manuel, qui consiste à regarder des photos pour comparer un individu à un autre, trouver des correspondances et identifier de nouveaux individus.

Les algorithmes développés dans le cadre de ce concours seront mis en œuvre dans Happywhale, une plateforme Web de collaboration, de recherche et de science citoyenne. Sa mission est d'accroître la compréhension globale et de prendre soin des environnements marins grâce à une science et une éducation de haute qualité en matière de conservation. Happywhale vise à rendre facile et enrichissante pour le public de participer à la science en créant des outils innovants pour impliquer toute personne intéressée par les mammifères marins. La plateforme est également au service de la communauté de la recherche avec de puissants outils collaboratifs.

Dans cette compétition, nous devons développer un modèle pour faire correspondre les baleines et les dauphins par des caractéristiques uniques, mais souvent subtiles, de leurs marques naturelles. Nous devons porter une attention particulière aux nageoires dorsales et aux vues latérales du corps dans des ensembles d'images provenant d'un ensemble de données multi-espèces construit par 28 instituts de recherche. L'objectif est de créer un modèle à la fois rapide et précis.

## 1.3 Conclusion de notre rapport précédent

Dans le cadre de ce projet, nous prévoyons dans un premier temps d'entraîner nos algorithmes sur un sous-ensemble de la base de données. Pour cela, nous sélectionnerons les individus dont nous avons un certain nombre de photos, par exemple minimum 100 pour commencer.

Nous utiliserons également des images redimensionnées. Pour cela, nous rognerons probablement l'image, pour ne garder que la partie importante via des bounding box ou des masques de segmentation.

Ces images rognées seront redimensionnées dans des tailles qui varieront en fonction de la performance de notre méthode.

Nous pourrons ensuite augmenter nos données.

Une fois la base données nettoyée et augmentée obtenue, nous entraînerons nos algorithmes de classification dessus.

Il est probable que nous nous contenterons d'architectures simples comme le ResNet ou le DenseNet, même si nous en essayerons plusieurs. Concernant la fonction de coût, nous utiliserons peut-être un softmax dans un premier temps pour une question de simplicité, puis nous essaierons d'utiliser ArcFace.

Une fois que cela fonctionnera, on pourra essayer d'améliorer notre modèle en optimisant des hyperparamètres, tels que l'optimiseur de notre réseau par exemple.

Nous prévoyons d'utiliser les API Pytorch et OpenCV pour nous permettre d'implémenter nos méthodes.

Au vu de la taille de la base de données, nos machines personnelles ne sont pas suffisantes pour faire toutes les étapes précédemment énumérées, nous ferons donc des tests en local sur des échantillons, pour valider que notre code fonctionne. Dans un second temps, nous utiliserons l'exécution cloud de Kaggle, pour un entraînement plus conséquent sur plus de données.

## 1.4 Ressources liées au projet

Notre projet complet est disponible sur GitHub à l'adresse suivante : <https://github.com/julien-levarlet/HappyWhale>.

Il est possible d'y retrouver notre code, les différentes ressources sur lesquels se base notre implémentation, ainsi que notre premier rapport.

L'état de l'art est détaillé dans notre rapport précédent. Dans ce rapport, nous reprendrons notre conclusion et développerons ce que nous avons mis en place.

## 2 Pré-traitements des données

### 2.1 Algorithme de localisation pour le rognage des images

Nous avons récupéré une architecture Yolo de détection pré-entraîné pour reconnaître des animaux et des objets de tout genre sur des images, disponible ici : <https://github.com/ultralytics/yolov5>.

#### 2.1.1 Notre implémentation

Nous avons créé une classe Python nous permettant de rogner nos images en utilisant les prédictions de Yolo. Il suffit de passer en argument le dossier qui contient les images et celui qui les contient. Il y a cependant des limites :

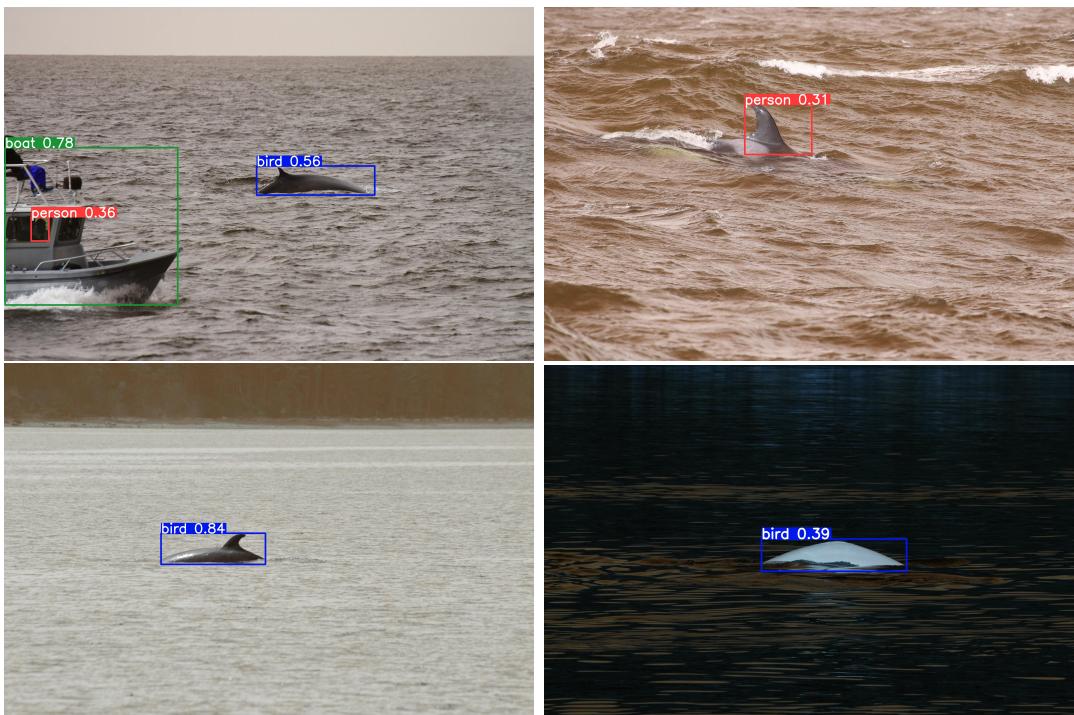


FIGURE 2.1 – Quelques images des prédictions que nous propose Yolo5

Vous l'aurez sans doute remarqué, mais Yolo ne reconnaît pas un dauphin comme "dauphin" mais plutôt comme un "oiseau" ou une "personne". En effet, Yolo n'a pas été entraîné à reconnaître des animaux marins. De manière générale, si le dauphin est seul sur l'image, il suffit de rogner sur la prédiction. Ce qui est plutôt simple, car on sait que c'est un dauphin qui a été trouvé. Cependant, quand il y a plusieurs options, le risque de se tromper est élevé.

Nous avons creusé et analysé les prédictions de Yolo sur nos données pour chercher à comprendre ce que reconnaît Yolo à la place d'un dauphin. L'idée étant de chercher à rogner les images avec plusieurs prédictions sur notre image.

```
yolo croped sumary:  
number of crop maked by single prediction: 17  
number of image ignore because of no prediction: 22  
number of image ignore because of ambigous prediction: 1  
dolphin can be predict as:  
prediction ponderation confidence  
0 bear 5 2.2886180877685547  
1 bird 5 1.7879080474376678  
2 surfboard 4 1.330518513917923  
3 cake 1 0.2604745626449585  
4 elephant 1 0.5278445482254028  
5 person 1 0.5348538756370544  
ambigous prediction are:  
[['elephant', 'bird']]
```

FIGURE 2.2 – Analyses des prédictions effectuées par Yolo sur 40 images

Comme on peut le voir, on ne peut pas distinguer un dauphin d'un autre animal ou d'une planche de surf. Donc on ne peut pas rogner l'image s'il y a plusieurs prédictions. Dans ce cas, on doit garder l'image originale.

Si ce n'est pas très fréquent, cela reste un gros problème, car il est très difficile d'apprendre sur les images qui ne ciblent pas parfaitement les ailerons. Les images ignorées lorsque que Yolo ne détecte rien sont des images déjà bien centrées sur le dauphin en général, donc ce n'est pas un problème.

Cependant, au-delà de 50 images, Yolo sature l'espace de notre carte graphique, donc il est impossible de s'en servir sur l'ensemble de nos 50000 images. Nous avons essayé de procéder par batch de 40 en vidant le cache de la carte graphique entre deux itérations sans succès.

### 2.1.2 Alternative

Suite aux problèmes précédent, nous avons cherché des alternatives à Yolo et la plus utilisée pour ce challenge est Detic<sup>2</sup>, qui est capable de reconnaître des dauphins entre autre. Cependant, l'installation du module python associé n'est disponible que pour Linux et MacOS. La majorité de nos machines étant sous Windows, nous n'avons pas pu utiliser ce module.

Malgré cela, nous avons pu exploiter un des points forts de Kaggle qui est la coopération entre les utilisateurs et récupérer une base de données rognée<sup>1</sup>, mise en place par l'utilisateur phalanx.

---

1. La base de donnée rognée que nous avons utilisée est la suivante <https://www.kaggle.com/competitions/happy-whale-and-dolphin/discussion/305503>

2. Le lien du dépôt GitHub de Detic est disponible ici : <https://github.com/facebookresearch/Detic>

## 2.2 Creation de plusieurs jeux de donnees

Apres avoir recupere la base de donnees rogne dont il est question ci-dessus, avons genere 3 jeux de donnees :

1. Un jeu de 750 images avec 5 classes (150 images par classes) pour tester l'apprentissage de nos models.
2. Un jeu de 1000 images avec 20 classes (50 images par classes) pour corser un peu la difficulte.
3. Le jeu complet avec toutes les images pour pouvoir preparer une soumission Kaggle.

Nous avons ensuite subdivise chacun des jeux de donnees en un ensemble de test contenant 20% des donnees, un ensemble de validation contenant 16% des donnees et notre ensemble d'entranement avec 64% des donnees.

Notre ensemble de validation sera utilise pour faire une recherche d'hyperparametres et contrôler l'apparition du sur-apprentissage au cours de l'entranement.

Pour effectuer la division, on conserve les proportions des classes dans le cas de nos deux premiers jeux de donnees (5 et 20 classes), car cela nous permet d'tre sur que nos ensembles d'entranement et de tests ont tous deux un ´equilibrage des classes similaire.

Dans le cas de la base de donnees finale, on proc`ede de faon alatoire, car certains individus n'tant representes qu'une fois, il n'est pas possible de faire un partage ´equitable entre entranement, validation et tests.

### 3 Augmentation de données

Nous avons fait une classe Python `Transformation` dans le fichier `transformation.py` qui permet de gérer l'augmentation des données, cette classe permet d'effectuer les traitements d'image suivants :

1. Une rotation sur l'image, nous appliquons une rotation, en s'assurant qu'aucun élément ne sorte de l'image. L'image d'origine est entièrement conservée, c'est l'image d'arrivée qui s'agrandit au besoin, puis avec un redimensionnement, celle-ci revient à la bonne taille.
2. Une translation de l'image selon l'axe des abscisses ou des ordonnées, il faut que soit légère pour ne pas perdre de l'information.
3. Un cisaillement de l'image, c'est une légère déformation de l'image qui a pour conséquence d'étirer l'image selon une diagonale.

Ces transformations<sup>1</sup> permettent d'étoffer la base de données dans l'esprit où certains ailerons de dauphins pourraient avoir des angles différents. De même, si la partie qui nous intéresse n'est pas en plein centre de l'image, une translation permet d'apprendre dans ces conditions. Et un cisaillement peut imiter un photographe mal positionnée.

De même, nous avons implémenté, d'autres transformations diverses comme :

1. Ajout de bruit, de type "pepper", "gaussian", "poisson", "s&p", "speckle".
2. Modification du contraste et de la luminosité des images.
3. Modification de la saturation des couleurs.
4. Changement de l'accentuation des couleurs des images.

Ces transformations ont pour objectif d'imiter les différences de luminosité, contrastes et saturation des images, pour mieux s'entraîner et s'adapter à ses différences.

Ces transformations sont toutes utilisables indépendamment les unes des autres en renseignant les paramètres de transformations. Cependant, il est aussi possible de renseigner une liste des fonctions à utiliser et des plages de transformations (comme faire une rotation entre -5 et 5 degrés) pour utiliser la fonction de génération aléatoire. Cette fonction permet de générer à partir d'une image autant d'images que l'on veut, de manière aléatoire, avec les transformations demandées et les plages d'ajustement renseignées. Cette fonctionnalité est très pratique pour l'augmentation des données, c'est celle que nous utilisons pour l'apprentissage. Pour plus de précision sur le fonctionnement de cette classe, le mode d'emploi de chaque paramètre et chaque fonction est détaillé directement dans la classe Python.

---

1. Les transformations utilisées sont inspirées de l'article suivant : <https://www.imaios.com/en/Company/blog/AI-for-medical-imaging-data-augmentation>

### 3.1 Résultats des transformations sur nos images



FIGURE 3.1 – La première image est celle d'origine, les autres sont générées par notre modèle d'augmentation de données

Comme on peut le voir, les modifications sont assez légères (c'est un choix que nous avons fait au niveau des paramètres aléatoires pour rester dans le thème de la base de données), mais

elles apportent une grande diversité d'images différentes, à la fois géométriquement et dans le contraste, la luminosité et nous permettrons de limiter au moins en partie le sur-apprentissage.

## 3.2 Intégration dans le code

Nous utilisons notre augmentation de données selon une probabilité donnée avant de fournir nos images au réseau, cette probabilité est renseignée lors de l'appel à la classe. Cela nous permet de fournir des images différentes d'une epoch d'entraînement sur l'autre.

Nous n'avons pas pu vérifier l'impact de l'augmentation de données lors de nos tests pour des soucis d'apprentissage, mais c'est une pratique fortement conseillée en machine learning, et d'autant plus dans le cas des réseaux de neurone.

Finalement, voici toutes les caractéristiques de l'augmentation de données utilisée pour notre soumission :

- probabilité d'utilisation : 0.8
- plage de rotation : [-5,5] degrés
- plage de translation : [-20,20] pixels
- plage de cisaillement : [-0.1,0.1]
- plage de contraste : [0.8,1.3]
- plage de luminosité : [0.9,1.2]

Ces valeurs ont été déterminées en faisant des tests, puis en visuellement estimant la qualité de la transformation.

# 4 Entraînement

## 4.1 Resnet

### 4.1.1 Architecture

Les gagnants d'un challenge Kaggle de 2015 avaient utilisé un ResNet pour obtenir leurs résultats<sup>1</sup>

Nous avons suivi leur méthode et implémenté les ResNet à 50,101 et 152 couches. Pour cela on part du réseau présenté ci-dessous.

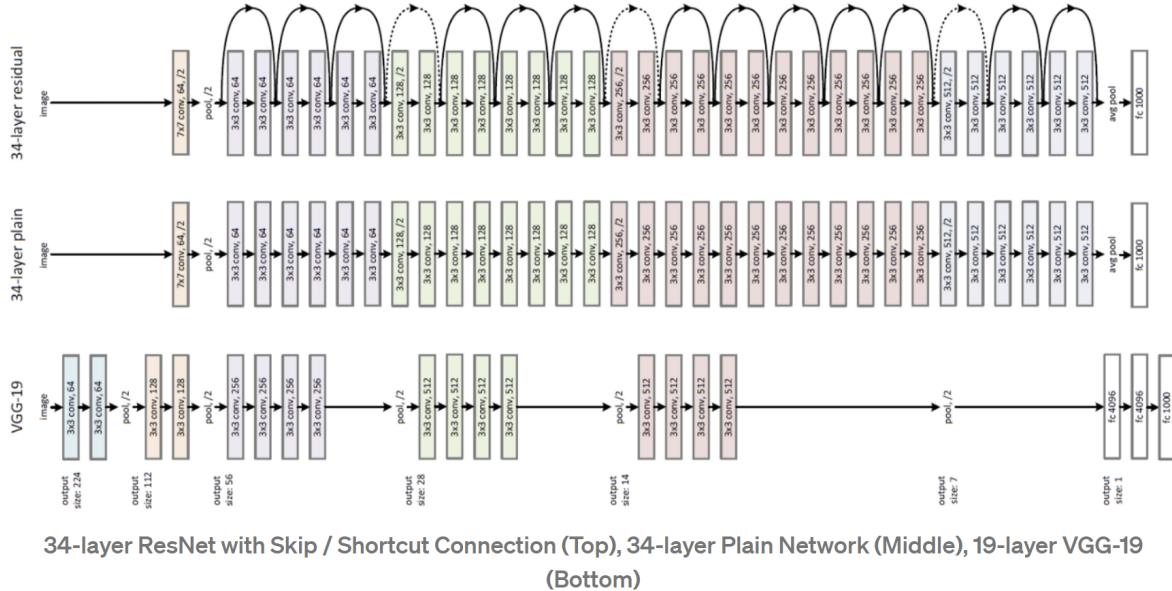


FIGURE 4.1 – Nous avons utilisé, en première architecture, celle du réseau résiduel à 34 couches

Ensuite on remplace les deux couches de convolutions entre chaque addition représentée par des flèches par des bottleneck comme ceci :

1. Méthode des vainqueurs d'un challenge de 2015 : <https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8>

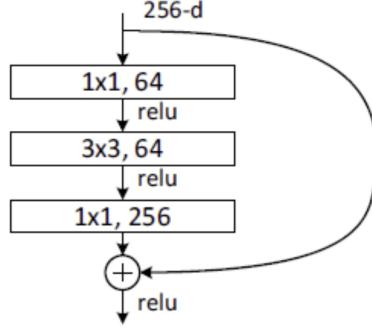


FIGURE 4.2 – Architecture du bottleneck qui vient remplacer les deux convolutions 3\*3 pour gagner en efficacité

Cela donne les réseaux ci-dessous selon le nombre de couches souhaitées :

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1			average pool, 1000-d fc, softmax		
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

The overall architecture for all network

FIGURE 4.3 – tableau qui récapitule les différentes couches de convolutions

Dans notre implémentation du ResNet (fichier ResNet.py), nous avons ajouté notre touche personnelle en donnant une profondeur variable pouvant varier de 3 à 5 pour s'adapter à des tailles d'image en entrée de 512\*512, 256\*256 ou 128\*128. Cela, combiné à une option ("small", "medium" ou "large"), garanti une bonne adaptabilité.

## 4.1.2 Résultats

Les résultats ne sont pas bons, avec 5 ou 10 classes, quel que soit le nombre d'epoch, avec ou sans augmentation de données, le réseau n'apprend pas. Le réseau est normalement plutôt bon pour des problèmes de classifications cependant ici les images restent peu normalisées, c'est-à-dire qu'il y a une grande variance de qualité des images, alors que l'identification se cache dans les détails. Malgré un zoom sur le dauphin, le fond varie systématiquement tout comme les couleurs, ce qui nous supposons rend trop difficile l'apprentissage.

## 4.2 EffNet

### 4.2.1 Pourquoi utiliser EffNet

EffNet est un modèle très rapide comparé aux modèles plus anciens tels que le ResNet et ce car il a besoin d'un nombre moins important de paramètres dans son réseau.

L'autre avantage est que l'on peut utiliser différentes versions de ce modèle selon la machine (GPU, CPU) dont on dispose.

Nous avons réalisé que la grande majorité des personnes participants à ce challenge utilisaient EffNet.

### 4.2.2 Objectif du EffNet et scaling

Les réseaux convolutifs (CNN) sont désormais omniprésents en vision par ordinateur. Ils permettent d'obtenir de très bons résultats dans de nombreux domaines, mais une de leur faiblesse est la mise à l'échelle du modèle (model scaling) c'est-à-dire décider d'agrandir la taille du modèle pour obtenir une meilleure précision.

Un CNN peut être mis à l'échelle en trois dimensions : profondeur, largeur et résolution. La profondeur du réseau correspond au nombre de couches dans un réseau. La largeur est associée au nombre de neurones dans une couche ou ici au nombre de filtres dans une couche convective. La résolution est simplement la hauteur et la largeur de l'image d'entrée. La figure ci-dessus donne une image plus claire de la mise à l'échelle dans ces trois dimensions.

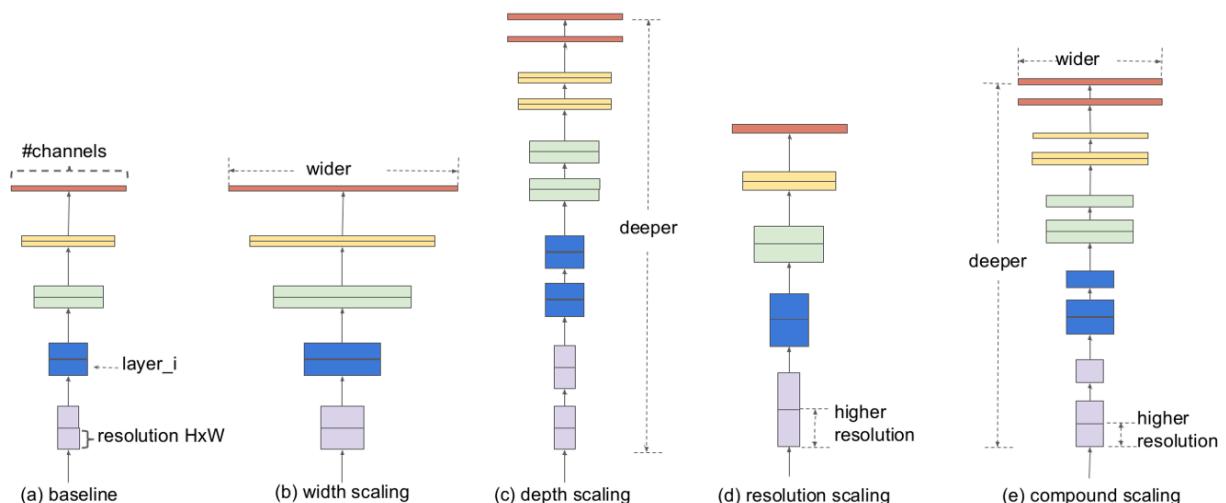


FIGURE 4.4 – (a) est un exemple de réseau de base; (b)-(d) sont des mises à l'échelle conventionnelles qui n'augmentent qu'une dimension de la largeur, de la profondeur ou de la résolution du réseau. (e) est la méthode de mise à l'échelle composée qui est présentée dans l'article et qui met uniformément à l'échelle les trois dimensions avec un rapport fixe.

source : <https://towardsdatascience.com/efficientnet-scaling-of-convolutional-neural-networks-101-10f3a2a2a2>

Pour connaître la taille optimale du modèle, il faut réaliser plusieurs essais, ce qui est coûteux en ressources et en temps.

Les EfficientNet sont des modèles créés par Google en 2019 dans le but de produire à la fois une meilleure précision, mais également d'améliorer l'efficacité des modèles en réduisant le nombre de paramètres.

Pour cela, ils ont créé une architecture simple, le EfficientNet-B0 et ont fourni une fonction permettant d'agrandir facilement la taille du modèle à partir de l'architecture B0.

La méthode de mise à l'échelle peut être généralisée à des réseaux existants tels que le ResNet, mais un réseau de base solide est nécessaire pour obtenir les meilleurs résultats.

EfficientNet-B0 obtient 77.3% de précision sur ImageNet avec seulement 5.3M de paramètres alors que Resnet-50 obtient 76% de précision avec 26M de paramètres.

#### 4.2.3 Architecture du EffNet

Il existe 8 modèles de EffNet. Tous les réseaux commencent par le "stem" et finissent par les couches finales. Ces deux éléments sont décrits sur l'image ci-dessous :

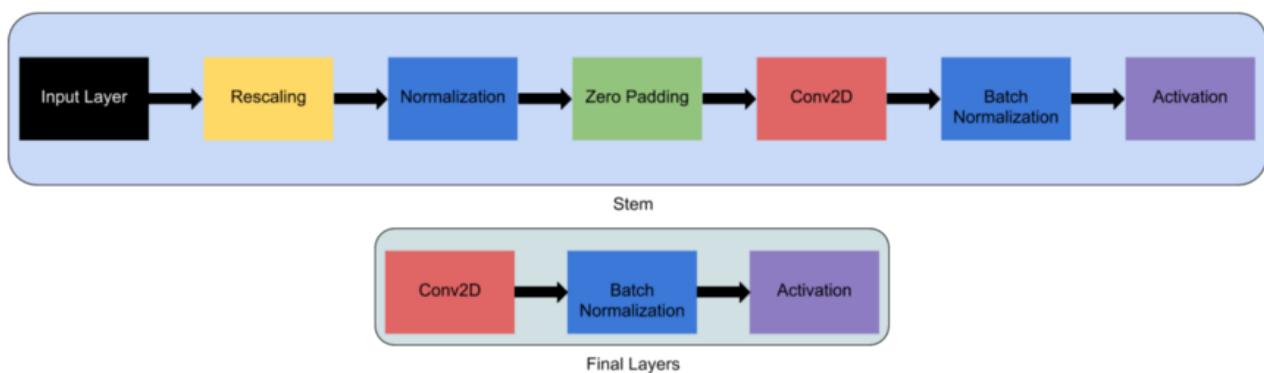


FIGURE 4.5 – source : <https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142>

Entre ces deux éléments viennent des blocs constitués de différents modules :

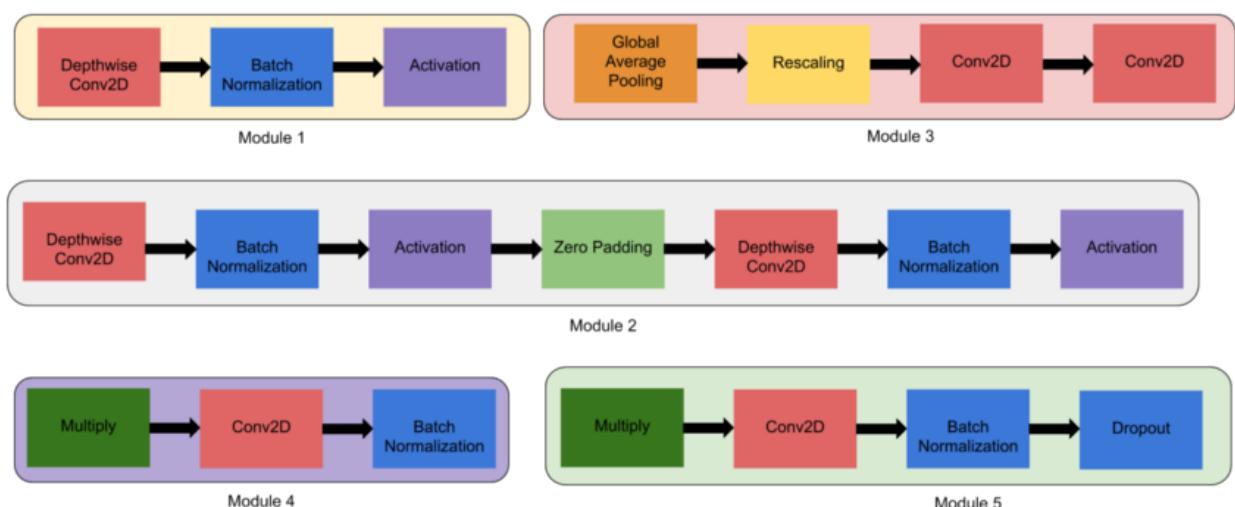


FIGURE 4.6 – source : <https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142>

Ces modules sont ensuite regroupés dans des sous-blocs :

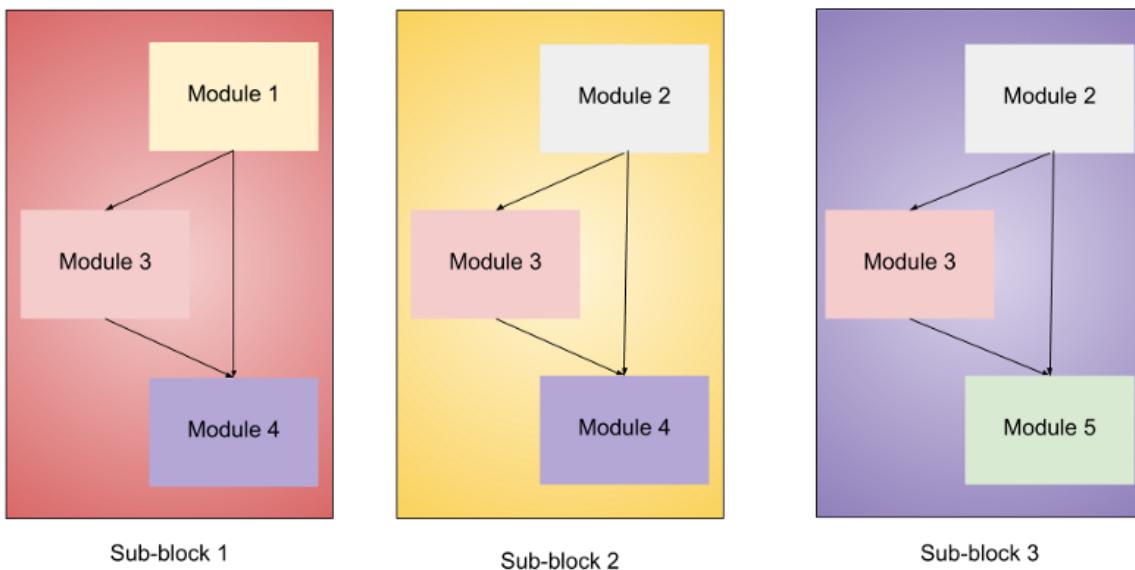


FIGURE 4.7 – source : <https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142>

Finalement, on assemble ces sous-blocs dans des "blocs" pour former une architecture :

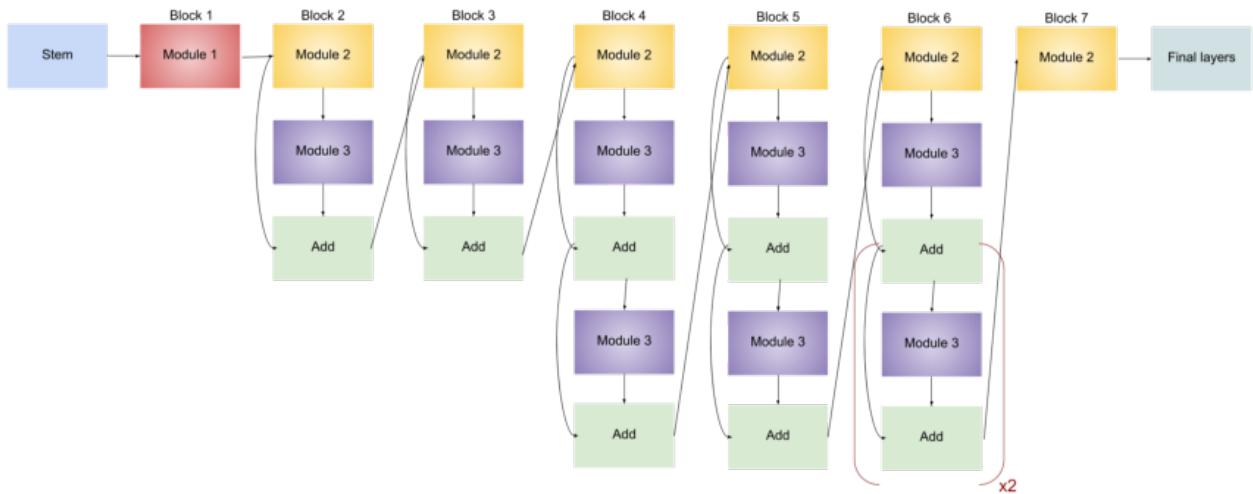


FIGURE 4.8 – source : <https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142>

Ici, EffNet-B0 commence par le stem, commun à tous. Puis il y a un sous-bloc 1 (et non pas "Module 1" comme indiqué à tort sur le schéma), puis un sous-bloc 2, etc.

Les EffNet-B0 à B7 ont exactement la même structure. On voit sur l'image ci-dessous, que le EffNet-B7 a simplement utilisé plus de sous-blocs 3 entre deux sous-blocs 2.

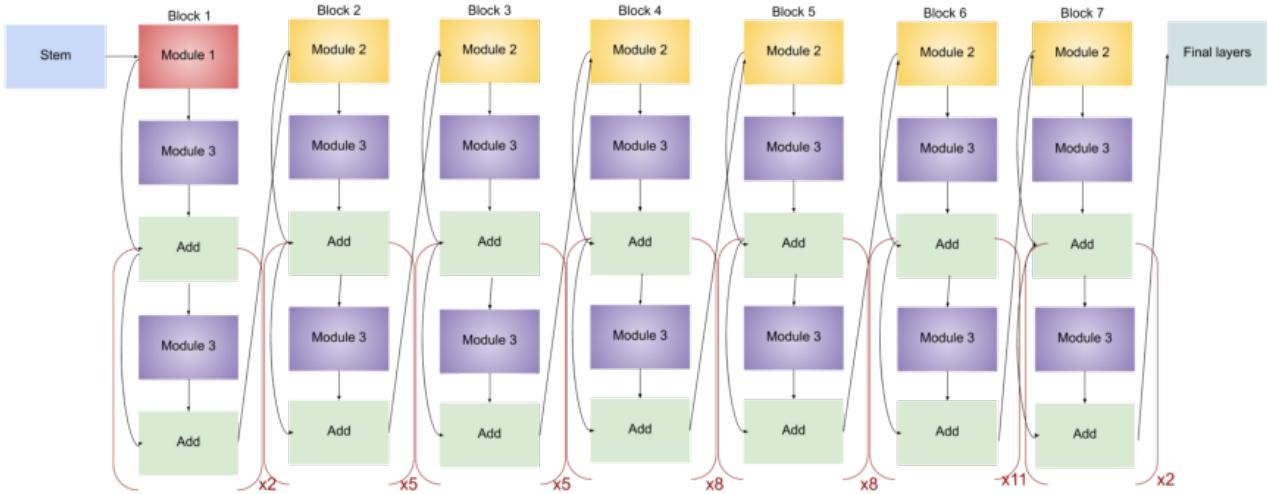


FIGURE 4.9 – source : <https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142>

#### 4.2.4 Transfer Learning

L'apprentissage par transfert (ou Transfer Learning en anglais) est une technique d'apprentissage automatique dans laquelle un modèle entraîné sur une tâche est réutilisé pour une deuxième tâche du même genre.

Les réseaux d'apprentissage profond sont gourmands en ressources et coûteux en calculs avec des millions de paramètres. Ces réseaux sont entraînés avec une quantité massive de données afin d'éviter le sur-apprentissage. Ainsi, lorsqu'un modèle de pointe est créé, les chercheurs consacrent souvent beaucoup de temps à son entraînement.

L'avantage de l'apprentissage par transfert est que nous pouvons réutiliser soit l'ensemble du modèle, soit une certaine partie de celui-ci. De cette façon, nous n'avons pas besoin d'entraîner le modèle entier. En particulier, l'apprentissage par transfert permet de gagner du temps et d'obtenir de meilleures performances.

Pour notre projet, nous avons choisi d'implémenter un EfficientNet-b0 pré-entraîné et disponible sur la bibliothèque "Timm". Timm est une bibliothèque d'apprentissage profond créée par Ross Wightman. Il s'agit d'une collection de modèles de vision par ordinateur, de couches, d'optimiseurs, etc. Nous n'avons pas réussi à savoir sur quoi il a été entraîné, ni pendant combien d'epoch.

#### 4.2.5 Résultats

Nous n'avons pas réussi à obtenir de résultats en utilisant ce réseau, cependant nous avons pu constater la différence d'efficacité entre ResNet et EffNet.

Après la deadline du challenge, nous avons appris qu'il faut fournir en entrée d'un réseau pré-entraîné des images de même taille que celles sur lequel il a été entraîné. Or notre EffNet a, comme beaucoup de modèles, été entraîné sur des images 224x224 et nous utilisions des images en 512x512 (par la suite redimensionnées en 256x256).

Ce n'est sûrement pas la seule chose qui ne fonctionne pas dans notre code, mais il serait intéressant de voir s'il y a une différence avec des images de taille 224x224.

## 4.3 ArcFace

### 4.3.1 Utilisation des embeddings

Un CNN typique pour la classification consiste en une extraction de caractéristiques et une classification. Au cours de l'entraînement, le modèle apprend les caractéristiques uniques du visage et produit les embeddings de caractéristiques dans le processus d'extraction de caractéristiques. Ces embeddings correspondent à la dernière couche du réseau avant la classification et sont comme une "empreinte digitale" de l'image.

Ces embeddings nous aident à comprendre s'il existe une similarité entre deux personnes en utilisant soit la similarité en cosinus, soit la distance carrée entre deux vecteurs. Les vecteurs de deux images différentes de la même personne auront une grande similarité et une faible distance, et deux images de personnes différentes auront une faible similarité et une grande distance.

### 4.3.2 Qu'est-ce qu'ArcFace

Le but de ce challenge est de reconnaître de façon unique un individu (dauphin ou baleine) à partir de ses caractéristiques physiques (aileron, etc). Cela ressemble fortement à un problème de reconnaissance de visage.

ArcFace offre une loss qui permet d'obtenir, à partir de 2 embeddings de caractéristiques, une similarité permettant de déterminer si les deux images à l'origine des 2 embeddings montrent la même personne.

ArcFace fut surtout mis sous les projecteurs après son utilisation lors de la Google Landmark 2019. Lors de cette compétition, il faut annoter les images d'une base de données de plus de 1.2 million d'images réparties sur 15 000 classes avec un nombre d'images par classe allant de 1 à plusieurs milliers.

Lors de cette édition, la 1ere et 3eme solutions ont utilisé ArcFace.

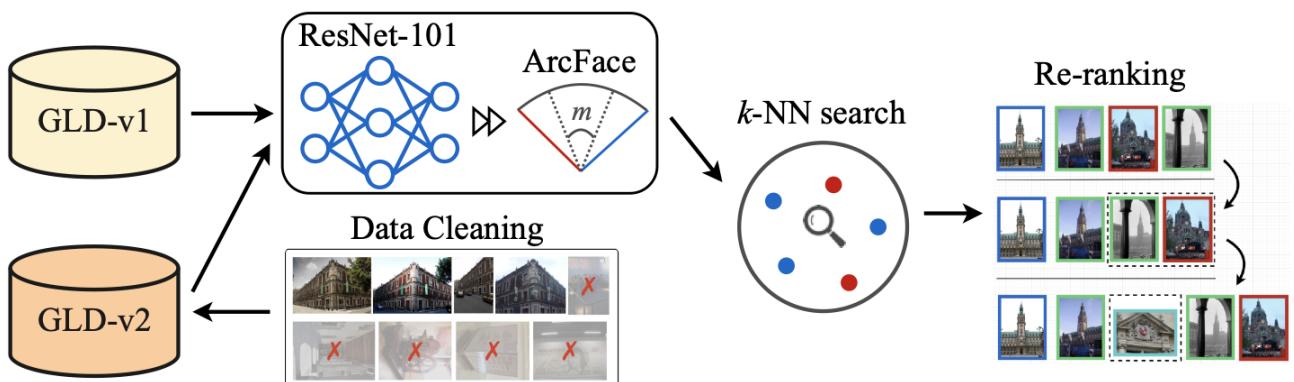


FIGURE 4.10 – source : <https://github.com/lyakaap/Landmark2019-1st-and-3rd-Place-Solution>

Comme on peut le voir sur l'image ci-dessus. Après avoir utilisé ArcFace pour projeter les embeddings dans un espace où les images se ressemblent sont proches, on fait un NN search.

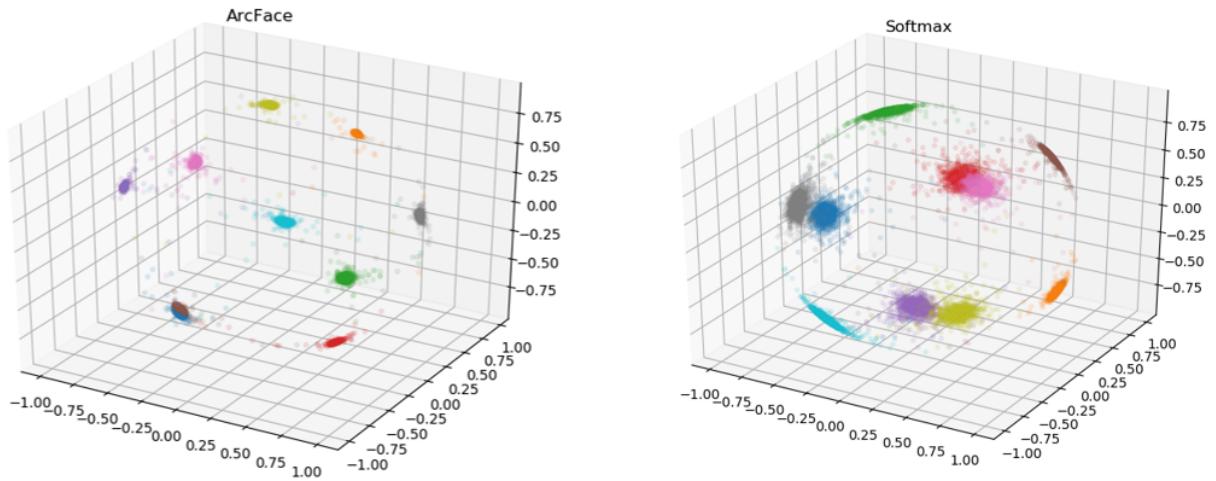


FIGURE 4.11 – source : <https://medium.com/analytics-vidhya/face-recognition-and-arcface-additive-angular-margin-loss-for-deep-face-recognition-44a>

Le NN search (recherche des plus proches voisins) permet, avec un embedding de test, d'obtenir une liste des classes dont il est le plus proche (ses voisins) avec une mesure de probabilité.

### 4.3.3 Notre implémentation et nos résultats

Nous avons utilisé ArcFace comme dernière couche de notre réseau de neurone et récupéré les embeddings.

Puis pour notre donnée de test, nous avons cherché les plus proches voisins dans ces embeddings et retourné les 5 classes les plus proches.

Nous avons également fixé un seuil de confiance, au-delà duquel un individu avait plus de chance d'être un nouvel individu, jamais rencontré, ce qui expliquerait la faible confiance des résultats.

Nous avons rencontré quelques soucis avec les plus proches voisins qui donnaient pour les premières images de test de très grosses confiances pour plusieurs classes et qui ensuite donnaient une confiance de 0 pour chaque image de test.

Il était difficile de savoir si le problème venait des plus proches voisins ou des embeddings.

## 5 Recherche d'hyperparamètre

Lors de l'entraînement de notre modèle, nous avons commencé par donner des valeurs par défaut aux paramètres comme le taux d'apprentissage, ou encore la marge utilisée par ArcFace. Or ce n'est pas forcément la bonne solution, car on pourrait passer à côté d'un modèle bien plus efficace par un simple changement de paramètre.

Nous avons donc mis en place une classe Python `HyperparameterSearchManager` pour répondre à ce problème.

Cela nous permet de parcourir une liste de valeurs pour nos trois paramètres principaux dans ArcFace : le taux d'apprentissage, la marge et l'échelle. Parmi ces valeurs, nous sélectionnons le meilleur modèle en utilisant le critère de la meilleure justesse sur l'ensemble de validation.

Remarque : le EffNet utilisé étant déjà pré-entraîné, nous ne pouvons pas modifier ses paramètres, car une structure différente empêcherait de charger les poids existants dans le réseau.

Le but était de faire une recherche d'hyperparamètre sur un de nos deux sous-ensembles de données, en espérant que le meilleur modèle sur ces petits ensembles soit aussi le meilleur sur la base de données complète.

Malheureusement, la recherche d'hyperparamètre étant très demandante pour nos machines et le serveur Kaggle étant monopolisé par entraînement complet, nous n'avons pas pu effectuer de recherche d'hyperparamètres concluante avant la fin de la compétition HappyWhale.

# 6 Soumission des résultats sur Kaggle

## 6.1 Mise en Place

Dans le but d'effectuer la soumission de nos résultats sur Kaggle, nous avons créé un notebook attaché à la compétition HappyWhale. Sur ce notebook, les bases de données originales et rognées sont toutes les deux accessibles, et notre dépôt GitHub est cloné dans l'environnement d'exécution. Une fois cela mis en place, le code écrit pour l'exécution en local est quasiment utilisable directement. Seul la classe `DataManager` a dû être modifiée pour s'adapter à l'arborescence de fichier des bases de données sur Kaggle.

Une fois une exécution lancée, la soumission sera directement disponible dans la compétition HappyWhale et sera attribué automatiquement un score. La soumission se faire sous la forme d'un fichier CSV contenant deux colonnes "image" et "predictions". La colonne "predictions" contient pour chaque image nos 5 identifiants d'individus, ou la mention "new\_individual" dans le cas où la prédiction ne correspond pas à un individu connu.

Les scores sont attribués selon la formule ci-dessous :

Submissions are evaluated according to the Mean Average  
Precision @ 5 (MAP@5):

$$MAP@5 = \frac{1}{U} \sum_{u=1}^U \sum_{k=1}^{\min(n,5)} P(k) \times rel(k)$$

## 6.2 Résultats

Durant toute la compétition, un tableau de classement des équipes est présenté, il représente le classement sur 24% des données de test. Ensuite, à la fin de la compétition, le classement privé est dévoilé et donne le score sur les 76% des données de test restante, son but est de vérifier que les modèles n'ont pas sur-appris pour maximiser le score sur l'ensemble de test.

Les résultats que nous présentons ici sont ceux du 18 avril 2022, fin de la compétition et donc n'ont pas été vérifiés par les organisateurs de la compétition.

Le meilleur score sur le classement public est de 0.897 par l'équipe "Rist Whales" et de 0.875 par "Preferred Dolphin" sur le classement privé. Pour comparaison, le score de la soumission d'exemple (une soumission aléatoire) est de 0.023 sur le classement public et 0.028 sur le classement privé.

Notre score est affiché ci-dessous :

1412	<b>WhalePlayed</b>		0.103	2	7h
1392	<b>▲ 20 WhalePlayed</b>		0.122	2	7h

FIGURE 6.1 – Notre classement public suivi de notre classement privé

Le nombre indiqué sur la gauche est notre classement, sur la droite, on retrouve notre score sur l'ensemble de test, suivi du nombre de soumissions et du temps passé depuis la dernière soumission.

Notre résultat est donc meilleur que l'aléatoire, mais n'est pas pleinement satisfaisant par rapport au score obtenu par les autres participants.

## 7 Conclusion

La participation à ce challenge Kaggle, bien qu'ayant donné des résultats mitigés, a été l'occasion pour nous de mettre en pratique nos connaissances à la fois sur le traitement de données et sur la mise en place de solution de machine learning.

Au cours de ce projet, nous avons pu mettre en place une démarche d'analyse, traitement et augmentation de données, notamment avec l'utilisation d'algorithme de localisation comme Yolo ou Detic. Ensuite, nous avons utilisé des méthodes de classification d'images comme ArcFace, combiné à un ResNet ou Effnet. Enfin, nous avons dû réfléchir à une méthode permettant de classifier les individus inconnus comme "nouveaux individus", la méthode que nous avons retenue étant un seuil sur le degré de confiance de notre réseau.

Ce projet a donc été l'occasion de se poser beaucoup de questions sur l'utilisation des méthodes que l'on a pu voir dans nos cours et les discussions sur Kaggle nous ont été d'une grande aide pour découvrir de nouvelles façons de procéder.

# Liens cités dans ce rapport

<https://github.com/facebookresearch/Detic>, 6  
<https://github.com/julien-levarlet/HappyWhale>, 4  
<https://github.com/lyakaap/Landmark2019-1st-and-3rd-Place-Solution>, 17  
<https://github.com/ultralytics/yolov5>, 5  
<https://medium.com/analytics-vidhya/>  
    face-recognition-and-arcface-additive-angular-margin-loss-for-deep-face-recognition-18  
<https://towardsdatascience.com/>  
    complete-architectural-details-of-all-efficientnet-models-5fd5b736142, 14–16  
<https://towardsdatascience.com/>  
    efficientnet-scaling-of-convolutional-neural-networks-done-right-3fde32aef8ff, 13  
<https://towardsdatascience.com/>  
    review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-11  
<https://www.imaios.com/en/Company/blog/>  
    AI-for-medical-imaging-data-augmentation, 8  
<https://www.kaggle.com/competitions/happy-whale-and-dolphin/discussion/305503>, 6