Mémoire de stage

Sujet : Mise au point de la chaîne de traitement des images de l'observatoire ORCHAMP, avec pour objectif la capacité à analyser end-to-end les centaines de milliers d'images de la faune arrivant en flux régulier depuis le terrain.

Étudiant : Elias CHETOUANE

Maître de stage : Wilfried THUILLER

Encadrant : Vincent MIELE

Master 2 Sciences des données et systèmes complexes

Année 2021 / 2022

Table des matières :

1. Le projet ORCHAMP

1.1 Introduction

En écologie, différentes méthodes peuvent être mises en place pour étudier les animaux, comme par exemple la technique de capture/recapture avec marquage qui consiste à capturer des animaux de l'espèce que l'on souhaite étudier pour les marquer et les libérer, puis les recapturer plus tard afin d'étudier l'évolution de leur population. Il est aussi possible de capturer les animaux et leur mettre un implant GPS pour étudier leurs déplacements. Une technique qui est de plus en plus utilisée consiste à mettre des « pièges photographiques » disséminés à divers endroits de leur lieu de vie ou de passage afin d'étudier leur présence (la fréquence à laquelle ils ont été observés et leur proportion par rapport à la population totale des espèces de ce même habitat par exemple).

Les pièges photographiques sont de petits appareils photographiques qui se déclenchent automatiquement et prennent une photographie lorsque du mouvement est détecté dans le champ visible de l'appareil. Cela permet donc de prendre une photographie automatiquement lorsqu'un animal se trouve à proximité.

La difficulté rencontrée avec la technique du piège photographique est que l'appareil se déclenche quelque soit l'animal à proximité, et peut aussi être déclenché par un simple coup de vent déplaçant des feuilles ou des branches, auquel cas la photographie obtenue ne présentera aucun animal. Cela a pour conséquence que pour chaque piège photographique on obtient une grande quantité d'images qui ne seront pas utiles à l'étude, ce qui requiert un pré-traitement pour sélectionner uniquement les images utiles.

Souvent, les équipes de recherche font ce travail de pré-traitement eux-même : les photographies sont toutes vérifiées et triées manuellement, ce qui est très long. Certaines équipes se tournent aussi vers les sciences participatives : les photographies sont rendues publiques et des volontaires peuvent proposer de classer les images. La classification qui est la plus récurrente pour chaque image est alors considérée comme la bonne. Mais ceci pose problème lorsque sur les images se trouvent des humains (cela peut arriver si le piège photographique est proche d'un sentier par exemple, et cela pose un problème de droit à l'image) ou des espèces rares ou protégées comme l'ours ou le loup.

C'est pourquoi, avec l'avancée de la vision par ordinateur de nombreux chercheurs se tournent vers le traitement automatique de ces images. C'est donc cette direction qu'a souhaité prendre l'observatoire ORCHAMP, pour lequel j'ai travaillé durant mon stage.

Figure 1 : image d’un piège photographique (utilisé par ORCHAMP)

1.2 Objectifs et protocoles

Avant de voir plus en détail quelles techniques de traitement d’image sont mises en place pour ORCHAMP [1], nous allons voir les objectifs de ce dernier et les protocoles qui ont été mis en place.

« L’Observatoire spatio-temporel de la biodiversité et du fonctionnement des socio-écosystèmes de montagne (ORCHAMP), porté par le Laboratoire d’Écologie Alpine (LECA), s’intéresse à la biodiversité et au fonctionnement des écosystèmes des Alpes françaises en réponse aux effets des changements climatiques et d’utilisation des terres. A cette fin, il s’agit d’observer, d’analyser et de modéliser les changements environnementaux sur la base de multiples protocoles de suivi. » (source : https://orchamp.osug.fr/)

Ces multiples protocoles de suivi sont organisés en placettes localisées suivant un gradient vertical, tous les 200 mètres d’altitude. Sur chacune de ces placettes sont régulièrement effectuées des analyses des sols et des relevés botaniques. En plus de cela, sur chaque placette sont installés un micro enregistrant les sons durant quelques secondes à intervalles réguliers pour des relevés acoustiques et deux pièges photographiques.

De très nombreuses images de la faune des Alpes sont donc prises en continu et doivent être classées selon l’espèce qui a été photographiée dans le but d’effectuer des analyses visant à étudier la présence de différentes espèces selon l’altitude et à étudier leur évolution et leur adaptation aux changements climatiques. Ces images étant très nombreuses le fait de les classer manuellement est très long, d’où le besoin de traiter ces images automatiquement par un algorithme de vision par ordinateur.

1.3 Chaîne de traitement

Dans le cadre de mon stage, mon rôle était de concevoir et de mettre en place une chaîne de traitement des images en provenance des pièges photographiques. Cette chaîne de traitement va de la récupération des images depuis les cartes mémoire des appareils à la classification des images, dans le but de permettre des requêtes pour faciliter les différentes analyses statistiques.

Le but final de cette chaîne de traitement est que les équipes chargées de la mise en place des pièges photographiques et de la récupération des images puissent récupérer les cartes mémoire des appareils, simplement les copier sur un support adéquat et qu'ensuite se lance automatiquement la classification automatique des images. Ensuite, il faut que la chaîne de traitement permette aux équipes chargées des analyses d'effectuer des requêtes sur les résultats obtenus après classification. Un exemple de requête qui doit pouvoir être effectuée serait la suivante : quelle est pour chaque espèce détectée le nombre de fois que cette espèce a été détectée en fonction de l'altitude ?

Pour cela, différents étapes doivent se succéder : la première consiste à récupérer les images et les stocker (donc trouver sur quel support ou système elles doivent être stockées), puis elles doivent être classées par le programme de reconnaissance d'images selon l'espèce présente sur la photo pour alimenter une base de données sur laquelle pourront être effectuées les requêtes nécessaires aux différentes analyses. Enfin, une dernière étape s'ajoute : une étape de validation manuelle sera effectuée régulièrement pour s'assurer que les images ont été bien classées par l'algorithme, ce qui implique qu'il faut un moyen de modifier à posteriori les prédictions de l'algorithme.

Dans le but d'avoir pour cette chaîne de traitement la meilleure performance possible concernant la reconnaissance d'image, j'ai travaillé en parallèle sur un autre projet nommé DeepFaune que nous verrons plus en détail par la suite. En effet, l'API de DeepFaune est utilisée dans la chaîne de traitement des images d'ORCHAMP pour la phase de classification des images. Le fait de travailler en parallèle sur ce projet m'a donc permis d'utiliser au mieux l'API développée et d'adapter cette dernière aux besoins du projet ORCHAMP.

2. Cadre du stage

Mon stage a été financé par le projet ORCHAMP, avec à sa tête Monsieur Wilfried Thuiller, directeur de recherche au CNRS, écologue et biostatisticien au Laboratoire d'Écologie Alpine (LECA) de Grenoble. C'est en revanche Monsieur Vincent Miele,

ingénieur de recherche CNRS du laboratoire de biométrie et biologie évolutive (LBBE) de Lyon, hébergé au LECA et spécialiste des techniques de vision par ordinateur par deep learning en écologie [2] qui m'a encadré.

Vincent Miele étant à la tête du projet DeepFaune [3] (dont nous verrons les détails par la suite), j'ai pu participer à ce projet en parallèle de mon sujet principal qui est la chaîne de traitement. En effet, ce projet est tout de même très lié à la chaîne de traitement, car il avait été décidé que c'est ce programme qui serait utilisé au cœur de la chaîne de traitement pour la prédiction des espèces sur les images.

Pour cela, j'ai été accueilli au LECA de Technolac (lieu situé au campus du Bourget du Lac, proche de Chambéry).

Pour travailler sur ORCHAMP et DeepFaune en parallèle, nous avons utilisé le gestionnaire de version git, ce qui a permis de simplifier les communications et de suivre plus simplement l'avancement de chaque projet, ainsi que des différentes tâches à réaliser via le système d'« issues ».

Afin de tester les différents programmes au fur et à mesure de ma progression, j'ai utilisé un calculateur situé à Lyon ainsi que de Gricad (machines de calcul de Grenoble). Pour cela, une communication via ssh a été mise en place afin que je puisse faire tourner nos programmes en situation réelle sur de nombreuses données hébergées sur ces machines distantes.

Nous avons aussi fait régulièrement des réunions en visioconférence afin de préciser les besoins des différents projets au fur et à mesure. Tout cela a donc permis d'éviter de trop me déplacer. J'ai donc réalisé la grande majorité de mes missions au LECA de Technolac.

Pour travailler sur ORCHAMP, j'ai été introduit dans une équipe qui comptait
Monsieur Bruno Bzeznik, ingénieur de recherche GRICAD au LECA de Grenoble,
Monsieur Julien Renaud, ingénieur d'étude pour ORCHAMP, ainsi que moi-même.

3. Le projet DeepFaune

3.1 Objectifs

DeepFaune est un logiciel dont le code a été écrit par Vincent Miele, pour lequel j'ai ensuite pu apporter ma contribution. Voici un descriptif complet des fonctionnalités et objectifs (ma contribution sera expliquée en détail par la suite).

L'objectif de DeepFaune est de développer un outil de reconnaissance des espèces animales en France à partir d'images ou de vidéos. C'est un outil se voulant le plus

simple d'utilisation possible : il possède une interface graphique facile à prendre en main et possède une version pour chaque système d'exploitation. Il est aussi peu demandeur en ressources : il est possible de faire fonctionner le programme sur son ordinateur personnel, sans avoir besoin de carte graphique ou de performances élevées en terme de mémoire et de processeur.

C'est un outil rapide (le temps passé sur chaque image est de l'ordre de la seconde) et paramétrable simplement. Les résultats sont donnés sous forme de csv mais il est aussi possible de les visualiser via l'application (il est possible de visualiser chaque image avec la prédiction qui lui a été associée avec le score qui lui a été attribué, et il est même possible de modifier cette prédiction manuellement pour qu'elle apparaisse dans les résultats).

Tout cela est fait dans le but que tout le monde puisse utiliser ce logiciel simplement et de manière intuitive même pour quelqu'un n'ayant pas l'habitude d'utiliser ce genre d'outil, mais il est aussi possible de l'utiliser en tant qu'API et donc de changer d'outil de détection au besoin ou l'outil de classification. C'est un logiciel gratuit et distribué sous licence CeCill, et le code est disponible sur le git dédié. [4]

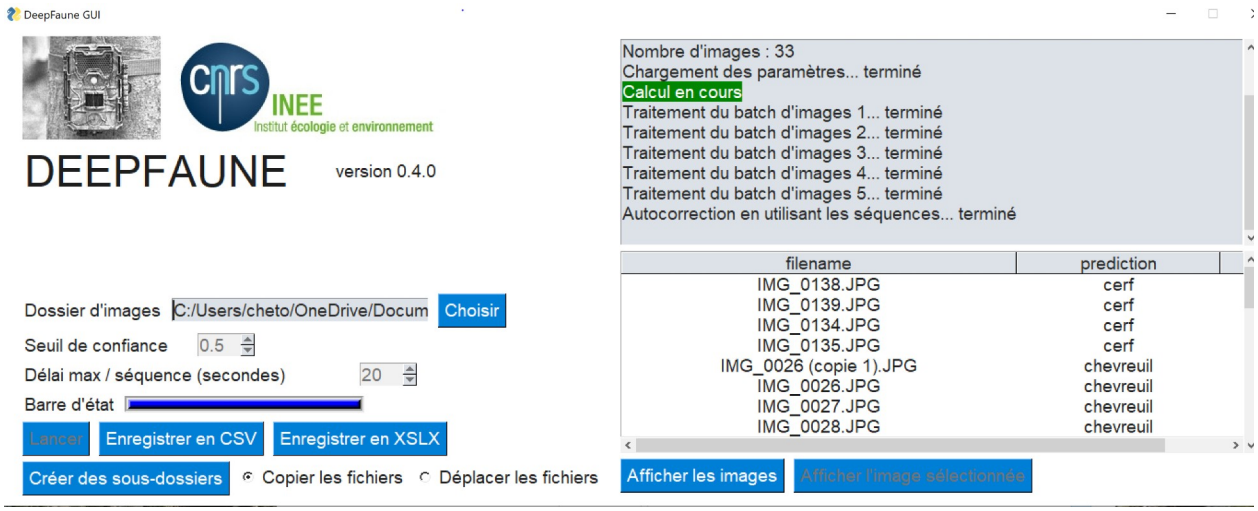

Figure 2 : capture d'écran de l'interface graphique (GUI) de DeepFaune

L'utilisation de la GUI de DeepFaune est simple : il faut, dans un premier temps, sélectionner la langue (entre français et anglais) ainsi que le type de données que l'on souhaite analyser (images ou vidéos). Ensuite, il faut choisir un répertoire sur notre ordinateur contenant les images (ou vidéos) à analyser (le bouton « choisir » permet d'ouvrir une fenêtre de navigation pour sélectionner le bon répertoire). Le programme va alors sélectionner toutes les images disponibles récursivement depuis le répertoire donné. Ensuite, il est possible de paramétrer le seuil de confiance (le programme associe à chaque prédiction un indice de confiance et, s'il est inférieur au seuil mentionné, la

prédiction est notée « indéfinie ») et le délai entre les images d'une séquence (les séquences d'images correspondent à une fonctionnalité qui sera expliquée par la suite). Ici, les valeurs affichées sur la figure 2 sont celles par défaut. Une fois que le dossier a été renseigné et que les paramètres sont ceux désirés, il suffit de cliquer sur « lancer ». Le programme va alors séparer l'ensemble des images en « paquets » de 8 images chacun (les « batches », aussi appelés tenseurs) et va afficher la prédiction obtenue à côté du nom de l'image. Augmenter la taille d'un tenseur augmente les performances mais nécessite plus de ressources (car plus d'images sont chargées en même temps). Il est ensuite possible de visualiser ces résultats depuis l'interface (il est d'ailleurs possible de visualiser les images avec la prédiction associée et l'indice de confiance calculé, et il est aussi possible de modifier la prédiction sur l'interface afin que cette nouvelle prédiction soit affichée dans les résultats à la place de la précédente). Il est aussi possible de sauvegarder ces résultats en csv ou xlsx. Le bouton « créer des sous-dossiers » permet quant à lui de créer un dossier « deepfaune » dans le dossier sélectionné où se trouvent les images et de déplacer ou copier (au choix) les images analysées dans ce dossier, dans lequel seront créés des sous-dossiers pour chaque espèce trouvée dans les images et chacun de ces sous-dossier contiendra les images dont l'espèce prédite est la même.

3.2 Implémentation et modèle

Le programme DeepFaune est implémenté en Python avec la librairie TensorFlow. L'approche se fait en deux étapes : la première étape est une étape de détection de l'animal dans l'image (le « détecteur ») via un modèle yolov4 (darknet) [5]. Une fois l'animal détecté, la partie de l'image où se trouve l'animal est séparée du reste de l'image pour la donner à un deuxième modèle faisant la classification (le « classifieur »). Pour cette étape de classification, le modèle s'appuie sur le « transfer learning », qui consiste à se baser sur des modèles pré-existants pour améliorer l'efficacité du modèle et faciliter l'apprentissage, ici nous utilisons EfficientNet-B3 (un réseau de neurones convolutifs provenant de ImageNet).

Lorsque le premier modèle ne détecte pas d'animal sur l'image (ou du moins ne détecte pas d'animal avec un indice de confiance suffisant, que nous avons fixé à 0,5), cette dernière est considérée comme vide et est immédiatement classée vide, sans passer par l'étape de classification (le détecteur étant supposé être plus performant pour trouver l'animal).

Le réseau de neurones permettant cette classification calcule un indice de confiance pour la classification. Si celle-ci est inférieure à celle définie par l'utilisateur (0,5 par défaut, l'indice allant de 0 à 1), l'image est classée comme « indéfinie ».

A ce jour, les espèces étant reconnues par le modèle sont les suivantes : blaireau, bouquetin, cerf, chamois, chat, chevreuil, chien, écureuil, humain, lagomorphe, loup, lynx, marmotte, micromammifère, mouflon, mouton, mustélidé, oiseau, renard, sanglier, vache, véhicule.

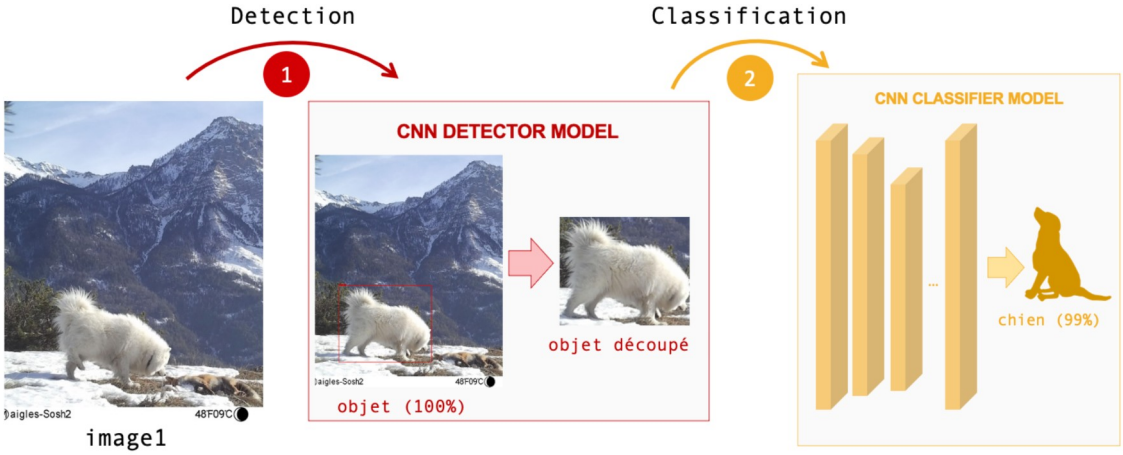

Figure 3 : approche en deux étapes pour la reconnaissance des taxons présents dans l'image
(source : https://www.deepfaune.cnrs.fr/)

Le fait de séparer ainsi la détection et la classification permet de rendre chaque tâche plus performante (car l'entraînement de chaque réseau de neurones se fait sur une tache plus précise), et permet d'éviter certains raccourcis d'apprentissage. Par exemple un réseau de neurones entraîné à différencier les images de chien et de loup risque de faire cette différence en s'appuyant sur le décor, et de par exemple systématiquement associer des images prises sous la neige à des loups sans même s'attarder sur l'apparence de l'animal en question.

Nous avons opté pour un détecteur sous yolov4 bien qu'il ne soit pas le plus performant en terme de résultats car il s'agit d'un détecteur extrêmement rapide et nécessitant moins de ressources que Megadetector [6] par exemple, qui est actuellement le détecteur le plus utilisé pour les compétitions de vision par ordinateur comme iWildCam [7]. En effet, l'objectif de DeepFaune étant de pouvoir être utilisé par n'importe qui, et sur des machines peu performantes, le fait que Megadetector puisse prendre 8 secondes par image (prises en résolution maximale, pour les performances optimales) quand il tourne sur processeur était trop handicapant par rapport à yolov4 qui ne prend qu'une seconde pour des images en résolution 608x608, résolution pour laquelle les performances restent très bonnes pour les espèces que l'on cherche à détecter.

Quant au classifieur, passer par une méthode de transfer learning est une approche très courante dans le milieu de la vision par ordinateur, et image net est une fois de plus très couramment utilisé dans le milieu des compétitions de vision par ordinateur.

Afin de réduire au maximum les erreurs de détection et de classification, DeepFaune possède un dernier élément mis en place par Vincent Miele : le système de vote par séquence.

En effet, les images obtenues à partir de pièges photographiques sont souvent des séquences de plusieurs images d'une même scène ayant été prises à des moments très proches dans le temps (il est possible de faire en sorte que l'appareil prenne des rafales de photographies ou alors que l'animal ayant déclenché l'appareil reste dans la zone de déclenchement et soit donc à nouveau détecté dans les secondes qui suivent).

C'est pourquoi, une fois les phases de détection et de classification effectuées, les images dont les dates sont espacées de 20 secondes ou moins (valeur par défaut, paramétrable par l'utilisateur) sont considérées comme faisant partie d'une même séquence (pour chaque image de la séquence, s'il y a une image dont la date est espacée de 20 secondes ou moins avec elle, elle est ajoutée à la séquence même si elle a un écart de plus de 20 secondes avec les autres images de cette séquence, ainsi une séquence est une série d'image ou chacune de ces images a une date espacée de 20 secondes ou moins avec l'image qui la précède et celle qui la suit dans la série). Lorsque la prédiction associée aux images d'une même séquence n'est pas la même pour chacune des images de la séquence, un vote est effectué pour déterminer quelle prédiction est la plus probable. En effet, nous estimons qu'il est très peu probable qu'en l'espace de moins de 20 secondes deux animaux différents se soient succédés dans un même lieu. Le résultat du vote dépend donc du nombre de fois qu'une classe à été associée à une image de la séquence, ainsi que l'indice de confiance qui lui est associé. On admet alors que l'animal se trouvant sur les images de la séquence est celui déterminé par le vote par séquence et la classe qui leur a été attribuée par le classifieur est alors remplacée par celle ayant remporté le vote.

DeepFaune est, comme nous venons de le voir, une application utilisable facilement et comportant une interface graphique simplifiant son utilisation. Cette interface graphique fait appel à diverses fonctions constituant l'API de DeepFaune. En effet, le code a été conçu et adapté de manière à pouvoir être utilisé indépendamment des détecteurs et classifieurs utilisés. Il est donc possible, via des appels à d'autres fonctions, d'utiliser un détecteur fonctionnant avec Megadetector pour de meilleurs résultats au prix de ressources et temps de calcul plus importants.

DeepFaune est donc aussi un outil pouvant servir comme API pour la création d'outils de vision par ordinateur utilisant des modèles qui ne sont pas forcément ceux présentés ici. C'est aussi sous sa forme d'API que DeepFaune est utilisé (en combinaison avec Megadetector) dans la chaîne de traitement des images d'ORCHAMP.

En ce qui concerne la classification de vidéos, DeepFaune transforme les entrées vidéos en séries d'images faisant partie d'une même séquence, et fonctionne ensuite de même que pour les images.

4. Travail réalisé

4.1 Modularité de DeepFaune et détecteur

Les premiers travaux qui m'ont été assignés au cours de mon stage furent sur DeepFaune. En effet, étant donné qu'il était prévu d'utiliser DeepFaune dans la chaîne de traitement pour ORCHAMP, il fallait dans un premier temps perfectionner la classification des images par DeepFaune. J'ai donc commencé par la résolution de certains bugs, ce qui m'a permis de me familiariser peu à peu avec le code.

J'ai ensuite fait tourner plusieurs modèles yolo sur le cluster du LBBE (sous SCRUM) afin d'essayer d'améliorer le modèle de détection. Pour cela j'ai réalisé deux modèles fonctionnant avec différentes résolutions d'images : le premier sur des images en 608x608, servant simplement de témoin (le modèle étant déjà en place étant lui aussi en résolution de 608x608) et le deuxième sur des images en 832x832. Cela avait pour objectif de déterminer si un modèle avec une résolution plus élevée permettait ou non de capter significativement plus de petits animaux tels que les oiseaux et les musaraignes qui avaient tendance à poser problème lors de la détection. J'ai entraîné ces deux modèles (en arrêtant l'entraînement dès la stabilisation de la courbe de perte, afin d'éviter que l'un soit mieux entraîné que l'autre) sur le même jeu de données qui m'a été fourni : un ensemble de plusieurs centaines de milliers d'images de la faune française, principalement des Alpes, ayant été classées manuellement.

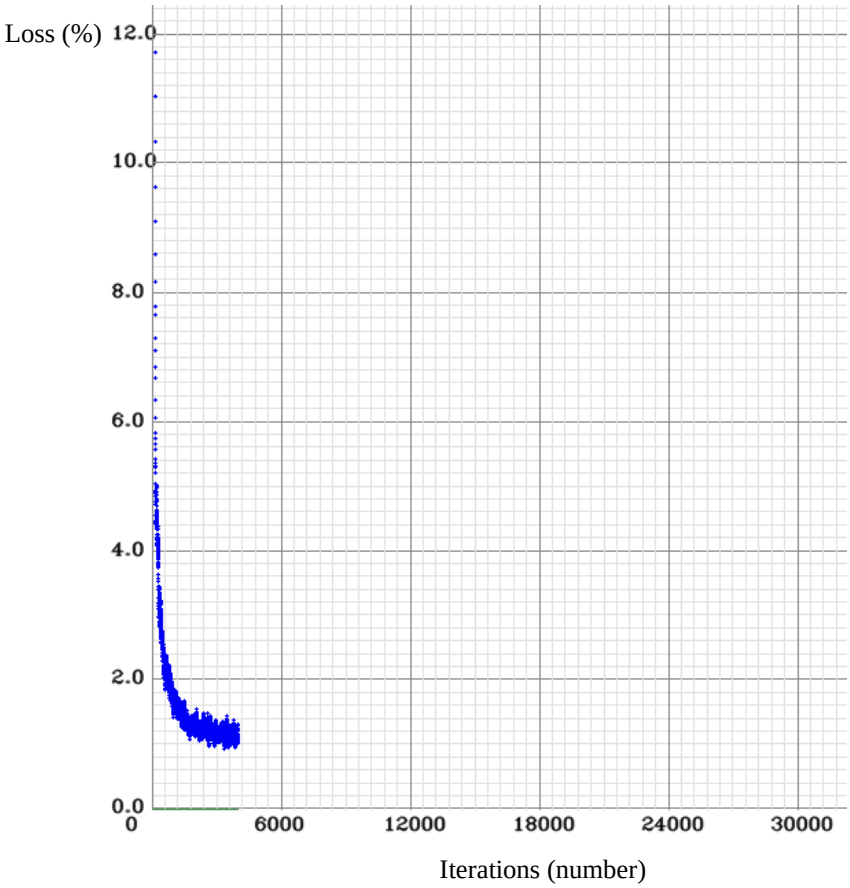

Figure 4 : courbe de perte (en %) en fonction du nombre d'itérations pour le modèle 832x832 sur les données d'entraînement

Nous voyons sur la figure précédente que la perte a commencé à se stabiliser autour de 1,3 %, c'est pourquoi j'ai stoppé l'entraînement au bout de 4000 itérations. Nous avons ensuite comparé leurs performances avec un jeu de données de test (différent du jeu d'entraînement). Suite à ces tests, nous avons vu que le modèle en 832x832 ne donnait pas des résultats suffisamment meilleurs pour justifier de l'utiliser, sachant que le temps de traitement d'une image est doublé par rapport au modèle en 608x608 (car il traite des images contenant deux fois plus de pixels). Je n'ai malheureusement pas gardé les chiffres nous ayant permis cette comparaison, car nous avons rapidement conclu qu'un modèle en 608x608 était préférable, mais que pour la chaîne de traitement nous utiliserions plutôt Megadetector qui offrait de meilleures performances que les modèles yolo.

Nous avons donc opté pour un modèle en 608x608 pour DeepFaune, mais avons privilégié Megadetector pour la chaîne de traitement, car les machines mises à disposition pour faire fonctionner cette dernière sont bien plus puissantes et peuvent donc aisément fournir les ressources dont Megadetector a besoin. Voici à titre indicatif les performances de Megadetector sur nos images de validation :

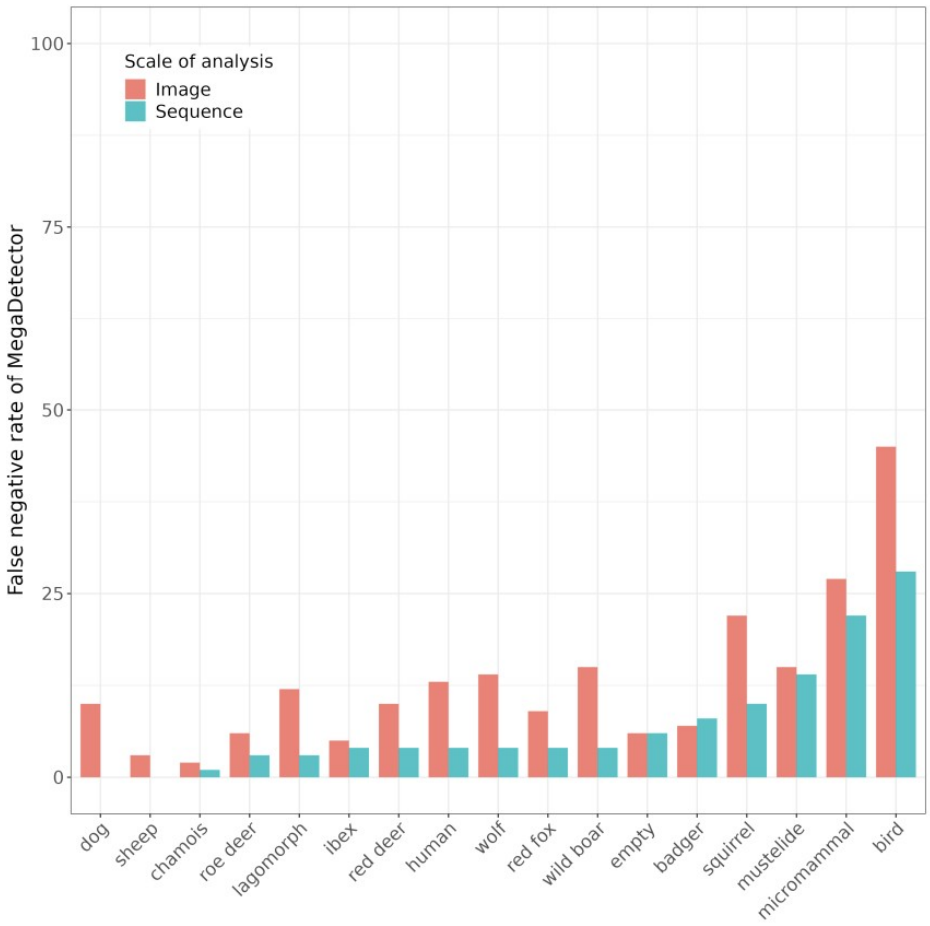

Figure 5 : Performances de Megadetector sur les données de validation, avec et sans utilisation des séquences. (Ici les faux négatifs correspondent aux images considérées comme vides alors qu'elles ne le sont pas).
Source : [3]

Pour faire fonctionner Megadetector avec DeepFaune au sein d'une chaîne de traitement, il a fallu rendre le code de DeepFaune modulaire, pour pouvoir utiliser les différentes fonctions séparément. Je me suis donc chargé de l'écriture et la mise au point de différentes classes (avec l'aide de Vincent Miele) permettant de remplacer plus facilement certaines parties du fonctionnement de DeepFaune, et ainsi utiliser Megadetector plutôt que yolov4. En effet, le programme tel qu'il a été conçu au départ ne permettait pas une telle prise de liberté et il s'avérait nécessaire d'utiliser des classes pour appeler des fonctions différentes suivant le type de détecteur utilisé et le type de données à analyser.

Le code est maintenant composé de 6 classes :
- une classe permettant d'utiliser le détecteur yolov4, permettant d'obtenir la zone de l'image où se trouve l'animal.
- une classe faisant la même chose que la précédente mais pour un détecteur Megadetector.

- une classe permettant d'utiliser le classifieur sur les images découpées.
- une classe permettant la création des séquences en fonction des dates des images
- une classe abstraite contenant la boucle principale permettant de lancer le détecteur, récupérer les résultats de ce dernier pour générer les images découpées puis lancer le classifieur pour obtenir les prédictions.
- une classe héritant de la classe abstraite adaptée à l'utilisation d'un détecteur yolo (version 4 ou inférieure).
- une classe héritant de la classe abstraite adaptée à l'utilisation d'un détecteur Megadetector (quelle que soit la version).
- une classe héritant de la classe abstraite adaptée à l'utilisation d'un détecteur yolo (version 4 ou inférieure) mais prenant des vidéos en entrée.

Megadetector étant un réseau de neurones convolutifs récurrent (RCNN) déjà entraîné (la version publique comprend un modèle dont les poids ont déjà été calculés), il n'a pas été nécessaire de refaire l'entraînement. Lors de son utilisation, les résultats sont fournis sous format json et il est ensuite possible d'utiliser ces résultats pour calculer les images découpées et les transmettre au classifieur de DeepFaune. C'est pourquoi il a fallu « repenser » la boucle principale de DeepFaune pour permettre ces différentes approches. J'ai donc imaginé été codé ces classes et installé Megadetector sur Gricad (machines sur lesquelles sera exécutée la chaîne de traitement). Au cours de mon stage, une nouvelle version de Megadetector a été publiée (Megadetector v5), et je me suis aussi chargé de la mise à niveau.

4.2 Conception de la chaîne de traitement

Je me suis ensuite, dans une seconde partie, chargé d'imaginer et mettre au point la chaîne de traitement des images d'ORCHAMP dans son ensemble (avec l'aide de Julien Renaud, Bruno Bzeznik et Vincent Miele). Voici le fonctionnement général de cette chaîne (nous verrons juste après comment fonctionne la partie d'analyse des images) :

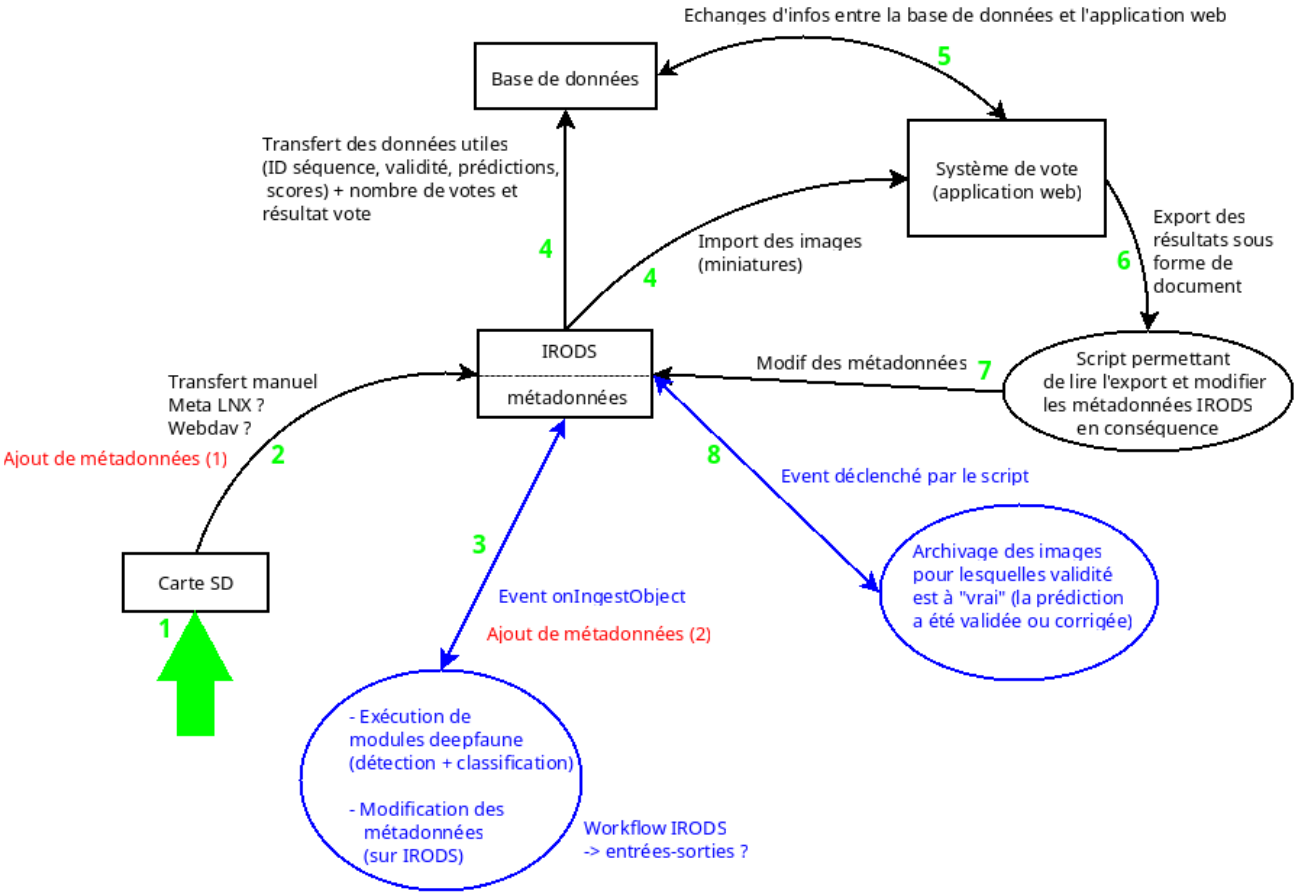

Figure 6 : schéma présentant le fonctionnement de la chaîne de traitement des images d'ORCHAMP.

Dans un premier temps, les écologues travaillant pour le projet ORCHAMP transfèrent les photos manuellement sur un système de fichiers iRODS [8] qui a l'avantage d'offrir de grandes performances dans la recherche d'informations sur les images (ce qui facilite les analyses statistiques sur ces images) via des métadonnées associées à chaque image. Ces métadonnées sont libres, ce qui nous permet de directement écrire les résultats des prédictions obtenues avec DeepFaune dans ces métadonnées. Une métadonnée est simplement une information qui n'est pas directement l'image elle-même. Tout système comprend déjà des métadonnées comme le nom d'un fichier ou sa date de création par exemple. L'avantage d'iRODS est de permettre l'ajout d'autres métadonnées comme le numéro de placette d'où la photographie a été prise, le numéro de la caméra, le numéro de séquence de l'image et l'animal ayant été détecté.

Pour l'envoi des images, nous avons décidé de passer par Meta LNX, qui est l'interface dédiée pour la communication avec le serveur iRODS. Il suffit de se

connecter et de sélectionner les images à envoyer, et il est aussi possible de créer des dossiers (appelés collections) dans son espace personnel.

Immédiatement après ce transfert dans un dossier (au choix de l'utilisateur) des images à analyser, il faut envoyer un fichier vide ayant un nom prédéfini qui déclenche un programme que j'ai été chargé de concevoir permettant d'utiliser différents modules de DeepFaune pour classer ces images et ajouter les métadonnées correspondantes sur le système iRODS (étape numéro 3 sur le schéma, que nous verrons par la suite). En effet, l'envoi de ce fichier vide déclenche un évènement permettant au programme de s'exécuter avec en entrée toutes les images venant d'être importées.

Les métadonnées dont il est question sont les suivantes :
- la date de l'image
- le nom de gradient d'où la photographie a été prise (ce qui correspond plus ou moins au nom de la montagne/chaîne de montagnes d'où la photographie a été prise)
- le numéro de placette (les placettes se trouvant tous les 200 mètres d'altitude, le numéro de la placette correspond à l'altitude d'où la photographie a été prise)
- le numéro de la caméra (étant donné que deux caméras sont installées à chaque placette, le numéro de la caméra correspond à un numéro arbitraire permettant de les différencier)
- le numéro de séquence (numéro de la séquence unique pour chaque import d'images)
- le numéro unique de séquence (numéro généré à partir du numéro de placette, de caméra, du nom de gradient et de la date de l'image afin de le différencier parmi les autres images ayant un même numéro de séquence mais ayant été importées par un autre utilisateur)
- la classe prédite sans prendre en compte le vote par séquences
- le score de la classe prédite sans prendre en compte le vote par séquences
- la classe prédite en prenant en compte le vote par séquences
- le score de la classe prédite en prenant en compte le vote par séquence
- la validité (un booléen indiquant si la classe prédite pour l'image a été vérifiée manuellement ou non).

Bruno Bzeznik s'est chargé de tout ce qui concerne l'installation d'iRODS et de pouvoir utiliser les différents bibliothèques Python sur les machines de Gricad et de faire fonctionner Tensorflow sur les cartes graphiques. Je me suis chargé de tout ce qui se rapporte à la science des données : l'adaptation et l'utilisation de l'API DeepFaune, l'installation et la mise en fonction de Megadetector, le choix des supports utilisés et la proposition de schéma pour la chaîne de traitement (qui a ensuite été discuté et adapté avec toute l'équipe), ainsi que la mise au point d'une interface ElasticSearch permettant d'effectuer des requêtes. En effet, ElasticSearch est la base de données mise en place sur Gricad qui est utilisée pour la chaîne de traitement, bien que l'import automatique des données depuis iRODS n'a pas encore été mis en place.

Julien Renaud a quant à lui commencé à travailler sur la partie web, bien qu'elle n'ait pas encore été mise en service (car la base de données n'est pas encore prête à recevoir les données depuis iRODS). Cette partie web a pour but de récupérer les résultats des prédictions effectuées sur les images depuis iRODS pour proposer une

interface permettant aux écologues d'ORCHAMP de vérifier manuellement ces prédictions et de les corriger si nécessaire. Ces vérifications manuelles auront donc lieu par campagnes régulières et uniquement les acteurs d'ORCHAMP seront autorisés à voir ces images (certaines comme celles des loups étant confidentielles).

L'archivage des données ainsi que les modifications des métadonnées suite à la vérification manuelle des résultats (parties 6, 7 et 8 du schéma) n'ont donc pas encore été mis en place.

4.3 Intégration de DeepFaune et mise en place de la base de données

Le système iRODS étant hébergé sur un serveur ayant pour but de stocker des données, il s'avère nécessaire de faire fonctionner le programme d'analyse des images sur une machine de calcul plus performante ayant un processeur rapide et une carte graphique (GPU). En effet, Megadetector est bien plus rapide s'il tourne sur GPU. Une bibliothèque iRODS existe sous Python afin de permettre des communications avec le client iRODS via des fonctions en Python, ce qui m'a permis d'écrire le programme d'analyse des images et d'écriture des métadonnées entièrement en langage Python.

Dans un premier temps, ce programme importe toutes les images qui ont été envoyées sur iRODS pour être analysées sur le nœud de calcul sur lequel le programme s'est lancé. En réalité cet import se fait par paquets de 1000 où le nouveau paquet n'est importé que si les 1000 images précédentes ont été traitées, et ce nombre est modifiable. En effet, cela permet d'éviter de surcharger le nœud de calcul avec une trop grande quantité de données.

Une fois l'import effectué, la totalité des images est traitée par un sous-processus permettant la détection des animaux dans les images par Megadetector. Le résultat est alors inscrit dans un fichier json qui contient les coordonnées des « boîtes » contenant l'animal détecté sur l'image. Cette boîte correspond à un rectangle délimitant où l'image doit être coupée pour séparer l'animal de l'environnement.

Ce fichier json est ensuite lu et les images découpées sont générées et transmises au classifieur (importé de DeepFaune) pour obtenir la prédiction. La prédiction et le score associé sont alors inscrites dans les métadonnées de l'image originale sur iRODS. Une fois que les prédictions et les scores ont été calculés pour toutes les images, nous arrivons à la phase de vote par séquence, une fois de plus importé d'un module de DeepFaune.

Ces métadonnées sont ensuite importées dans une base de données ElasticSearch pour permettre les requêtes.

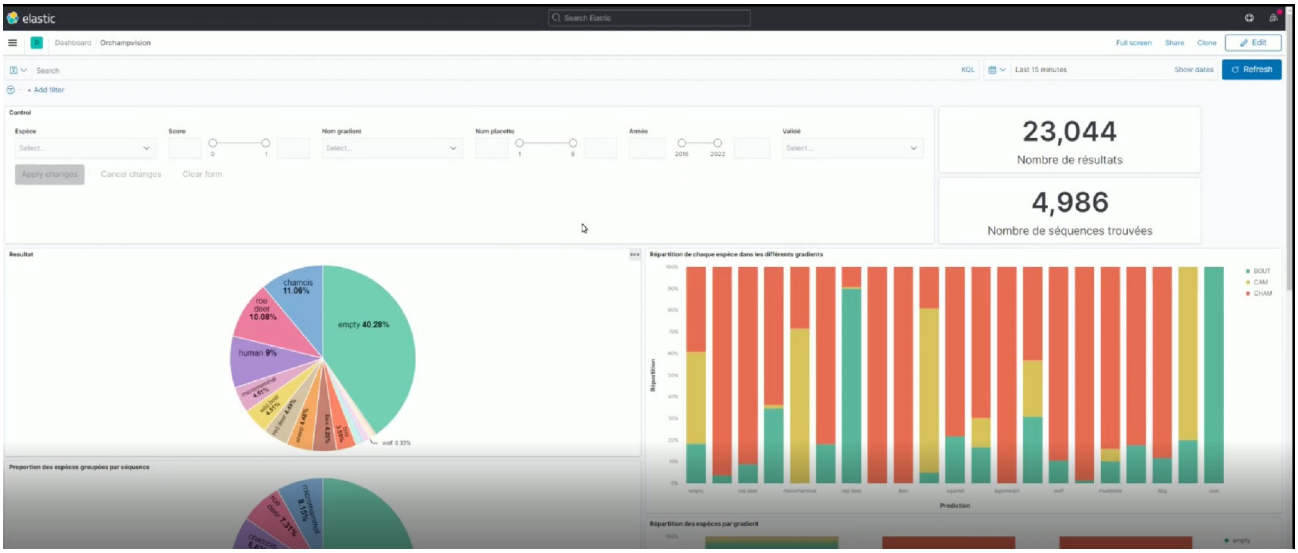

Figure 7 : aperçu de l'interface permettant d'effectuer les requêtes

Cette interface ElasticSearch a pour but de récupérer les métadonnées des images sur iRODS (fonctionnement qui n'est pas encore automatique mais qui est faisable manuellement) et permet d'effectuer des requêtes préconçues (que j'ai conçues à partir de l'interface de développement de Kibana, qui est le nom de l'instance d'ElasticSearch mise en place sur Gricad) sur ces métadonnées importées en csv et affiche le résultat obtenu sous forme de graphiques. Il est aussi possible d'avoir ces résultats sous forme de csv pour des analyses statistiques plus poussées. Il est aussi possible d'appliquer différents filtres que j'ai mis en place pour des analyses plus spécifiques : ces filtres permettent de sélectionner les espèces à l'étude, les indices de confiance (au cas où l'utilisateur souhaite uniquement travailler sur des images dont l'indice de confiance est élevé, afin de réduire les chances que la prédiction soit mauvaise), le nom de gradient à l'étude, le numéro de placette (pour étudier uniquement les images provenant de certaines altitudes), l'année à laquelle la photographie a été prise et la validité (pour le cas où l'utilisateur souhaiterait uniquement travailler sur des images dont la prédiction a été vérifiée manuellement). Ce système de filtre est paramétrable facilement depuis l'interface de Kibana et permet donc de sélectionner uniquement les lignes de la table qui vérifient toutes les condition (une à une).

L'avantage de ElasticSearch est qu'il permet un accès très rapide aux données, au prix de modifications coûteuses. Étant donné que les campagnes de validation des prédictions et les récupérations des images depuis les cartes mémoire auront lieu à des moments prédéfinis et seulement une à deux fois par an, ElasticSearch s'avère être un choix intéressant, d'autant plus que l'accès rapide aux données permet des analyses plus rapides, sachant que ces données sont très nombreuses. De plus, la possibilité de créer une interface graphique simple d'utilisation est aussi un atout majeur.

5. Perspectives

La chaîne de traitement des images d'ORCHAMP est fonctionnelle pour ses fonctions principales (sauf l'application de vote), ce qui constitue une grande avancée. Mais il reste cependant quelques points d'amélioration envisagés pour la suite. Tout d'abord, Megadetector v5 utilise PyTorch et non Tensorflow, tandis que le classifieur de DeepFaune utilise Tensorflow. Cela implique donc que depuis le passage à Megadetector v5, les deux bibliothèques doivent être importées et mises en place sur le nœud de calcul. Cela représente un risque supplémentaire de bugs lors de mises à jour des bibliothèques ou des outils utilisés et représente des imports inutiles de fonctions. Une première perspective serait de créer un classifieur sous PyTorch pour DeepFaune, ainsi seul PyTorch serait utilisé pour la chaîne de traitement.

Une autre avancée prévue est bien entendu de terminer l'application de vote afin de valider les prédictions manuellement, et donc l'archivage des images.

Sur chaque placette à l'étude est installé un micro pour effectuer des études acoustiques. Ajouter ces études acoustiques à la chaîne de traitement est aussi une option envisagée, afin que toutes les analyses soient réalisées par un même programme pouvant traiter les deux types de données (images et audio).

6. Conclusion

L'objectif de mon stage étant la mise au point de la chaîne de traitement des images de l'observatoire ORCHAMP, j'ai pu découvrir et participer à la mise en place d'un outil d'intelligence artificielle au service de la recherche. En Effet, DeepFaune étant un projet de grande envergure dont le but est de servir le plus grand nombre, je suis très heureux d'avoir pu y contribuer. J'ai pu tester des modèles afin d'essayer d'améliorer celui qui avait été mis en place, chercher ce qui se fait de mieux dans le domaine de la vision par ordinateur (principalement via les compétitions comme iWildCam) afin de donner mon point de vue et de proposer des axes d'amélioration. J'ai pu participer activement au projet en ayant moi-même mis en place les différentes classes permettant maintenant d'utiliser Megadetector comme détecteur pour DeepFaune, ce qui implique de bien meilleures performances pour la chaîne de traitement.

Le fait d'avoir été libre quant à l'utilisation des outils pour la chaîne de traitement m'a donné une vraie responsabilité et m'a permis de confronter différents points de vue, de défendre mes opinions et chercher ce que j'estimais être ce qu'il y a de mieux à mettre en place, et d'apprendre à m'intéresser aux autres domaines (principalement l'écologie) afin d'apporter des arguments pertinents et de mettre en place un outil qui

soit adapté à la demande. J'ai tout particulièrement apprécié le fait de continuellement chercher à m'adapter aux demandes et aux nouveautés apportées à DeepFaune en parallèle ainsi qu'aux outils envisagés. En effet, un point crucial de mon stage qui m'a posé des difficultés était de devoir faire en sorte que les différents outils puissent fonctionner ensemble correctement. Cela me demandait de comprendre exactement comment ils fonctionnent pour adapter DeepFaune et permettre plus de libertés sans pour autant compromettre son fonctionnement. L'utilisation de git dans un contexte plus poussé (et plus professionnel) via les issues et les branches fut aussi un pas à franchir, n'ayant pas l'habitude d'utiliser git de façon régulière.

Ce stage m'a beaucoup appris, tant sur le plan humain et professionnel via l'utilisation de git et les nombreuses réunions que sur le plan technique, étant continuellement dans une recherche de créer l'outil le plus performant possible.

REMERCIEMENTS

Je tiens à remercier tous ceux que j'ai eu la chance de côtoyer durant mon stage, et tout particulièrement Vincent Miele qui a été un encadrant très présent et à l'écoute, et qui m'a donné beaucoup de très précieux conseils.

RESSOURCES

[1] MARTINEZ-ALMOYNA, Camille, THUILLER, Wilfried, CHALMANDRIER, Loïc, *et al.* Multi-trophic β-diversity mediates the effect of environmental gradients on the turnover of multiple ecosystem functions. *Functional Ecology*, 2019, vol. 33, no 10, p. 2053-2064.
Pour plus de publications liées à l'observatoire ORCHAMP :
https://orchamp.osug.fr/PublicationsAjour

[2] MIELE, Vincent, DRAY, Stéphane, et GIMENEZ, Olivier. Images, écologie et deep learning. *Regards sur la biodiversité*, 2021.

[3] RIGOUDY, Noa, BENYOUB, Abdelbaki, BESNARD, Aurelien, *et al.* The DeepFaune initiative: a collaborative effort towards the automatic identification of the French fauna in camera-trap images. *bioRxiv*, 2022.

[4] https://plmlab.math.cnrs.fr/deepfaune/software/-/tree/master

[5] BOCHKOVSKIY, Alexey, WANG, Chien-Yao, et LIAO, Hong-Yuan Mark. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[6] https://github.com/microsoft/CameraTraps/blob/main/megadetector.md

[7] TAN, Mingxing et LE, Quoc. Efficientnet: Rethinking model scaling for convolutional neural networks. In : *International conference on machine learning*. PMLR, 2019. p. 6105-6114.
Lien pour les résultats de la compétition iWildCam
https://www.kaggle.com/c/iwildcam2021-fgvc8/

[8] RAJASEKAR, Arcot, MOORE, Reagan, HOU, Chien-Yi, *et al.* iRODS primer: integrated rule-oriented data system. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 2010, vol. 2, no 1, p. 1-143.