Projet BeesForLife
Rapport

Master 2 Informatique

Hugo Gaquere Maxime Bonnal

27 mars 2023

Table des matières

1 Introduction

Dans le cadre du Projet de Fin d'Etudes, nous avons pu participer à la réalisation de BeesForLife, un projet universitaire se présentant comme une application de détection de nids de frelons asiatiques à partir de données issues de drones. Ce projet est réalisé pour le compte de M.Willaert, pilote de drone et membre de l'association BeesForLife [2], et encadré par Pascal Desbarats, Marie Economidès et Alexis Hoffmann qui en était en charge de la version précédente de l'application lors de son stage de fin d'études. Pour plus de précision, le but de cette application est donc la détection de nids de frelons asiatiques sur des images par calculs d'inférences à l'aide de modèles de réseaux de neurones profonds. Le développement du projet BeesForLife ayant débuté il y a quelques années déjà, l'application est désormais fonctionnelle et nécessite donc des améliorations. Ainsi les voies d'améliorations énoncées dans le sujet étaient les suivantes :

1. Modifier la GUI de l'application pour y ajouter l'affichage d'informations permettant l'analyse (notamment pour faciliter la localisation du nid en temps réel).

2. Proposer une méthode d'analyse et d'affichage de la température à partir des images infrarouges conjointement à la détection du réseau utilisé.

3. Tester d'autres réseaux en pseudo-temps réel (Entraînement sur PlaFRIM ou Jean Zay, adapter l'architecture du dataset)

A partir de ces tâches et des diverses discussions avec les encadrants du projet, nous avons pu estimer les besoins nécessaires et tenter de les implémenter au mieux. Il existe néanmoins des contraintes dans la réalisation. En effet, l'application doit fonctionner sous MacOS et le traitement des données doit s'effectuer sur CPU afin d'être utilisable sur l'ordinateur de M.Willaert. De plus, n'étant pas un professionnel de l'informatique, l'utilisation de l'application doit rester accessible et ergonomique.

2 Etat de l'art

2.1 Analyse de l'existant

Nous avons récupéré la dernière version en date du projet, une application déjà fonctionnelle en Python 3.8, s'installant dans un environnement Anaconda[1]. Cette application se présente sous la forme d'une interface graphique utilisant le framework QT[4].

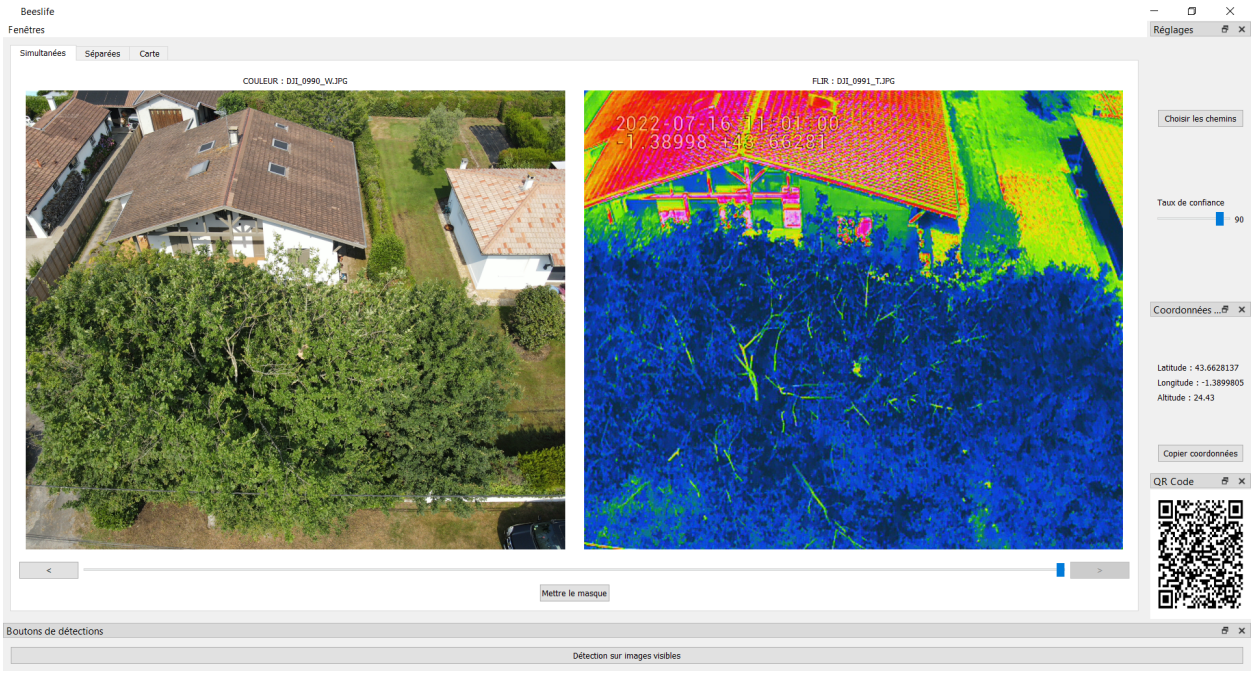

FIGURE 1 – Capture de l'application dans sa v4 dans l'onglet "Simultanées"

Cette interface, visible sur la figure 1, comporte 3 onglets, le premier "simultanées" permet de visionner les images par paires entre les images dites couleurs et les infrarouges FLIR. Le second onglet "séparées" permet de visionner les albums d'images couleurs et infrarouges indépendemments. Le dernier, "Carte" se présente sous la forme d'une carte obtenue à l'aide la bibliothèque `folium` [6] donnant la position approximée du nid en utilisant les coordonnées GPS du drone lors de la prise des photos.

A cela s'ajoute plusieurs widgets déplaçables. Le premier bouton sur la colonne de droite permet de choisir le chemin d'accès des images. Le second est un curseur permettant de modifier le seuil du taux de confiance lors de la détection. Viens ensuite les coordonnées GPS des photographies ainsi qu'un QRCode permettant de les récupérer afin d'afficher la position sur Google Maps. Pour finir nous avons en bas un bouton permettant de lancer la détection des nids de frelons sur l'ensemble des images couleurs permettant d'obtenir de nouvelles images présentant un masque autour de ceux-ci s'ils sont détectés. Ces images résultantes sont affichables en appuyant sur le bouton "Mettre le masque".

L'environnement anaconda est déployé sur la machine de l'utilisateur au moyen d'un fichier yaml contenant la liste des paquets et dépendances nécessaires au bon fonctionnement de l'application. L'installation de celui-ci se fait simplement en utilisant la commande `conda env create -file environment.yml`

2.2 Critique de l'existant

Cette version présente plusieurs problèmes et voies d'améliorations. Le premier désagrément se trouve au niveau de la latence qui survient lors du chargement des images. En effet, les images se chargent uniquement lorsqu'on les visionne, ce qui cause de forts ralentissements lors du parcours. Ce sont des images de 8000x6000 pixels qui font en moyenne 14Mo, lourde

à charger sur les configurations peu puissantes comme celle possédée par le client.
Hormis pour le visionnage, les images infrarouges ne sont utilisées dans aucune fonctionnalité de l'application.
Pour ce qui est de l'interface, plusieurs fonctionnalités sont sous-exploitées ou peu ergonomiques. On a tout d'abord l'onglet "Séparées" qui s'avère être peu utile hormis pour minimiser les ralentissements en permettant de parcourir séparément les images couleurs des images infrarouges.
Sur l'onglet "Carte", la position ne s'affiche qu'après avoir procéder à une détection, et nous permet seulement de visualiser la position moyenne de tous les nids détectés.
Au niveau de l'affichage des images, on peut regretter l'absence de zoom afin de mieux voir les nids, qui sont généralement de petite taille. De plus, le ratio des images n'est pas conservé lors de l'affichage, ce qu'il leur fait perdre en qualité.
L'interface souffre de problèmes de dimension lorsque l'on modifie la taille de la fenêtre, notamment pour les images et le QRCode qui devient illisible.
L'utilisation du bouton "choisir les chemins" est inutile étant donné qu'il est nécessaire d'utiliser un chemin prédéfini dans le code pour charger les images.
Le déploiement des mises à jour de l'application chez le client est peu ergonomique pour un usager occasionnel en programmation informatique car il nécessite de passer par des commandes git dans un terminal, ce qui nécessite généralement la présence d'un membre de l'équipe de projet.
On peut aussi regretter une architecture logicielle assez compliquée à maintenir en raison d'un mélange entre la vue et de la logique.

2.3 Analyse de l'existant de la détection

L'application utilise comme modèle de détection Yolov7 Tiny, en raison de son efficacité dans la détection d'objets et sa vitesse, permettant possiblement le temps réel [16]. Cette possibilité est à envisager car l'application pourrait être amenée à traiter en temps réel les données reçues du drone à l'avenir. La prédiction est obtenue en appliquant un calcul d'inférences sur l'image en la partitionnant en sous-images de taille 1000x1000. Depuis la dernière version du projet, Ultralytics a publié en Janvier 2023 Yolov8 [8] qui apparaît comme étant plus performant que Yolov7 [3].

3 User Stories

A la suite de discussions avec les parties prenantes du projet, plusieurs User Stories ont pu être extraites :

1. En tant qu'utilisateur, je souhaite détecter rapidement et précisément les nids de frelons asiatiques sur les images à l'aide d'un réseau de neurones profond, afin d'obtenir les coordonnées géographiques du nid.

   Cette User story constitue l'objectif premier du projet, c'est-à-dire améliorer les performances de la détection afin de réduire le plus possible le nombre de faux positifs. Les faux positifs dans notre cas sont les détections d'éléments qui se révèlent ne pas être

des nids, ce qui réduit l'intérêt de l'usage de l'application si une intervention humaine est nécessaire pour certifier la présence réelle d'un nid sur une image. Pour se faire, nous avons décidé de mettre à jour le code et les modèles du réseau de neurones pour utiliser Yolov8 d'Ultralytics [8], ce qui se fait de mieux actuellement pour la détection sur des jeux données d'images.

2. En tant qu'utilisateur, je souhaite utiliser mes images infrarouge, afin d'améliorer la précision de la détection des nids de frelons asiatiques.

   Le drone de M.Willaert est aussi équipé d'une caméra infrarouge, mais la position et le moment de la prise diffèrent de la caméra standard, en plus de la différence de résolution, avec 8000x6000 pour la caméra standard contre 640x512 pour l'infrarouge. Ces nombreuses différences rendent l'utilisation des images infrarouges complexes dans l'application, notamment au niveau de l'entraînement du réseau de neurones, où les couleurs, étant calculées selon la chaleur présente sur la photo, peuvent changer totalement en fonction de l'angle de la photo en cas de présence d'élément très chaud comme le soleil par exemple. Ces fortes différences de couleurs entre les images prises parfois dans des angles proches peuvent fausser la détection en cas de reconnaissance par couleur, ce qui rend actuellement leur utilisation préjudiciable. Il est tout de même dommage de ne pas pousser leur exploitation car elles peuvent se révéler utile en combinaison des images couleurs, notamment pour la vérification de la présence d'un nid à une position par sa température. Nous avons mis en place la première étape de ce processus en implémentant une fonctionnalité permettant de lire la température du pixel sur lequel le curseur de la souris est posé au moyen de la bibliothèque thermal_base[5].

3. En tant qu'utilisateur, je souhaite pouvoir partager mes images avec le QRcode afin de faciliter le partage de coordonnées.

   Chaque photographie est associée aux coordonnées du drone lors de la prise, il est ainsi pratique de pouvoir partager la position directement avec l'image si un nid a été détecté sur celle-ci au moyen du QRcode déjà généré par l'application. Pour se faire, nous avons mis en place un nouveau système d'export des images résultantes de la détection de nids.

4. En temps qu'utilisateur, je souhaite pouvoir parcourir et zoomer sur mes photos rapidement.

   Les albums visionnés sont composés d'un grand nombre d'images en haute résolution ce qui rend leurs chargements assez lent, surtout sur les machines peu puissantes comme celle du client, faisant perdre en fluidité à l'application. De plus, les nids sont souvent petits et cachés par des éléments de l'image, ce qui peut amener à la nécessité de pouvoir zoomer. Pour répondre aux attentes, nous avons revu l'affichage des images afin de pouvoir rétablir le ratio des images, jusqu'alors légèrement déformées, et d'ajouter la possibilité de zoomer et de se déplacer sur l'image. Pour optimiser le parcours, nous

avons mis en place un préchargement des images.

5. En tant qu'utilisateur, je souhaite utiliser une interface mise à jour.

   Dans sa v4, l'interface est fonctionnelle mais manque d'ergonomie et souffre de problèmes d'affichage comme la préservation des dimensions des éléments de la fenêtre. La présence des onglet "Séparées" et "Carte" semblent peu utile, et la présence des widgets sur la droite est assez envahissante pour souvent une utilité moindre. De plus l'architecture logicielle de l'interface graphique mélange la vue et la mise à jour des données, ce qui complique l'implémentation et la maintenance d'autres fonctionnalités. Nous avons mis en place une nouvelle interface avec une architecture MVC et mis a jour les différentes fonctionnalités déjà présentes.

6. En tant qu'utilisateur, je souhaite pouvoir installer et mettre à jour le logiciel facilement, afin de gagner du temps.

   Le client n'étant pas un professionnel de l'informatique, l'utilisation de l'application se doit d'être simple d'accès, néanmoins le partage des mises à jour peut s'avérer complexe en raison de l'utilisation de commandes git entre autres, ce qui oblige la mobilisation d'un membre de l'équipe pour assister le déploiement. L'idée aurait été d'implémenter un moyen de déployer les mises à jour sur l'ordinateur du client à l'aide d'un script ou d'un exécutable, mais cette tache étant assez secondaire, nous n'avons pas eu le temps de la traiter.

Ces user stories sont ordonnées par ordre d'importance, avec notamment les deux premières qui traitent de l'amélioration de la détection des nids et l'utilisation de l'infrarouge qui constitue le coeur du projet.

De ces users stories ont découlé les différentes tâches que nous avons traitées au fur et à mesure de l'avancement du projet.

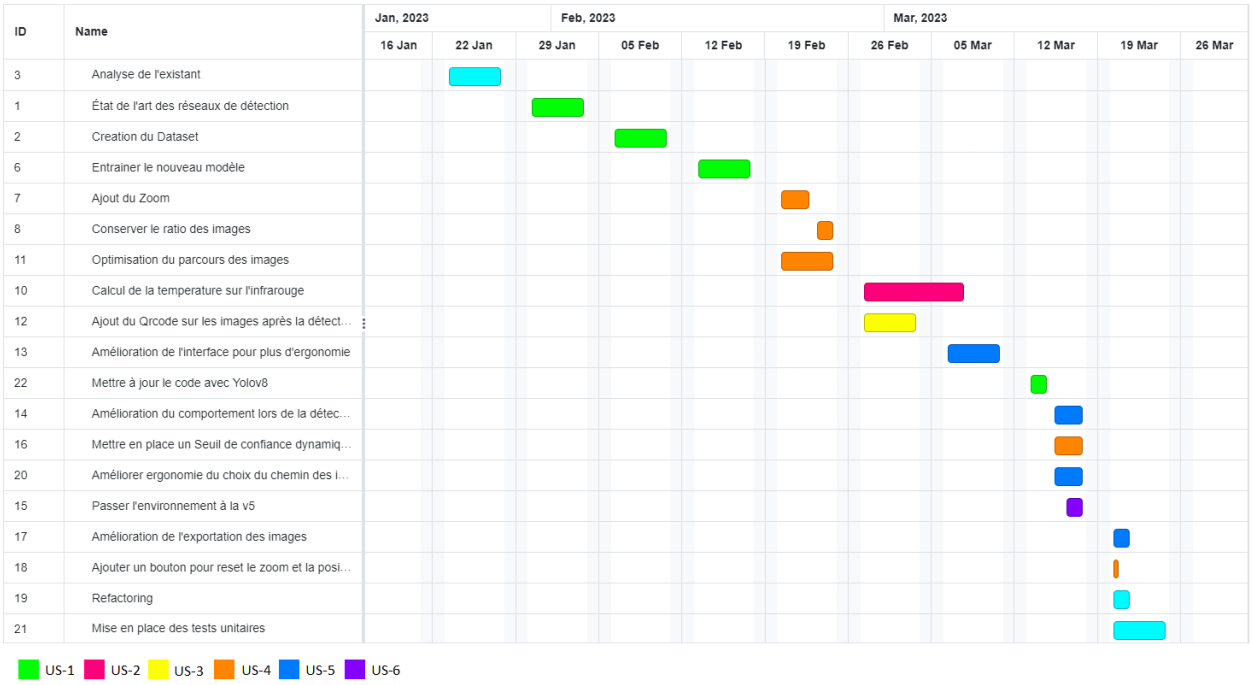

FIGURE 2 – Diagramme de Gantt du projet

On peut voir sur cette figure 2 le diagramme de Gantt représentant les différentes tâches traitées au long du projet avec un code couleur en fonction des users stories auxquelles elles sont associées.

4 Implémentation

4.1 Travaux réalisés

Nous allons à présent détailler notre avancée dans le projet et ce que nous avons réussi à implémenter.

Afin de gagner en performance sur la détection des nids, avec pour principaux objectifs d'augmenter la précision de la détection et surtout de réduire le nombre de faux-positifs, nous avons tout d'abord mis à jour les modèles de détection et créé un nouveau jeu de données que nous expliquerons plus en détails dans la partie Résultats à la page 13.

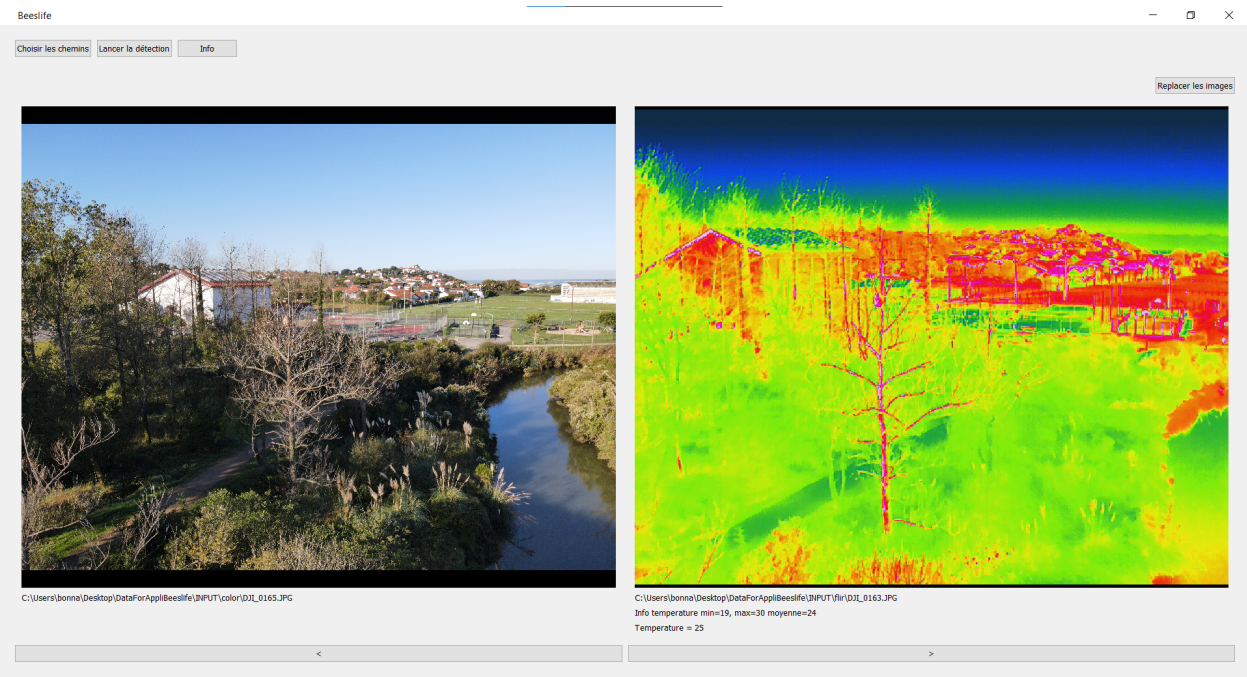

FIGURE 3 – Capture de l'interface au lancement de l'application

On a ici figure 3, la nouvelle interface de l'application. On peut tout d'abord remarquer le retrait des différents onglets et des widgets présents sur la droite au profit de boutons présent en haut.
Le premier bouton "choisir le chemin" reprend le fonctionnement de l'ancienne version à ceci près qu'il permet désormais de charger correctement un autre dossier d'images après avoir procédé à une détection sans avoir à fermer l'application.
On peut aussi observer une amélioration au niveau de la latence lors du parcours des images car nous avons mis en place un système de préchargement des images en arrière plan par lots de 5 ainsi qu'un déchargement progressif afin de ne pas surcharger la RAM.
Nous avons modifié la méthode d'affichage des images en utilisant la bibliothèque PyQtGraph [12] qui nous permet de zoomer sur les images. A l'aide de la bibliothèque Thermal_base [5], nous pouvons analyser l'image FLIR pour afficher la température minimale, maximale, moyenne ainsi que celle au niveau du curseur de la souris sur les images infrarouges, comme on peut voir figure 4.

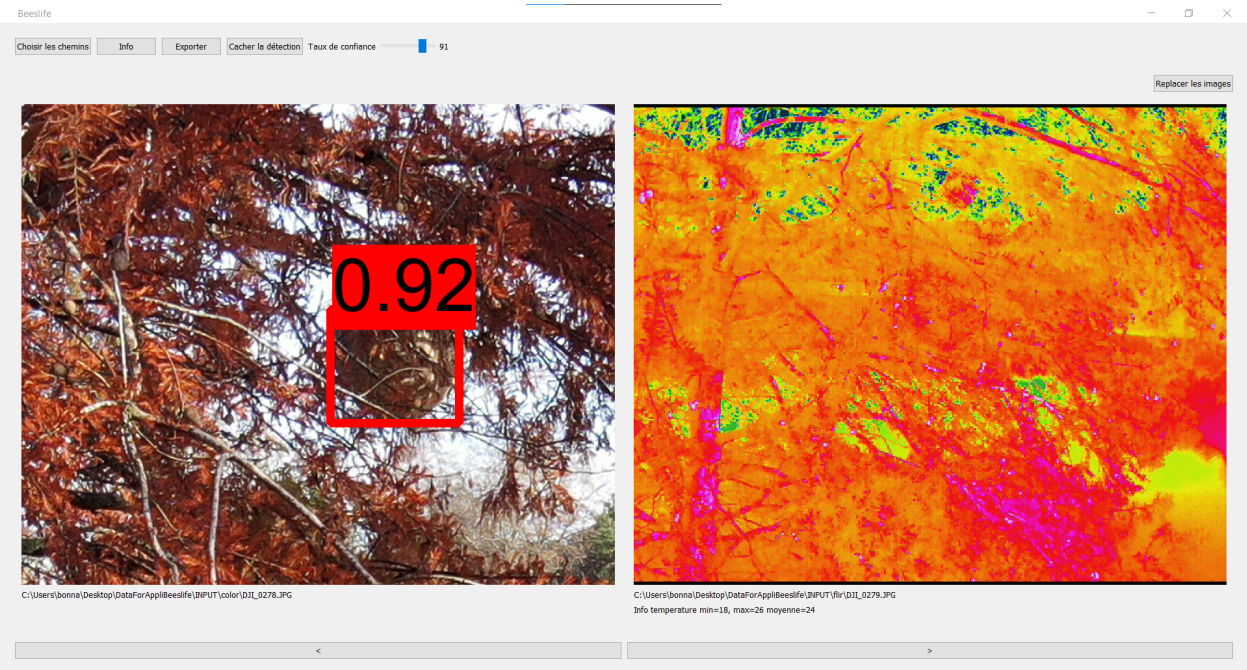

FIGURE 4 – Capture de l'interface après la détection en zoomant sur un nid

Désormais, les informations auparavant contenues dans les widgets sont affichables dans une fenêtre ouvrable à l'aide du bouton "Info" (figure 5), permettant d'afficher les coordonnées et de placer correctement la position du drone sur la carte contrairement à l'ancien onglet "Carte" qui plaçait une position moyenne à partir de l'ensemble des coordonnées des photographies.

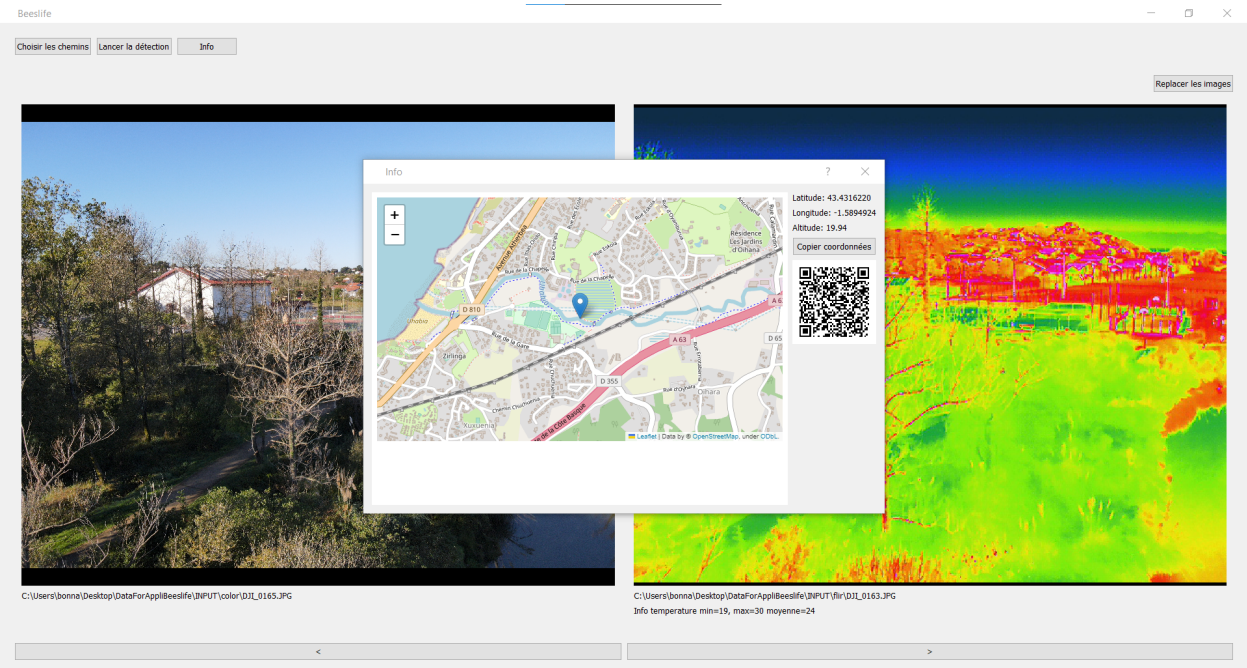

FIGURE 5 – Capture de l'interface au déclenchement du bouton "Info"

Nous avons résolu des problèmes de latence et de gel de la fenêtre de l'application lors

de la détection des nids. Une fois la détection effectuée, le curseur permet désormais de filtrer l'affichage des masques en fonction du taux de confiance de la détection de manière dynamique comme on peut le voir figure 6. On peut voir aussi l'ajout du bouton "Exporter" qui permet de sauvegarder dans le dossier de son choix les images résultantes, en donnant la possibilité d'y ajouter le QRcode renvoyant la position du drone sur l'image, en s'assurant qu'il ne recouvre pas une potentielle détection.

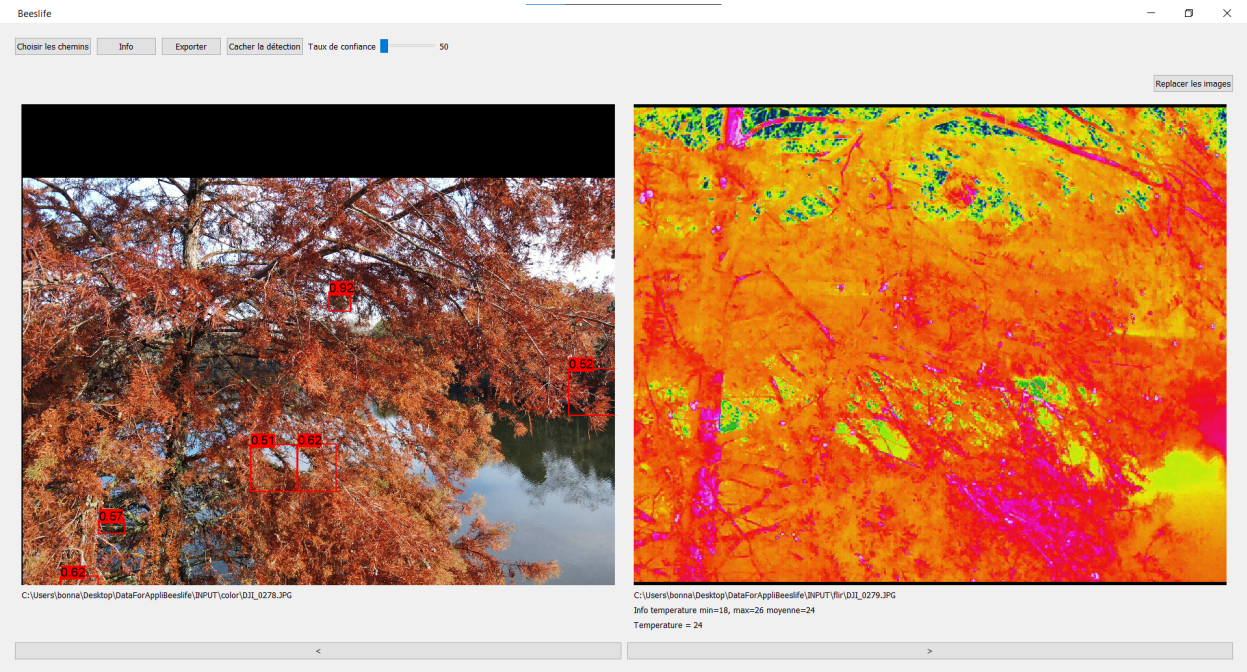

FIGURE 6 – Capture de l'interface après la détection en baissant le taux de confiance

4.2 Architecture du projet

En raison de certains problèmes de modularités, nous avons choisi de revoir l'architecture logicielle du projet en ne gardant seulement que quelques classes.

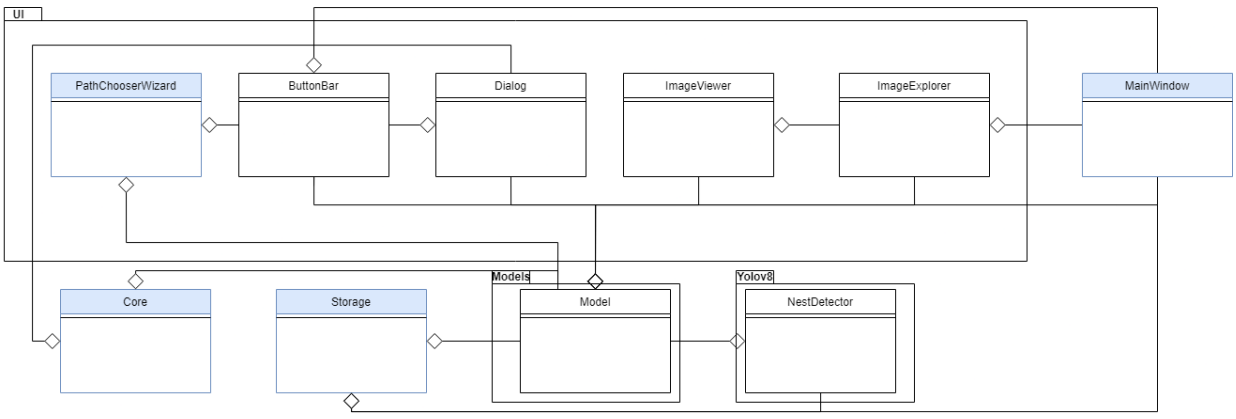

FIGURE 7 – Diagramme de classes simplifié de l'application

Les classes en bleu sur la figure 7 sont celles issues de la version précédente.

L'architecture a été retravaillée afin de respecter une sorte de modèle MVC modifié, qui nous semble particulièrement adapté au projet en raison de l'interface graphique faisant office de Vue et de Controleur et des différentes fonctionnalités et données liées à celle-ci pouvant être traitées comme modèles. Nous avons gardé intact le module **Storage** utilisé pour la gestion du stockage des images et **Core** servant notamment pour la gestion des coordonnées GPS avec la classe `GPSCoordinates` et le chargement des images depuis un dossier avec `ImagesLoader`. La classe **MainWindow** a été revu et épuré en raison de la refonte de l'interface. Désormais elle ne sert plus qu'à mettre en place la fenêtre et appeler les différents layouts.
Les différents éléments graphiques sont pour la plupart hérités de classes de pyQT5 [11] .

4.2.1 Models

Le module **Models** va contrôler l'état et la logique du logiciel. Si un état change, il va alors émettre un signal qui sera intercepté par les composants graphiques qui mettront à jour l'interface. Ces différents signaux sont émis notamment lors des interactions de l'utilisateur avec les différents boutons afin de les déclencher, comme par exemple lors du lancement de la détection.

4.2.2 Ui

Ce package contient les classes en charge de la gestion de l'interface graphique de l'application, avec différents modules qui jouent à la fois le rôle de vue et de contrôleur. Elle contient notamment la classe `ImageViewer` et `ImageExplorer` qui ont pour rôle de mettre en place les éléments en rapport avec l'affichage des images et de mettre à jour l'interface suite à la réception d'un signal. La classe `ButtonsBar` met en place la barre de boutons en haut de l'application et leurs signaux de déclenchement. Le module `dialogs` va lui implémenter les différentes fonctions déclenchées en fonction du bouton pressé. On retrouve aussi la classe `PathChooserWizard` permettant le choix des chemins des dossiers, que nous avons repris de la version précédente.

4.2.3 Yolov8_nest

La classe **NestDetector** contient la logique en rapport avec la prédiction des nids. Le calcul de l'inférence se fait désormais en utilisant la bibliothèque Sahi [7], permettant de découper l'image en blocs et d'y calculer l'inférence en lui donnant un modèle, Yolov8 dans notre cas.

4.3 Tests

Coverage report: 66%

coverage.py v7.2.2, created at 2023-03-26 23:01 +0200

Module	statements ↓	missing	excluded	coverage
src/BflQt/models/models.py	210	59	0	72%
src/BflQt/ui/dialogs.py	139	105	0	24%
src/BflQt/ui/wizards.py	120	77	0	36%
src/BflQt/ui/Image_viewer.py	117	34	0	71%
src/BflQt/ui/buttons_bar.py	102	22	0	78%
src/BflQt/Core.py	98	0	0	100%
src/BflQt/ui/image_explorer.py	59	4	0	93%
src/BflQt/Storage.py	55	0	0	100%
src/BflQt/yolov8_nest/nest_detector.py	55	29	0	47%
src/BflQt/MainWindow.py	27	0	0	100%
src/BflQt/__init__.py	4	1	0	75%
src/BflQt/models/__init__.py	0	0	0	100%
src/BflQt/ui/__init__.py	0	0	0	100%
src/BflQt/yolov8_nest/__init__.py	0	0	0	100%
Total	**986**	**331**	**0**	**66%**

FIGURE 8 – Couverture des tests

Comme présenté sur la figure 8, le taux de couverture de nos tests est de 66%. En effet nous avons manqué de temps, cependant les principaux composants de l'application comme **Storage** qui gère le chargement et la manipulation des fichiers, sont couverts à 100%.

4.4 Suivi de Projet

Pour nous accompagner dans ce projet, nous avons pu organiser des réunions avec M.Desbarats et M.Hoffmann tout les lundi matin jusqu'au vacances de Février afin de se familiariser avec le fonctionnement de l'application et de bien comprendre les attentes et objectifs du projet. A la suite des vacances de Février, la fréquence des réunions a été plus sporadique, au gré des emplois du temps, en raison de la présence de M.Desbarats à la fois en temps que responsable de projet et responsable du TD de MOCPI, ce qui lui permettait de suivre l'avancée du projet chaque semaine et de nous orienter dans nos travaux.
Nous avons aussi pu nous entretenir directement avec le client du projet, M.Willaert, lors d'une réunion en visioconférence au cours de laquelle nous avons pu discuter de ses habitudes d'utilisation, ses attentes ainsi que des défauts d'ergonomie afin de pouvoir mieux estimer les besoins.

Pour ce qui est de l'organisation, nous avons utilisé un Kanban sur Trello afin de lister les tâches et leurs avancées le long du projet. Pour la planification du travail, nous nous sommes inspirés des sprints des méthodes agile comme Scrum. Ces sprints étaient constitués de tâches, extraites des Users Stories, que nous estimions nécessaires et réalisables en une

semaine. Pour la planification, nous nous retrouvions tout les mardis au TD de MOCPI afin de préparer le prochain sprint. De plus, nous profitions de ce créneau pour faire la revue de code des différentes avancées de la semaine.

Pour gérer le versionning et les avancées du code, nous avons utilisé un repository gitlab en clonant la version précédente du projet.

5 Entraînement et résultats

5.1 Jeu de données

5.1.1 Jeu de données existant

Le dataset existant, qui a servi à entrainer le modele Yolov7 [16], est composé de 2219 images de nids de frelons asiatiques, labelisé par Hoffmann Alexis sur labelme [9]. Le jeu de données ne contient pas d'images thermiques.

5.1.2 Nouveau jeu de données

Notre objectif avec l'intégration du nouveau modéle Yolov8 [8], est d'augmenter la précision du modèle et de diminuer les faux positifs. Pour cela nous avons amélioré et préparé le jeu de données en 3 étapes :

1. Ajout d'images d'arrière-plan

   Premièrement, nous avons ajouté des images d'arrière-plan (background images) au jeu de données. Ce sont des images sans nid, afin de réduire les faux positifs (FP). Il est recommandé [14] d'ajouter entre 0 et 10% d'images d'arrière-plan pour réduire le nombre de faux positifs. Nous avons choisi d'ajouter au jeu de données 10% d'images d'arrière-plan. Il est désormais composé de 2447 images.

2. Préparation des données

   Puis pour entraîner le modèle sur des images de 640x640 pixels, nous devions préparer les images. Pour cela nous avons découpé chaque image autour du nid. La distance entre le haut gauche de la bounding-box du nid, et le haut gauche de l'image est choisi aléatoirement. Ainsi la position du nid dans l'image est variable, permettant d'augmenter la diversité du jeu de données.

3. Data Augmentation

   Le jeu de donnée est réparti de la façon suivante. Entraînement : 88% - Validation : 8% - Test : 4%.

   En dernier, nous avons utilisé des techniques de data augmentation pour augmenter la partie entraînement de notre jeu de données. Celle-ci est décrite dans le tableau 1. Nous multiplions par 3 la taille de notre jeu de données d'entraînement. Le jeu de données final est composé de 5 883 images.

Augmentation	Commentaire
Flip horizontal	50% de probabilité
Noir et blanc	15% de probabilité
Recadrage	entre 0% et 20% de l'image
Luminosité	entre -25% et +25%
Exposition	entre -25% et +25%

TABLE 1 – Pipeline de data augmentation

5.2 Modèle

Ultralytics propose différents modèles pré-entraînés sur le jeu de données COCO [15], sur lequel nous pouvons faire du transfer-learning.

Nous avons sélectionné le modèle YOLOv8n présenté dans le tableau 2, pour sa rapidité d'inférence sur CPU. De plus, étant donné que nous n'avons qu'un seul objet à détecter, un modèle plus gros n'est pas nécessaire.

Model	Size (pixels)	mAP 50-95	Speed CPU ONNX (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	3.2	8.7

TABLE 2 – Caractéristiques du modèle YOLOv8n

5.3 Résultats

Model	Size (pixels)	Precision	Recall	mAP 50	mAP 50-95
YOLOv7 Tiny	640	0.9651	0.9369	0.9753	0.7765
YOLOv8n	640	0.981	0.955	0.981	0.863

TABLE 3 – Comparaison des modèles

Après l'entraînement, le modèle a été convertit au format ONNX, car selon la documentation d'Ultralitycs [13], le format ONNX permet d'augmenter par 3 la vitesse d'inférence sur CPU. Il est également intéressant de noter pour la suite du projet, que le format TensorRT permet de multiplier par 5 la vitesse d'inférence sur GPU.

La table 3 présente les résultats obtenus pour l'ancien modèle YOLOv7 Tiny et le nouveau modèle YOLOv8n. Nous remarquons que le modèle YOLOv8n est plus performant en tout point, notamment pour la mesure mAP 50-95 pour laquelle nous observons une augmentation de presque 10%.

Puis, nous avons comparé le temps d'exécution complet (découpage des images et détection) sur 200 images, table 4. Nous remarquons que notre temps d'exécution est deux fois plus lent. Cela est expliqué par l'utilisation d'approches différentes pour le pré-traitement des images.

En effet, les images à analyser sont de l'ordre 8000x6000 pixels, alors que les modèles ont été entraînée sur des images de 640x640 pixels. L'ancienne intégration se déroulait en deux

Model	Temps d'inférence (200 images sur CPU)
YOLOv7 Tiny	16 min
YOLOv8n	30 min

TABLE 4 – Comparaison temps d'inférence

étapes : découpage en sous-image de 1000x1000, puis chaque sous-images est redimensionné en 640x640 avant d'être envoyé au modèle. Cette approche générait pour 1 image, 48 sous-images à traiter par le modèle.

Dans l'objectif d'améliorer la précision de la détection, nous avons fait le choix de découper les images en sous-images de 640x640 pixels. Chaque sous-images est décalé de 20% en haut à gauche, pour couvrir les potentiels nids qui se trouvent à l'intersection de différentes sous-images. Notre approche génère 192 sous-images, soit 4 fois plus que l'ancienne approche.

5.4 Perspectives

Au terme de ce projet, nous n'avons pas pu implémenter toutes les fonctionnalités prévues, en raison d'un manque de temps. Il reste alors des perspectives d'améliorations pour le futur de l'application.
En effet, l'utilisation de l'infrarouge est encore peu poussée et pourrait à l'avenir être bien plus utile. L'idée serait de réussir à replacer le masque sur l'image infrarouge et, en calculant la température, certifier ou non la présence d'un nid à cette position.
Pour ce qui est de la détection, nous avons été limités par la puissance des machines à notre disposition pour l'entraînement des modèles. Il serait possible à l'avenir d'utiliser de plus puissantes machines comme celles de PlaFRIM [10] afin d'améliorer les performances.
Nous n'avons pas pu explorer l'aspect temps réel en raison du manque de temps disponible et des contraintes matérielles du projet. En effet, l'impératif de devoir utiliser uniquement le CPU lors du traitement bride ainsi les performances et rend la possibilité d'utilisation de l'application en temps réel avec les images du drone complexe.

6 Conclusion

Ce projet a été l'occasion de mettre en application les connaissances en réseaux de neurones et en QT que nous avons acquises au cours de notre formation. Nous avons contribué à l'avancement du projet BeesForLife, en travaillant tout d'abord sur l'amélioration du système de détection afin de réduire le taux de faux-positifs. Pour se faire, nous avons augmenter la taille du jeu de données au moyen de divers processus afin d'améliorer l'entraînement de nos modèles, eux aussi mis à jour en passant de Yolov7 à Yolov8.
Nous avons également commencé la mise en place de l'utilisation des images infrarouges en ajoutant le calcul des températures. Nous avons également participé à l'amélioration de l'ergonomie de l'application en revoyant l'interface et ses fonctionnalités afin d'en augmenter les performances. Nous avons ainsi amélioré le parcours, le chargement et l'affichage des images et amélioré l'utilisation des images résultantes de la détection au niveau du placement des masques et de l'exportation des images.
Au terme de ce projet, nous avons pu répondre à la majeure partie des attentes et besoins

fixés dans les User Stories.

Pour conclure, ce projet nous a permis d'approfondir nos connaissances dans le domaine de la détection d'objets. De plus, il a était très enrichissant de participer à ce projet, et contribuer à la lutte des frelons asiatiques.

Références

[1] Anaconda inc, page internet anaconda. https://anaconda.org/. Accessed : 2023-03-27.

[2] Bees for life, page internet de l'association. https://www.beesforlife.fr/. Accessed : 2023-03-27.

[3] Comparaison des performances de yolo d'après stereolabs. https://www.stereolabs.com/blog/performance-of-yolo-v5-v7-and-v8/. Accessed : 2023-03-27.

[4] The qt company, page internet qt. https://www.qt.io/. Accessed : 2023-03-27.

[5] Repository github de detecttechnologies. https://github.com/detecttechnologies/thermal_base. Accessed : 2023-03-27.

[6] Repository github de folium. https://github.com/python-visualization/folium. Accessed : 2023-03-27.

[7] Repository github de obss. https://github.com/obss/sahi. Accessed : 2023-03-27.

[8] Repository github de ultralytics. https://github.com/ultralytics/ultralytics. Accessed : 2023-03-27.

[9] Repository github de wkentaro. https://github.com/wkentaro/labelme. Accessed : 2023-03-27.

[10] Site internet plafrim. https://www.plafrim.fr/. Accessed : 2023-03-27.

[11] Site pyqt5. https://pypi.org/project/PyQt5/. Accessed : 2023-03-27.

[12] Site pyqtgraph. https://www.pyqtgraph.org/. Accessed : 2023-03-27.

[13] Site ultratytics, mode export. https://docs.ultralytics.com/modes/export/. Accessed : 2023-03-27.

[14] Ultrallitycs recommendation of background images. https://github.com/ultralytics/yolov5/issues/2844#issuecomment-851338384/. Accessed : 2023-03-27.

[15] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco : Common objects in context, 2015.

[16] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7 : Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *CoRR*, abs/2207.02696, 2022.