Rapport provisoire de stage de fin d'étude

Alexis Hoffmann

01/04/2022 - 31/08/2022

Table des matières

Chapitre 1

Introduction

Ceci est le rapport provisoire de stage du Master 2 d'ASPIC sur l'année scolaire 2021-2022. Encadré par M. Desbarats en tuteur de stage et par M. Chaumette en tant que référent de stage.

Le but du stage est d'améliorer une application qui fait une détection de nid de frelons asiatiques à partir d'images visibles et infra-rouges capturées depuis un drone. L'application est codée en Python dans un environnement Anaconda avec Qt pour l'interface.

1.1 Cadre de l'entreprise

Depuis le LaBRI, j'ai travaillé pour l'entreprise Bees for Life pour le client et pilote M. Willaert. Cette entreprise agit dans la protection des abeilles en luttant contre les frelons asiatiques. J'ai suivi une méthode agile, car tout du long du stage, j'ai eu beaucoup d'interactions avec le client. J'ai ensuite pu adapter ma méthode de développement en consacrant du temps sur les tâches prioritaires.

Le pilote a fourni au LaBRI un jeu de données provenant de 2 drones, le DJI Phantom 4 qui embarque une caméra couleurs et la caméra thermique FLIR Duo Pro et le DJI Mavic Entreprise 2 Advanced qui embarque pour sa part, une nacelle qui a une caméra couleurs et une thermique construitent par DJI.

1.2 Reprise du projet

Ce projet d'application date maintenant depuis plus de 2 ans.

La première version de l'application[1] utilise le modèle Mask RCNN. La deuxième version a apporté une amélioration dans l'interface utilisateur. La

troisième version quand à elle, a amené une amélioration dans l'expérience utilisateur et a permis de visualiser les images de manière simultanée (visible et termique correspondantes).

Chapitre 2

Application

2.1 Étude de l'existant

La version 3 que j'ai récupérée au début du stage le 1 avril 2022 contenait les éléments suivants :

— 4 boutons pour lancer la détection :

- modèle de détection sur les images thermiques entraîné sur les images de la FLIR Duo Pro
- modèle de détection sur les images thermiques entraîné sur les images de caméra thermique du Mavic
- modèle de détection sur les images visibles entrainés sur des images visibles
- détection se faisant sur les images synchronisées

— coordonnées en degrés décimaux sur l'image visible courante

— un bouton copier les coordonnées

— un qrcode renvoyant à la position de l'image courante

— 3 onglets de visualisation

- onglet avec les images simultanées, des flèches et une glissière, un bouton pour activer ou désactiver le masque sur le nid sur les images
- onglet avec les images séparées, des flèches pour chaque image, un bouton pour activer ou désactiver le masque sur le nid sur les images
- onglet de la carte qui montre la position moyenne des images où il y a eu détection.

— des menus pour afficher les courbes de l'entraînement des modèles.

2.2 Critique de l'existant

Le code de la création des boutons ou des éléments figure dans 2 méthodes de 300 lignes, ce qui fait long pour une réactualisation quand on bouge ou redimensionne la fenêtre par exemple.

Les éléments ont une position fixe ou alors selon un facteur de largeur de la fenêtre. Ce qui génère de nombreux bugs d'affichage.

L'onglet de la carte ne marche pas, il lève un `SEGFAULT`. Après la résolution du bug, aucun marqueur n'apparaît pour la position de la ruche sur la carte.

Le bouton de la détection sur des images synchronisées ne marche pas, impossible de rectifier l'erreur.

Pour installer l'environnement Anaconda, il y a un script bash ou bat contenant les commandes pour installer chacune des bibliothèques utilisées dans le code et plus. Or Anaconda fourni une façon de faire pour télécharger des paquets à partir d'un fichier yaml.

Aucun test n'a été réalisé ou trouvé dans tout le projet avec un framework de tests.

Aucune convention de nommage ou de documentation n'est présente dans le code.

Lorsque la détection est finie, il n'y a pas d'actualisation dans les images avec le masque, on est obligé de relancer l'application. De plus, les images générées ont un padding appliqué par le modèle.

Le point le plus criticable est le fait que l'application en tant que tel ne fonctionne pas sur le Mac du client. Le script d'installation échoue sur son ordinateur.

2.3 Objectifs

Voici la liste des objectifs dans l'ordre :

1. Réussir à installer l'environnement de développement Anaconda sur le pc du client.
2. Comprendre d'où venait l'erreur `unrecognized selector` sur son Mac, et la régler.
3. Refonte de l'architecture à la manière de Qt.
4. Remanier l'occupation de l'espace par les éléments avec de la mise en page Qt.
5. Rectifier les coordonnées affichées fausses.
6. Rafraichir les images en sortie et enlever le pad.
7. Défiler les images de manière optimisée avec mise en cache.

8. Charger les images en fonction du type de drone.
9. Séparer les données de leur chargement et leur utilisation.
10. Suivre les conventions PEP8 et formatage black.

2.4 Réalisation

2.4.1 Architecture

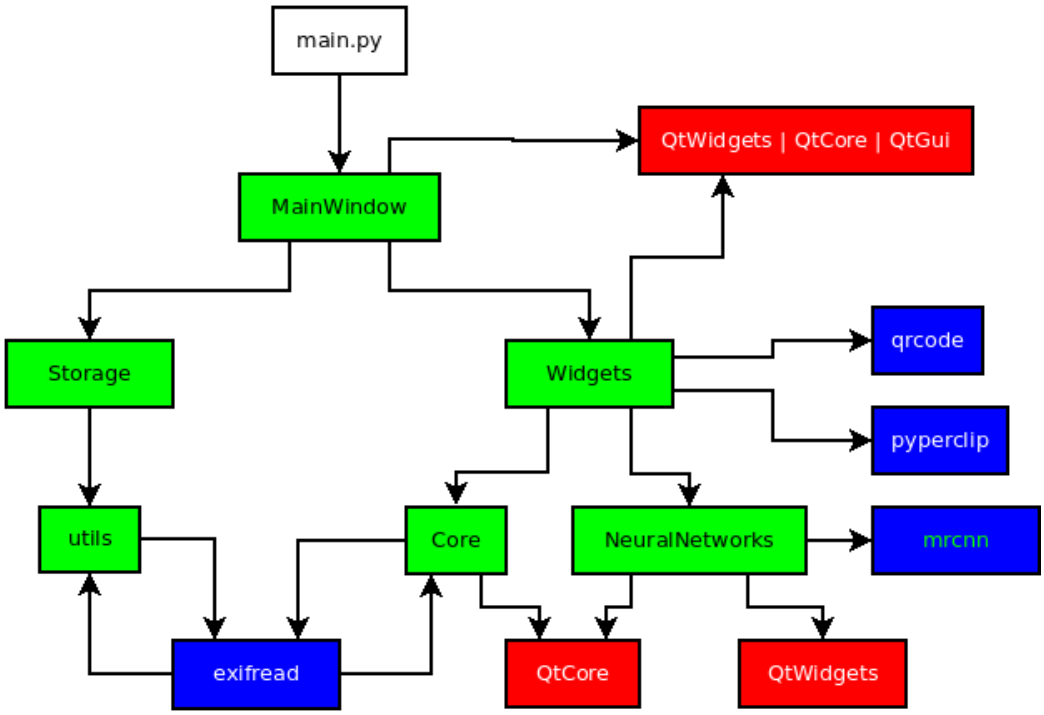


FIGURE 2.1 – Architecture Simplifiée

Les classes provenant de Qt sont en rouge, les classes provenant de BflQt sont en vert, les bibliothèques utilisées sont en bleu.

Une remarque sur `mrcnn`, cette bibliothèque est celle du Mask RCNN provenant de leur dépôt git.

Les liens avec les classes de `PyQt` sont d'origine multiple, en effet il y a de l'héritage, de l'utilisation, du typage et surtout de la manipulation de fichiers avec `QtCore.QDir`.

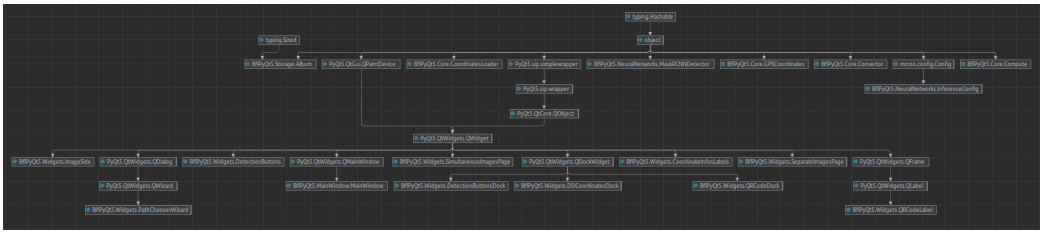


FIGURE 2.2 – Architecture générée par PyCharm

2.4.2 Outils

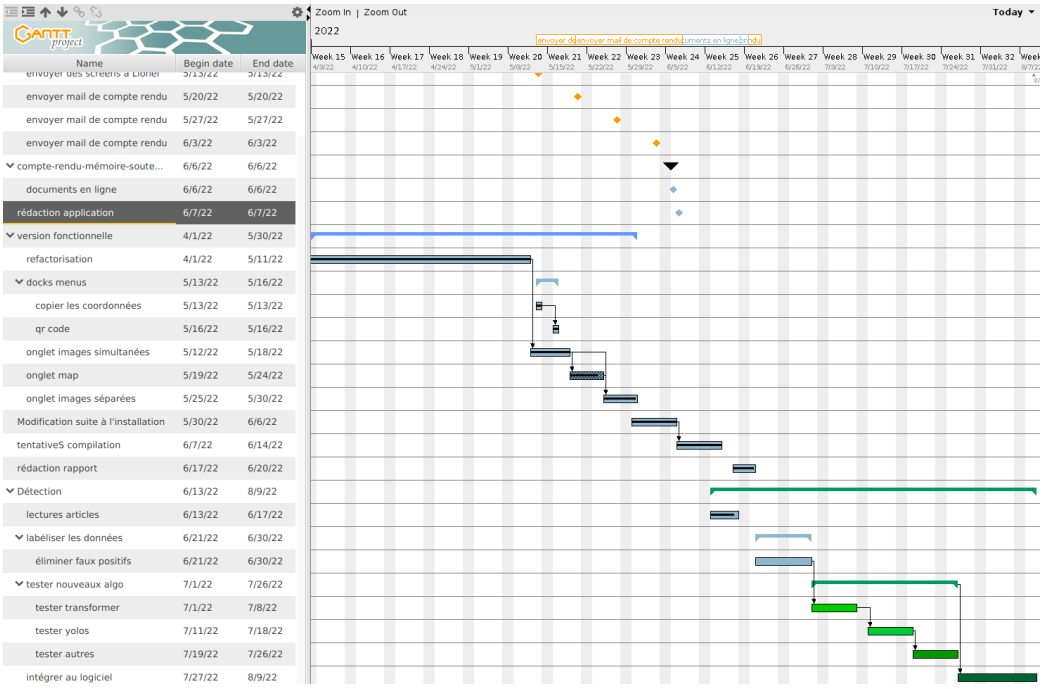
Concernant les différents outils, le principal est Qt que se soit `PyQt5` ou `PySide2`, les composants principaux sont primordiaux dans l'architecture.

Anaconda

N'ayant jamais utilisé Anaconda auparavant, j'ai passé un moment conséquent à me documenter et comprendre la phylosophie. Dorénavant, dès que je ferai un projet en python, je l'utiliserai systématiquement au vu de la puissance de l'outil et de la connaissance et de l'expérience acquis.

Pour ce qui est de l'environnement, le créer à partir d'un fichier comme ceci `conda env create --file environment.yml` est tout de même nettement plus simple qu'utiliser un script bash et la documentation sur ce point est très fournie.

Gantt



2.5 Tests

Les tests couvrent 67% du code. Les principales fonctionnalités qui sont testés à 100% sont les plus importantes. Notamment, `Core.py` avec des méthodes de chargements des coordonnées dans les images. Ainsi que `utils.py` qui a des fonctions de chargement d'image dans des dossiers. Par exemple, si on a 1000 fichiers mélangés avec des images visibles et infra-rouge et au hasard, des fichiers cachés .DS_STORE, on veut que le client n'ait pas à attendre 5 minutes que chaque fichier soit passé au crible et surtout on ne veut pas de faux positifs. D'où la nécessité de tester rigoureusement ces fonctions.

```
└─> pytest --cov=src/BflPyQt5 tests/
============================================================================================ test
platform linux -- Python 3.6.15, pytest-6.2.4, py-1.11.0, pluggy-0.13.1
PyQt5 5.15.6 -- Qt runtime 5.15.2 -- Qt compiled 5.15.2
rootdir: /home/alexis/Documents/BeesForLifeVersions/beesforlife/tests, configfile: pytest.ini
plugins: qt-4.0.2, dependency-0.5.1, xvfb-2.0.0, cov-3.0.0
collected 39 items

tests/test_core.py ..x.............
tests/test_mainwindow.py .
tests/test_neuralnetworks.py s
tests/test_storage.py s
tests/test_utils.py ..............
tests/test_widgets.py s....

---------- coverage: platform linux, python 3.6.15-final-0 -----------
Name                             Stmts   Miss  Cover
----------------------------------------------------
src/BflPyQt5/Core.py                77      0   100%
src/BflPyQt5/MainWindow.py          83     31    63%
src/BflPyQt5/NeuralNetworks.py      50     33    34%
src/BflPyQt5/Storage.py             44     18    59%
src/BflPyQt5/Widgets.py            454    162    64%
src/BflPyQt5/__init__.py             4      0   100%
src/BflPyQt5/utils.py               28      0   100%
----------------------------------------------------
TOTAL                              740    244    67%
```

2.6 Compilation

Dans cette partie, je voulais aborder l'intérêt que l'on aurait à avoir un fichier exécutable compilé à partir du code. M. Willaert possède un Mac, il doit utiliser le terminal pour activer lancer l'environnement conda, puis lancer l'application.

Après lui avoir configurer son Mac, pour mettre à jour le projet, il peut simplement faire un `git pull` pour avoir la dernière version.

Or tout cela n'est pas très pratique pour lui. J'ai donc exploré la piste de trois bibliothèques de compilation de projet python :

2.6.1 FBS

FBS permet de compiler pour n'importe quelle architecture.

Problème : il se télécharge que depuis pip. Or il a des dépendances avec pyinstaller qui provoquent des conflits irrésovables. (pyinstaller de pip)

2.6.2 Pyinstaller

Pour Pyinstaller du conda-forge, après lecture de la documentation, de multiples tests et configuration, j'ai réussi à compiler le projet sous linux.

Problème : la compilation est dépendant au système d'exploitation sous lequel on se trouve. Or sous MacOS, il y a cette erreur spécifique liée au noyau `_sysdata_darwin_darwin not found` insolvable.

2.6.3 Nuitka

Bibliothèque la plus compliquée à appréhender surtout que pour chaque compilation, il faut attendre entre 30 et 45 min. Nuitka permet de convertir le code en C, puis il compile ce code C en executable. Pour que la compilation puisse être en un seul dossier, on lance avec l'option `--standalone` ou en un executable `--onefile`. Nuitka suit les `import` des fichiers python et les génére en C ou utilise les binaires des bibliothèques importées.

Problème : une erreur avec la bibliothèque openmp survient à toutes les tentatives que j'ai essayé. Tensorflow l'utilise pour paralléliser ses calculs. Soit sur le conda-forge, elle est corrompue, soit le suivi des `import` ne fonctionne pas spécifiquement pour cette bibliothèque particulière. Ou bien je l'utilise mal.

Quoi qu'il en soit, après toutes les tentatives possibles, j'ai laissé tombé la compilation du projet. Si je trouve une autre solution avant la fin du stage ou que j'ai une illumination, je retenterai.

Chapitre 3

Recherche

En cours...

Voici simplement une idée des titres qui pourront se trouver dans le rapport final.

3.1 État de l'Art avec les Transformers (ViT ou DETR)

3.2 Amélioration de la robustesse

3.2.1 Élimination des faux positifs

3.3 Nouvel entraînement

3.3.1 Labélisation des images

3.3.2 Apprentissage en ligne ou hors-ligne ?

3.4 Évaluation de la méthode

Chapitre 4

Conclusion

4.1 Retour expérience

4.2 Mon avis

4.3 Perspectives

Bibliographie

[1] Tooba Shams, Pascal Desbarats : Detection of asian hornet's nest on drone acquired FLIR and color images using deep learning methods. IPTA 2020 : 1-6