Logiciel de classification générique : Application à la détection de maladie par marqueurs photoniques

Rapport de projet de fin d'étude

Auteurs :
Florian Dayre
Elodie Gaudry
Hugo Lecomte
Hugo Trarieux

Client :
Fabien Baldacci

28 mars 2022

Table des matières

1 Contexte et enjeux

1.1 Présentation du sujet

La classification d'images par ordinateur s'inscrit dans le domaine de l'apprentissage automatisé et de la vision par ordinateur. Elle consiste à entraîner un algorithme à différencier différents types d'images suivant des labels définis au préalable. Dans certains cas, le jeu de données utilisé pour entraîner l'algorithme nécessite d'être pré-traité pour optimiser l'apprentissage. Dans ce contexte, notre projet de fin d'étude consiste à concevoir une solution logicielle complète permettant la résolution de problèmes de classification. Pour cela, nous devions rendre possible depuis notre logiciel le chargement d'un jeu de données, l'application de pré-traitements, la réalisation d'augmentations de données, le lancement de l'entraînement d'un modèle de classification et enfin la classification d'une image à partir d'un modèle pré-entraîné.
Pour garantir l'aspect générique de notre application, nous devions nous assurer de pouvoir traiter des problèmes de classification d'images basiques comme par exemple différencier des animaux les uns des autres. Mais nous devions également pouvoir traiter des problèmes plus spécifiques nécessitant des pré-traitements, comme celui de la détection de la maladie de Lyme avec un jeu de données fourni par la PME qui a proposé notre sujet.

1.2 Présentation de l'entreprise cliente

Ce sujet a été présenté par l'entreprise « SARL Développement Durable », basée à Toulouse, spécialisée dans l'optimisation et la valorisation de déchets gras alimentaires, tels que les huiles alimentaires usagées. L'entreprise cherche actuellement à diversifier son activité, notamment en travaillant sur une technologie photonique d'imagerie macroscopique par effet couronne. Cette technologie permettrait la mise en évidence des états de santé d'un individu de manière non-invasive et rapide.

1.3 Portée du logiciel

Les premiers utilisateurs visés par ce logiciel sont les salariés de l'entreprise, afin de les aider à visualiser et à classifier les résultats de leurs expériences. Cette solution logicielle se veut générique pour permettre son utilisation et l'ajout de nouvelles fonctionnalités au plus grand nombre, développeur ou non.

2 Analyse de l'existant et état de l'art

2.1 Existant côté entreprise

Le jeu de données fourni avec le sujet est constitué de séries d'images d'empreintes digitales obtenues grâce à leur technique d'imagerie, correspondants à des individus atteints de la maladie de Lyme et des individus témoins. Pour ce jeu de données, il existait déjà une solution logicielle pour déterminer si une personne était malade ou non. Dans cette solution, on retrouve des pré-traitements spécifiques à ce jeu de données ainsi qu'une méthode de classification opérationnelle. Ces pré-traitements consistent à effectuer des rotations et des translations manuellement afin de corriger la mauvaise orientation et position des empreintes de la plupart des sujets. Étant donné que l'entreprise prévoit d'autres projets de classification avec cette même technique, il était nécessaire de réaliser une application générique dans laquelle il était facile d'ajouter de futures fonctionnalités. Nous avons choisi de ne pas partir de leur base de code, car pour la partie pré-traitement, nous avons souhaité

établir une solution automatique qui serait nettement plus rapide et efficace. De plus, pour la partie classification, nous ne voulions pas restreindre l'utilisation du logiciel à une solution spécifique, mais le garder le plus modulaire possible.

3 Scénarios et besoins

3.1 Scénarios utilisateurs

Au début du projet, nous avons imaginé différents scénarios utilisateurs pour notre application. Ils ont été amenés à évoluer au fil des semaines et nous sommes arrivés à l'élaboration de ce scénario principal constitué d'étapes interchangeables :

1. Ouverture de l'application
2. Chargement d'une base de données
3. Choix de pré-traitements à effectuer sur la base de données
4. Sauvegarde de la base de données modifiées
5. Choix d'augmentation de données à effectuer sur la base de données
6. Sauvegarde des augmentations d'images
7. Entrée des différents paramètres d'entraînement pour la classification
8. Lancement de l'entraînement
9. Affichage du résultat d'entraînement
10. Entrée des différents paramètres pour la classification
11. Affichage de l'image à classifier
12. Lancement de la classification
13. Affichage du résultat de la classification

4 Organisation

4.1 Gestion du Git

Pour la gestion du dépôt Git de notre projet, nous avons choisi de créer une branche pour chaque nouvelle fonctionnalité à ajouter à l'application. Ces branches étaient nommées explicitement par rapport à la fonctionnalité à implémenter. Une fois la tâche réalisée, le développeur à l'origine de la branche ramène les potentielles modifications de la branche principale sur sa branche et soumet son travail à un autre développeur de l'équipe. Ce dernier est chargé d'une revue de code, de garantir le respect des conventions adoptées, et de mener une batterie de tests exhaustifs sur l'application pour éviter une régression dans le projet. Passé ces étapes, les modifications de la branche secondaire sont compressées en un seul commit, décrivant en détails la fonctionnalité ajoutée, et ramenées sur la branche principale. La branche secondaire est ensuite renommée avec le préfixe "MERGED_", pour bien indiquer qu'elle n'est plus active. Cette stratégie garantit une branche principale soignée, sans commits superflus, et évite au maximum de passer du temps à régler des conflits lors du développement de fonctionnalités concurrentes. De plus, il est plus facile d'observer l'évolution du développement d'une fonctionnalité en particulier (Figure 21).

Enfin, nous nous sommes accordés sur l'utilisation des préfixes suivants dans l'écriture des messages de commits :

- **ADD** : Lors d'un ajout de fonctionnalité, de fichier ou de code.
- **CHANGE** : Décrit des modifications effectuées sur des fonctionnalités, sections de code déjà existantes.
- **FIX** : Pour les résolutions de bogues.
- **REMOVE** : Lors de suppression de fonctionnalité, de fichier ou de code.

4.2 Trello et Gantt

Pour la répartition des tâches dans l'équipe au fil des semaines, nous avons utilisé la méthode Kanban. Cette méthode permet de communiquer sur l'avancement, la répartition et la description des tâches. Il est primordial avec cette méthode de rédiger correctement les scénarios utilisateurs, afin de pouvoir les découper aisément en tâches d'une à quatre heures. Nous avions besoin d'accéder facilement à un Kanban en ligne, et de voir ses modifications en temps réels. Nous avons donc choisi le site Trello. Bien que cet outil soit propriétaire, il n'en reste pas moins efficace et rapide à prendre en main. Ce système a contribué à garder l'équipe motivée et organisée.
Notre Kanban est organisé en six parties distinctes :

- **Scénarios** : Description des scénarios utilisateurs.
- **À faire** : Tâches générales du projet.
- **Tâches de la semaine** : Ajoutées au début de chaque semaine, ces tâches doivent être réalisées avant la fin du sprint en cours.
- **En cours** : Permet de voir les tâches en cours de réalisation.
- **À tester/fusionner** : Tâches terminées qui demandent à être revues et testées par un autre membre de l'équipe n'ayant pas participé à la réalisation de la tâche, avant d'être ramenées sur la branche principale.
- **Terminé** : Tâches complètement terminées, ayant été revues et ramenées sur la branche principale.

Nous avons également ajouté un code couleur permettant de visualiser quelles tâches appartiennent à quels scénarios. De plus, lorsqu'une tâche était prise par une des personnes du groupe, celui-ci devait étiqueter la tâche avec ses initiales (Figure 1).

FIGURE 1 – Modèle de Kanban que nous avons utilisé pour le projet

Nous avons choisi de ne pas concevoir de diagramme de Gantt prévisionnel, étant donné la durée relativement courte du projet. Nous avons préféré se baser sur le Kanban et mettre à jour au fur et à mesure un diagramme de Gantt effectif afin de visualiser notre avancée (Figure 19). Comme nous pouvons l'observer sur le diagramme de Gantt, les tâches qui nous ont pris le plus de temps sont celles relatives à l'élaboration des scénarios utilisateurs, l'implémentation des pré-traitement et l'écriture des tests.

4.3 Interaction avec la PME

Au cours de nombreuses visioconférences et échanges écrits, les retours de personnes non-informaticiennes, extérieures à l'université, nous ont permis de voir sous un autre jour notre travail et de mettre en lumière certains aspects du projet auxquels nous n'aurions probablement pas porté autant d'attention sans eux. Il a été particulièrement intéressant de tenter de retranscrire et d'évaluer la faisabilité des besoins formulés par ces clients dans le cadre de notre travail. Cette expérience à donné lieu à une bonne mise en situation dans la réalité qui nous attend au terme de notre formation, dans le milieu professionnel.

5 Choix techniques

5.1 Outils utilisés

5.1.1 C++, QT et OpenCV

Pour développer notre projet, nous avons choisi le langage C++, avec lequel nous avons beaucoup d'expérience grâce à notre formation, et qui est reconnu pour sa rapidité et son efficacité, utile dans le traitement de large jeu de données. Concernant l'interface graphique, nous sommes à l'aise avec la bibliothèque QT, une référence en la matière. Enfin, pour le traitement des images, la bibliothèque OpenCV s'est imposée comme une évidence.

5.1.2 Interfaçage avec Python

Les bibliothèques principales de classification sont interfacées avec Python. Il était donc évident que nous devions gérer la possibilité d'interagir avec ce langage. De plus, ce langage est plébiscité aussi bien par les développeurs que les non-développeurs pour sa facilité d'utilisation. La capacité de prendre la main sur notre logiciel et ses états courants était séduisante.

5.1.3 GoogleTest

Nous souhaitions expérimenter cette bibliothèque de tests unitaires pour le C++. Nous l'utilisons pour évaluer le code métier de notre application.

5.1.4 Gcov et Lcov

Pour réaliser des évaluations de la couverture des tests nous avons choisi d'utiliser Gcov, le programme de couverture de code du compilateur Gcc. Afin d'avoir un retour plus détaillé sur le code que nos tests couvrent,

nous avons décidé de générer un rapport de couverture sous la forme d'une page internet avec Lcov.

5.2 Modalités de développement

5.2.1 Github

Nous avions besoin d'un hébergement pour le dépôt Git de notre projet. L'utilisation de GitHub nous a permis de montrer facilement le code aux clients de l'entreprise. La possibilité de montrer ce projet dans nos porto-folio respectifs ou pendant un entretien professionnel nous a également poussé vers cette solution.

5.2.2 Licence du logiciel

Le logiciel libre est une thématique qui nous tient à cœur, en particulier pour un projet universitaire. Nous avons naturellement choisi la Licence Publique Générale GNU (GPL-3.0-or-later), une des licences qui défend le mieux les droits et libertés des utilisateurs.

6 Description du travail réalisé

6.1 Fonctionnalités

6.1.1 Chargement des données

À n'importe quelle étape dans le logiciel, il est possible de charger des images afin qu'elles soient utilisées dans différents onglets de l'application. Pour ce faire, une barre de menu a été mise en place, dans laquelle on y retrouve le menu File, puis un sous-menu Open contenant les différentes méthodes pour ouvrir des données sous forme de .png, .jpeg, .jpg et de .tif (Figure 2). Actuellement, le logiciel permet deux méthodes distinctes de chargement et de gestion de données, à savoir Image Selection et Lyme Database.

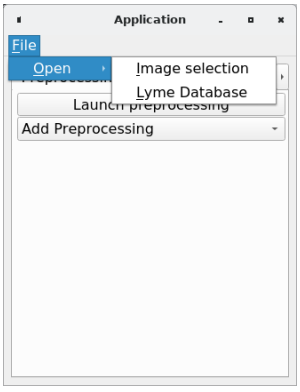

Figure 2 – Chargement de données dans l'application

— **Image Selection** : Permet de sélectionner manuellement les images à charger situées dans un même dossier.
— **Lyme Database** : Permet de charger toutes les images se trouvant dans tous les sous-dossiers du répertoire sélectionné.

Ces méthodes de chargement garantissent également la bonne gestion des données, pour l'affichage de la pré-visualisation ou leur sauvegarde une fois modifiées. Ensuite, nous allons expliciter les fonctionnalités principales de la solution logicielle, qui s'articule en quatre onglets distincts.

6.1.2 Onglet Pré-traitement

L'onglet de pré-traitement se divise en deux parties principales (Figure 3). L'onglet comprend sur sa gauche un aperçu de la base de donnée chargée (encadré en rouge), avec la possibilité de naviguer dans celle-ci à l'aide de boutons Précédent et Suivant, qui changent les images affichées dans l'aperçu. Sur la droite, encadré en bleu foncé, on retrouve l'affichage des transformations d'images. Ces dernières peuvent être ajoutées grâce au bouton déroulant «Add Preprocessing». Lorsqu'un pré-traitement est sélectionné, il est automatiquement ajoutée à la chaine de pré-traitements courante, ici encadrés en vert. Les pré-traitements fonctionnels dans l'application sont ceux ci-dessous :

— **Mirror** : Symétrie sur un axe vertical ou horizontal. (Fig. 14)
— **Grayscale** : Conversions des images en niveau de gris. (Fig. 15)
— **Automatic Rotation** : Rotation et translation automatique des images dans un même sens (implémenté pour les images du client). Notre client avait déjà une technique de rotation pour leurs images, mais à réaliser manuellement pour chaque image. Nous avons donc réalisé cette rotation automatiquement. Pour ce faire, nous avons utilisé la bibliothèque OpenCV qui nous a permis de réaliser une analyse en composantes principales (ACP) et de la détection de contours sur les images du client afin de trouver l'orientation de la forme du doigt. En effet, l'ACP est un processus de réduction de dimensions et permet notamment, pour un ensemble de points 2D, d'en extraire son orientation générale. Pour notre cas des images d'empreintes digitales, cela nous a permis de mettre en évidence l'orientation de l'empreinte par rapport à son axe vertical, et ainsi d'effectuer une rotation de l'empreinte par rapport à l'axe vertical de l'image. (Fig. 16 et 17)
— **Morphological Transformation** : Appliquer une dilatation ou une érosion aux images, paramétrable avec la taille du noyau et le nombre d'itérations de la transformation morphologique. (Fig. 18)

Chaque pré-traitement est appliqué aux images pré-visualisées, pour donner un retour dynamique à l'utilisateur. Il est également possible de sauvegarder l'ensemble du jeu de données chargé en mémoire, en lui appliquant les pré-traitements.

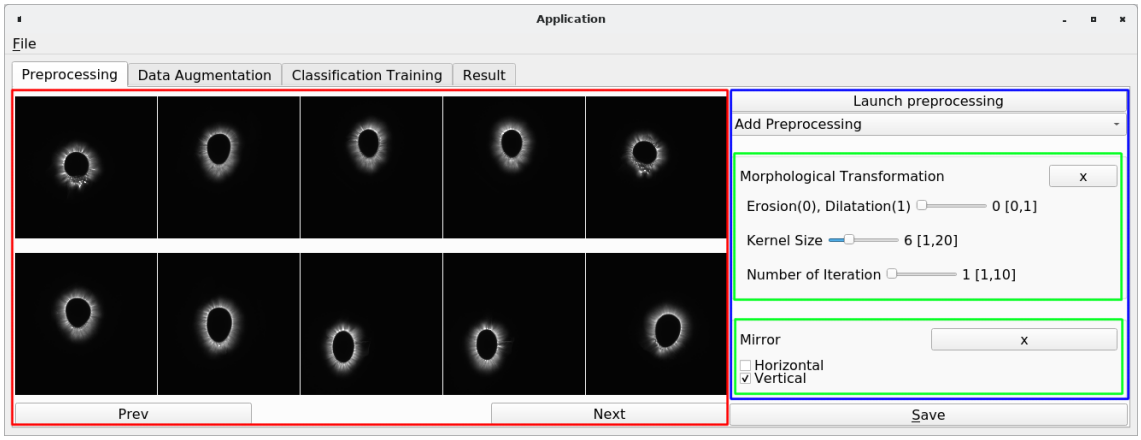

FIGURE 3 – Exemple d'utilisation de l'onglet Preprocessing dans l'application

6.1.3 Onglet Augmentation de données

Tout comme pour la partie Preprocessing, l'onglet d'Augmentation de données possède un aperçu du jeu de données chargé et la possibilité d'ajouter des transformations d'images. Cependant, ces transformations sont spécifiques à l'augmentation de donnée pour des expériences d'apprentissage automatisé. Il est possible d'ajouter plusieurs augmentations de données en même temps, de pré-visualiser les effets de ces dernières (encadré bleu ciel de la Figure 4) et de sauvegarder toutes ces nouvelles images.

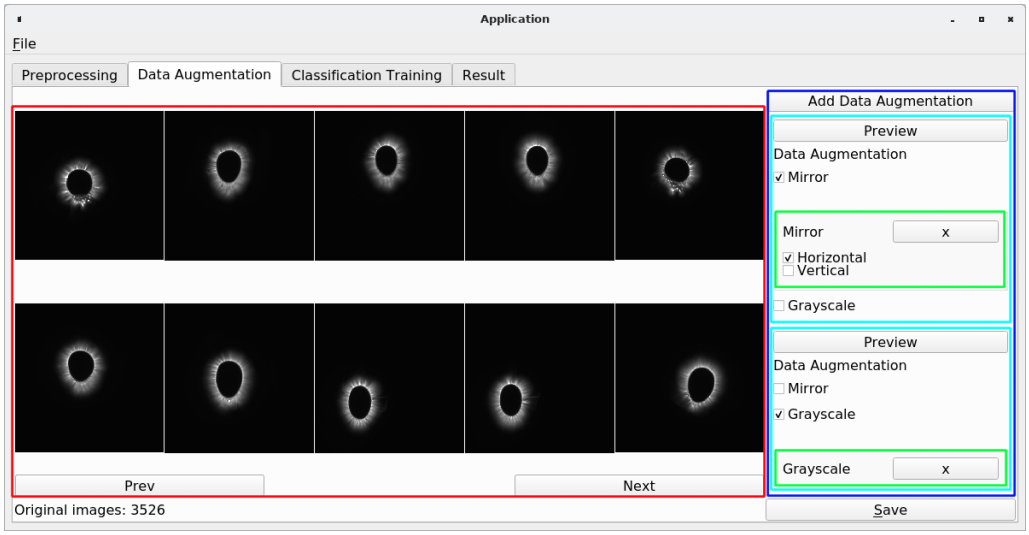

FIGURE 4 – Exemple d'utilisation de l'Onglet Data Augmentation dans l'application

6.1.4 Onglet Entraînement des méthodes de classification

Actuellement, l'application permet d'utiliser des modèles de classification d'apprentissage profond et les forêts aléatoires. Ces modèles doivent avoir été créés en amont par l'utilisateur, sous la forme de fichier python, en spécifiant les paramètres que l'on souhaite passer à l'application. Par exemple, pour les deux types de modèles, nous pouvons passer en paramètre une base de données d'entraînement et la taille des images de cette base de données, pour l'apprentissage profond, on peut ajouter le nombre d'époques que l'on souhaite effectuer pour l'entraînement et pour les forêts aléatoires, nous pouvons renseigner le nombre d'arbres dont se composera cette forêt. Il serait très facile dans le futur d'ajouter de nombreux autres paramètres que l'on souhaiterait passer à différentes méthodes de classification qu'on pourrait ajouter en créant de nouvelles classes enfants aux classes gérant l'affichage et le traitement de l'entraînement de la classification. Le fichier python d'entraînement chargé dans l'application s'exécute avec la commande "python3" à l'aide d'un QProcess de la bibliothèque QT qui permet de lancer des commandes dans un terminal. La commande d'exécution peut être modifiée dans les variables globales du fichier constants.h. La sortie standard ainsi que la sortie d'erreur de l'exécution du fichier python est redirigée dans deux fichiers textes distincts. Le fichier correspondant à la sortie standard est ensuite lu par l'application et son contenu est affiché pour l'utilisateur.

Nous allons maintenant voir comment cela se traduit sur l'application (Figure 5). L'encadré rouge est une liste déroulante permettant de choisir le type de modèle d'entraînement de classification que l'on souhaite effectuer (apprentissage profond ou forêt aléatoire). L'encadré bleu correspond au formulaire qui s'affiche une fois le type de modèle choisi, ici l'apprentissage profond. Une fois les paramètres renseignés, on peut lancer l'entraînement en appuyant sur le bouton correspondant. Cet entraînement va s'effectuer dans un thread parallèle à l'application, ce qui nous permet de naviguer dans cette dernière pendant l'entraînement. Les messages de l'encadré vert nous informent sur l'avancée de l'entraînement. Une fois terminée, l'application affiche dans la section de l'encadré jaune la sortie de l'entraînement qui correspond à la sortie d'exécution du fichier python d'entraînement, redirigé dans un fichier texte.

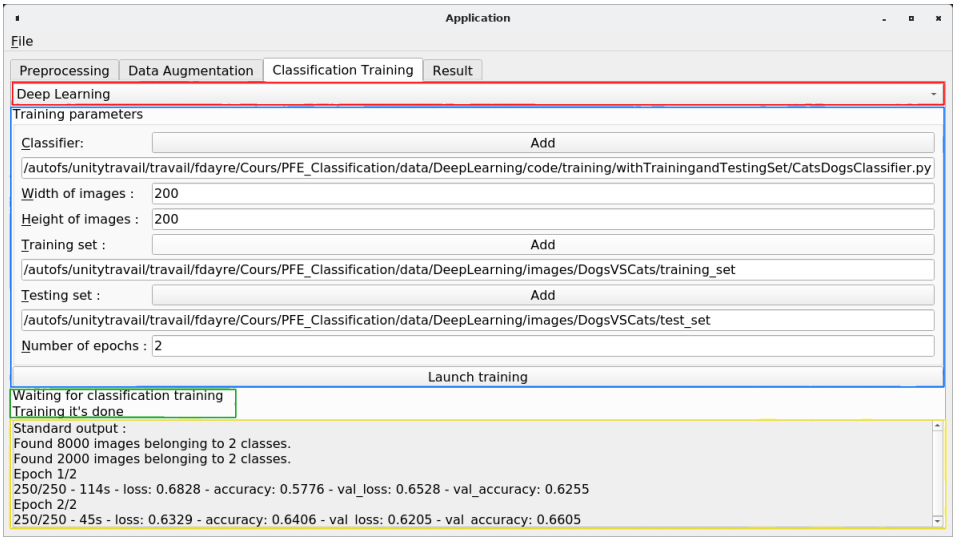

FIGURE 5 – Exemple d'utilisation de l'onglet Classification Training dans l'application

6.1.5 Onglet Résultat de classification

Pour la partie de classification, nous nous sommes rendu compte qu'il n'était pas utile de différencier la partie d'apprentissage profond de la partie forêt aléatoire. En effet, dans les deux cas, il faut pouvoir renseigner en paramètre un fichier de prédiction python qui se charge de récupérer les paramètres qu'on va lui passer via l'application, que sont : un modèle pré-entraîné, une image à classifier et un fichier texte possédant les labels des images sur lesquelles s'est entraîné le modèle. De la même manière que pour la partie entraînement, le fichier python de prédiction va ensuite être exécuté à l'aide d'un QProcess de QT et la sortie standard et d'erreur vont être redirigées sur deux fichiers textes distincts. Comme précédemment, nous pouvons voir comment cela se traduit sur l'application (Figure 6). L'encadré rouge correspond au formulaire où l'on indique les différents paramètres pour lancer une classification. Une fois les paramètres renseignés, il suffit d'appuyer sur le bouton correspondant pour démarrer la classification. Dans ce cas, l'image à classifier s'affiche sur l'application, à l'emplacement de l'encadré bleu, puis des messages apparaissent, comme pour l'entraînement, pour informer l'utilisateur du déroulement de l'entraînement. Enfin, le résultat de la classification s'affiche au niveau de l'encadré jaune. L'affichage du résultat correspond à la sortie standard du fichier de prédiction python.

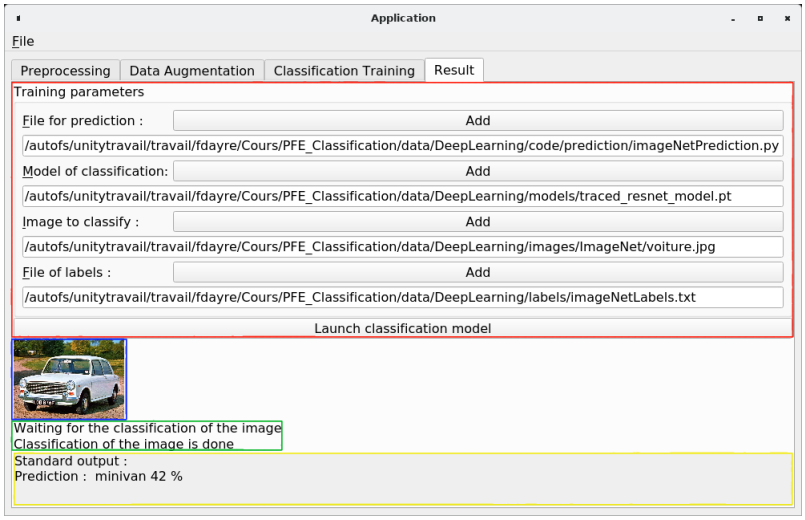

FIGURE 6 – Exemple d'utilisation de l'onglet Résultat dans l'application

6.2 Architecture

Pour notre application, nous avons choisi de travailler en programmation orientée objets et de prendre le soin de bien séparer l'interface graphique du code métier. De plus, nous avons essayé au maximum de rendre notre code générique pour qu'il soit relativement simple d'ajouter de nouvelles fonctionnalités à l'avenir.
L'application se découpe en quatre onglets, respectivement les quatre classes fille de la classe mère Tab, elle-même dérivant de la classe QWidget (7). On trouve ainsi l'onglet des pré-traitements (PreprocessingTab), de l'augmentation de données (DataAugmentationTab), de l'entraînement de la classification (ClassificationTrainingTab) et de la classification (ResultTab).

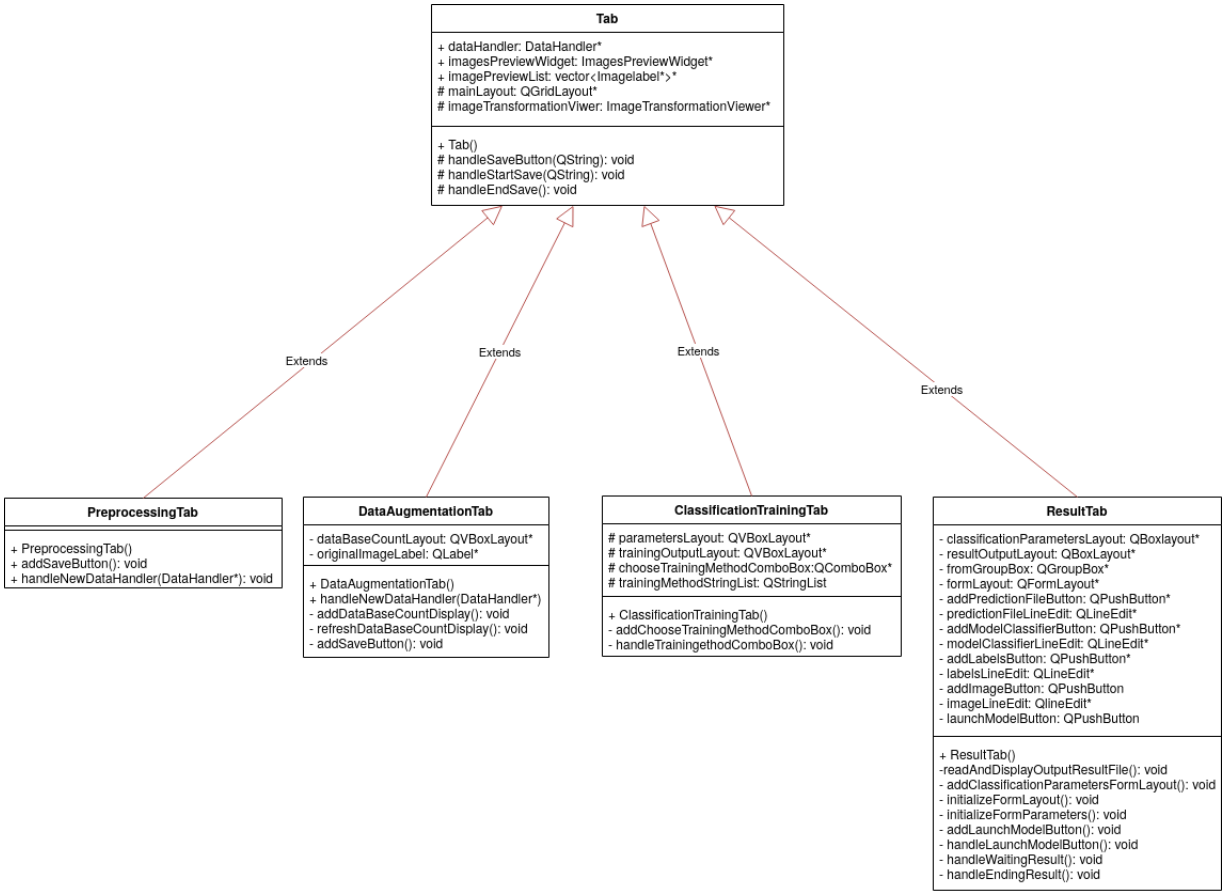

FIGURE 7 – Diagramme de classe des onglets

Pour commencer, une des classes les plus importante de notre application est la classe DataHandler. Cette dernière est responsable du chargement d'un jeu de donnée et de sa correcte utilisation lors de la sauvegarde et de l'affichage de l'aperçu. Il était prévu dès le début que notre application puisse traiter différents problèmes de classification, ce qui implique de charger de multiples jeux de données n'ayant rien en commun les uns avec les autres. En effet, on ne sauvegarde pas de la même manière quelques images sélectionnées dans un dossier, et des milliers d'images réparties en sous-dossiers "/Malades/Patient53/MainGauche/Annuaire.tif". Pour palier à ce problème, nous avons créé la classe DataHandler. Cette classe abstraite contient notamment les méthodes virtuelles de sauvegarde et de sélection des chemins vers les images. Actuellement, l'application comporte deux classes héritant de DataHandler : ImageSelection et LymeDataBase. Il est très simple d'implémenter au besoin d'autres classes héritant de DataHandler pour gérer le jeu de donnée d'un autre problème de classification, avec sa propre architecture de dossiers.

Dans la partie fonctionnalité, on a pu voir que les onglets Preprocessing et Data Augmentation étaient très similaires. Ces deux onglets se découpent en deux parties principales (Figure 3 et 4), à savoir une partie pré-visualisation de la base de données chargée (en rouge), ainsi qu'un affichage d'une ou plusieurs transformations d'images (en bleu foncé).

Afin d'éviter la duplication de code, et surtout d'avoir une structure robuste, permettant une bonne scalabilité du code, (Figure 10). Pour commencer, la pré-visualisation est gérée dans la classe ImagesPreviewWidget. Il est possible grâce à celle-ci d'ajouter, dans n'importe quel onglet, une pré-visualisation de la base de données chargée. C'est elle qui contient un lot d'image (imagePreviewList), sous forme de vecteur d'ImageLabel*, char-

gées à la volée lorsque l'utilisateur navigue avec les boutons Prev/Next. Le nombre d'images affichées dans cette pré-visualisation est paramétrable en changeant la constante MAX_NUMBER_OF_IMAGES_TO_LOAD dans le fichier constants.h, fixée par exemple à 10 pour le LymeDataBaseDataHandler. Cette décision, de ne pas charger toutes les images en même temps, mais plutôt de les charger au fur et à mesure avec deux boutons nous a paru logique pour que l'application puisse être utilisée pour traiter d'importants jeux de données tout en restant fluide.

Pour les transformations d'images, nous avons séparé le code métier de la partie affichage. Chaque transformation d'image est codée dans une classe héritant de la classe mère ImageTransformation. Cette classe possède une méthode virtuelle applyImageTransformation(), qui doit être surchargée pour appliquer la transformation d'image de la classe enfant sur une QImage. Actuellement, nous avons 4 classes qui héritent de ImageTransformation : GrayscaleImageTransformation, MirrorImageTransformation, AutomaticRotationLymeDataImageTransformation et MorphologicalTransformationImageTransformation. Un des avantages d'avoir un objet pour chaque traitement est de pouvoir faire des listes d'ImageTransformation qui pourront être utilisées pour la sauvegarde ou pour la pré-visualisation (Figure **??**). Pour intégrer ces traitements à l'interface graphique, il fallait cependant créer des widgets propres à chacun. Chaque transformation d'image possède donc sa classe héritant de ImageTransformationWidget avec une méthode à surcharger, displayUI(), pour afficher les objets nécessaires dans l'interface. À noter que les ImagesTransformationWidget possèdent une référence vers un objet de type ImageTransformation. Cette structure en 2 parties (Figure 9) permet de travailler à plusieurs sur de potentielles nouvelles transformations d'images et d'en ajouter très facilement. Un des avantages d'un objet pour chaque traitement est de pouvoir faire des listes d'ImageTransformationWidget qui pourront être utilisées pour la sauvegarde ou pour la pré-visualisation (Figure 8).

```
void ImageTransformationViewer::launchActivatedPreprocesses()
{
    for(ImageTransformationWidget *imageTransformationWidget : this->imageTransformationWidgetList)
    {
        if(imageTransformationWidget->isActivated)
        {
            imageTransformationWidget->imageTransformation->runImageTransformationOnPreviewList(this->imagePreviewList);
        }
    }
}
```

FIGURE 8 – Exemple d'utilisation des listes d'ImageTransformationWidget dans la méthode LaunchActivatedPreprocesses. Dans ce cas, on parcourt la liste et on applique chacun des pré-traitements activés à l'image courante.

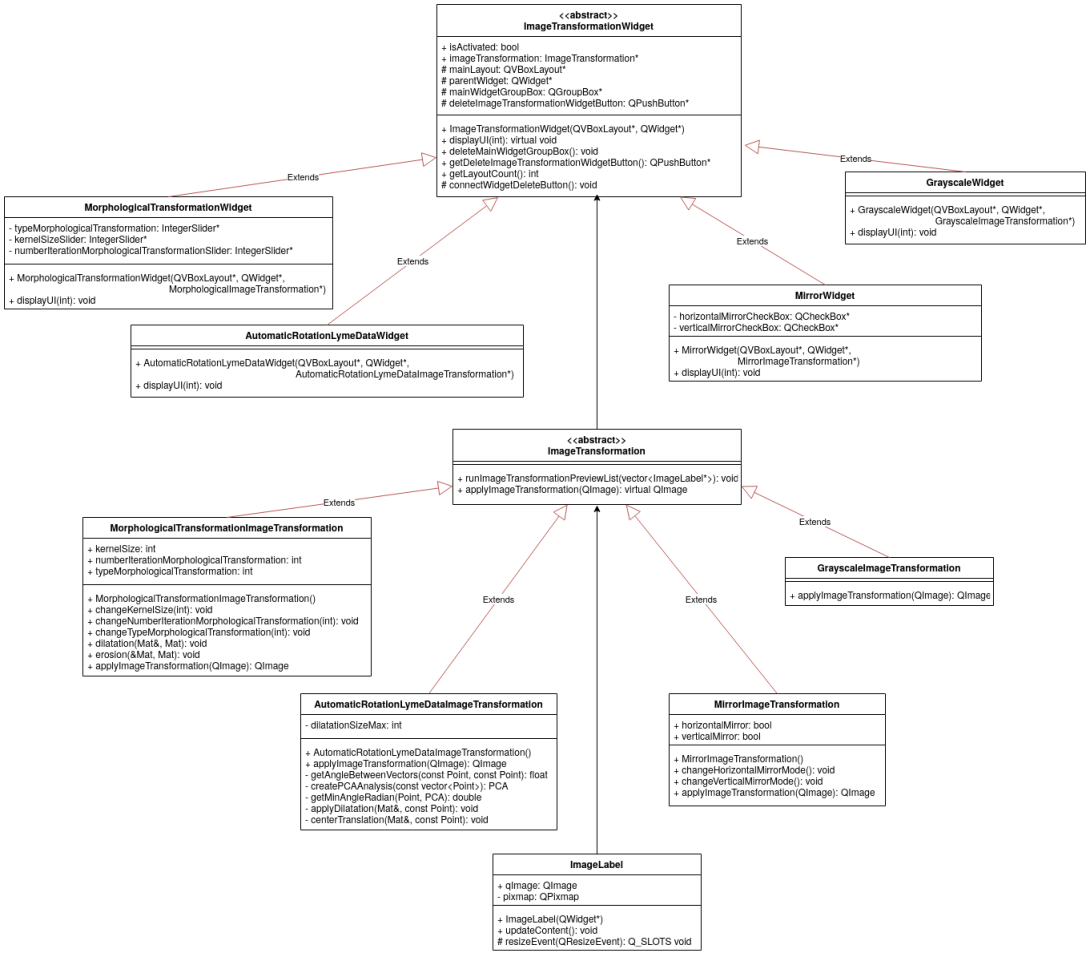

FIGURE 9 – Diagramme de classe pour la partie transformation d'images

Un autre point commun entre les deux onglets qui nous a poussé à généraliser le code est l'affichage des transformations d'images dans l'interface graphique. Que ce soit dans l'onglet réservé au pré-traitement ou dans l'onglet d'augmentation des données, il y a toujours cette possibilité d'ajouter une ou plusieurs transformations d'images puis de les supprimer. Pour ce faire, nous avons développé une classe mère ImageTransformationViewer, de laquelle hérite les enfants PreprocessingViewer et DataAugmentationViewer. Ces enfants ont donc de nombreux points communs réunis dans la classe mère, principalement un vecteur d'ImageTransformationWidget et des méthodes pour créer les Widgets associés aux ImageTransformation. Comme montré sur la figure 4, le DataAugmentationViewer (encadré bleu foncé) se différencie du PreprocessingViewer car il possède des DataAugmentationWidgets (encadrés bleu ciel). Ces derniers sont des objets pouvant contenir un ou plusieurs ImageTransformationWidget, et possédant un bouton Preview. Afin que les transformations d'images soient effectives sur la pré-visualisation, il est nécessaire de donner à l'objet ImagesPreviewWidget une référence vers une ImageTransformationViewer, pour lui faire communiquer la liste d'ImageTransformationWidgets à appliquer aux images de l'aperçu.

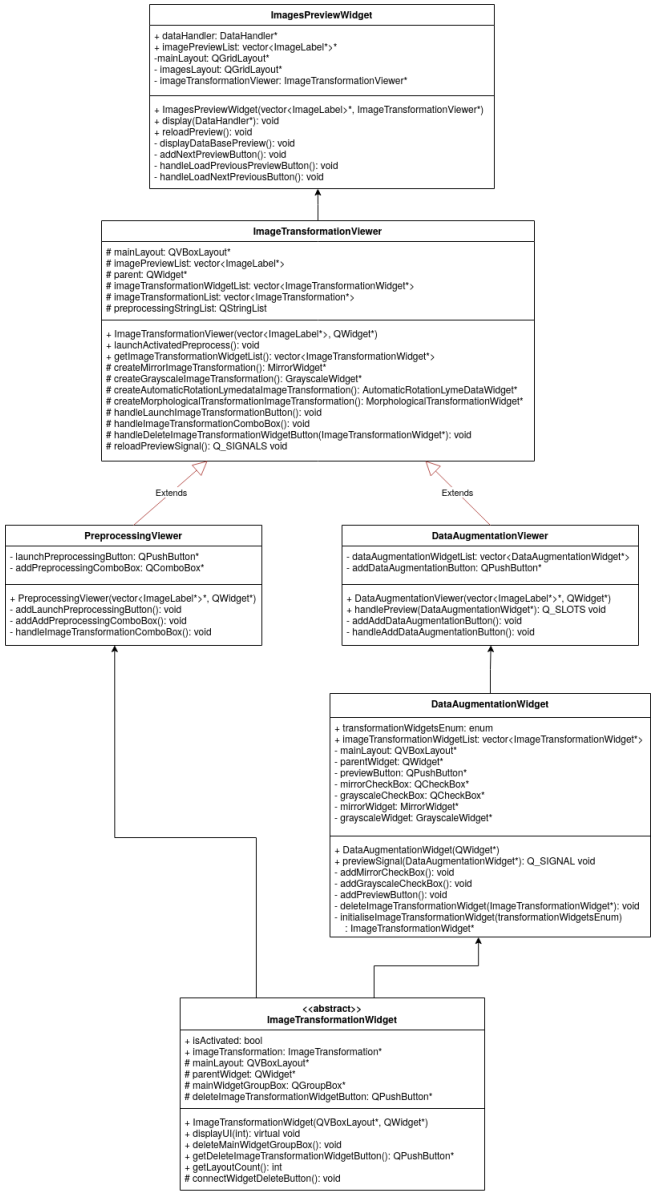

FIGURE 10 – Diagramme de classe dans l'onglet Preprocessing et Data Augmentation

Pour la partie classification, nous avons implémenté la partie affichage avec une classe mère abstraite, ClassificationTrainingWidget, dont dérivent deux classes enfants, DeepLearningWidget et RandomForestWidget (11). Ces deux classes prennent en charge l'affichage d'un formulaire à remplir pour renseigner les différents paramètres d'entraînement. Ces paramètres sont ensuite passés à la partie traitement qui se charge de lancer l'entraînement des modèles de classification. En ce qui concerne cette partie, nous avons implémenté une classe mère ClassificationThread, qui hérite de QThread de la bibliothèque QT, dont dérivent deux enfants DeepLearningThread et RandomForestThread (Figure 11). Lors du lancement de l'entraînement via le bouton correspondant, un thread est instancié (DeepLearningThread ou RandomForestThread) puis lancé. Ces deux classes surchargent la méthode run() de QThread, qui récupère les paramètres indiqués par l'utilisateur et les transmets à la classe mère via la méthode launchClassification() qui lance l'exécution du fichier python d'entraînement avec les paramètres correspondants.

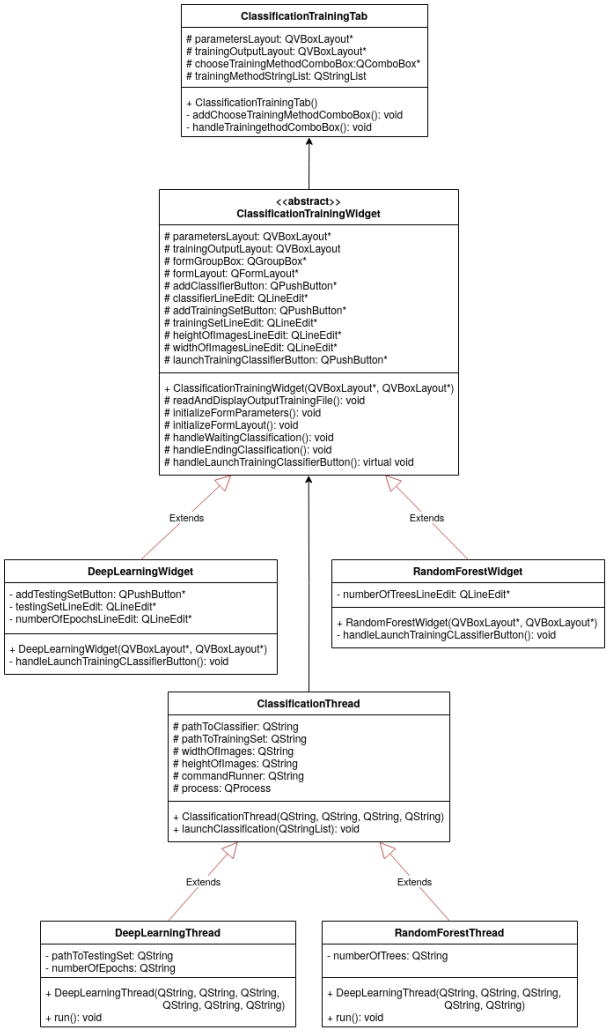

FIGURE 11 – Diagramme de classe de la partie d'entraînement de la classification

Pour la partie de résultat de la classification, la partie affichage est gérée par la classe ResultTab et le lancement de la classification par la classe ResultThread (Figure 12. Contrairement à la partie d'entraînement, nous n'avons pas implémenté des classes spécifiques pour gérer l'affichage et le traitement différemment suivant le type de modèle de classification que l'on souhaite utiliser. En ce qui concerne le lancement de la classification, nous reprenons l'idée de la partie d'entraînement en implémentant la classe ResultThread, dérivant de QThread, qui va se charger de lancer l'exécution du fichier de prédiction, implémenté par l'utilisateur, en lui passant les paramètres renseignés précédemment.

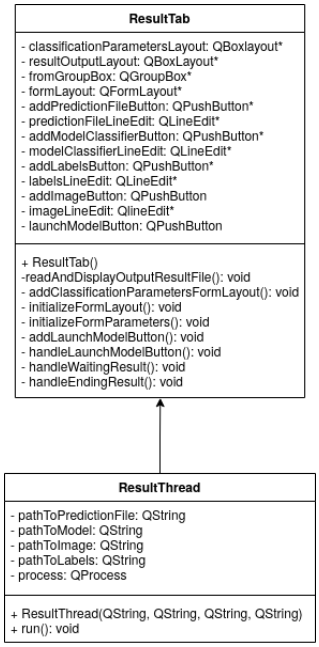

FIGURE 12 – Diagramme de classe de la partie de résultat de classification

6.3 Tests

En ce qui concerne les tests de notre application, nous avons dû tester différemment les parties du code métier et du code d'interface graphique de QT.

6.3.1 Tests unitaires

Pour le code métier, nous avons avons utilisé la bibliothèque de tests unitaire GoogleTest. Cette dernière ne fonctionnait pas quand il fallait tester des sections de code appelant des méthodes de QT. Ainsi pour les sections de code liées à l'interface graphique nous avons utilisé QTest, la bibliothèque de test unitaires de QT. Enfin nous avons choisis de ne tester que les méthodes publiques de notre application, partant du postulat que si ces dernières sont valides, le code protégé et privé l'est aussi.

6.3.2 Tests de scénarios

Par manque de temps et de compétence sur la bibliothèques de test unitaires de QT, nous n'avons pas réalisé de tests de scénarios automatisé.

6.3.3 Couverture des tests

Nous avons évalué la couverture des tests sur le code du projet, avec les outils Gcov et Lcov, permettant un affichage dynamique de l'évolution de cette couverture pendant l'écriture des test. De plus, cela nous a permis de localiser des sections de code mort, ou de code inaccessible.

On peut constater sur la fique 13 que les tests ne couvrent pas tout le code. En effet, comme dit plus haut nous n'avons pas implémenté les tests de scénario.

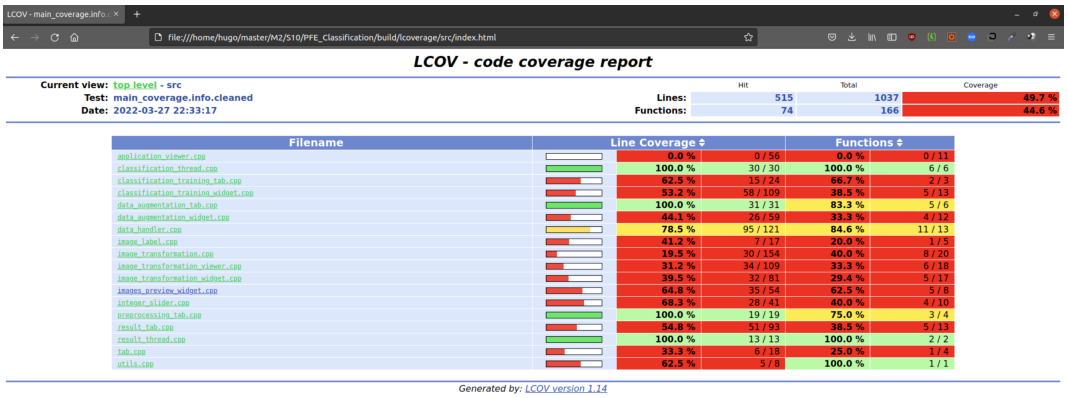

FIGURE 13 – Page LCOV actuelle de l'application

6.3.4 Limitations des tests

Dans le développement du projet, nous avons fait l'erreur de repousser le développement de nos tests afin d'avoir plus rapidement des résultat dans l'application. C'est en effet seulement après quelques semaines de développement que nous nous sommes penché sur ces derniers. Cependant, il a fallu une bonne semaine de mise en place et d'adaptation de GoogleTest et de QTest à notre CMakeLists afin de pouvoir rapidement et facilement implémenter des tests. Une fois les tests prêts à être écrit, nous avions quasiment déjà 1000 lignes de code à tester réparties sur de nombreuses classes, cela a donc été une tache plutôt fastidieuse.

6.4 Qualité du code

Nous avons souhaité que notre code soit le plus clair et compréhensible possible, pour nous-même ou d'autres développeurs. Ainsi, nous avons rigoureusement observé les conventions de codes sur lesquelles nous nous étions mis d'accord au préalable. Également, nous avons soigné les noms de nos variables, fonctions et constantes en faisant le choix d'être le plus explicite possible, pour éviter au maximum d'avoir recours à des commentaires lorsque nous pouvions l'éviter.

7 Analyse critique des résultats obtenus

Nous pensons avoir réussi à réaliser la conception et l'implémentation d'une d'un logiciel de classification assez modulaire, pour faciliter la mise en place de futures nouvelles fonctionnalités. Nous avons pu observer les avantages d'une telle méthode quand nous avons implémenté de nouveaux pré-traitements et une nouvelle méthode de classification, quelques semaines après avoir fini les implémentations de base. De plus, nous avons trois des quatre scénarios qui sont fonctionnels : les pré-traitements, l'entraînement d'un modèle et la classification d'une image.

En ce qui concerne les objectifs non atteints, le plus important concerne la sauvegarde de l'augmentation de données qui ne s'effectue pas comme attendu. En effet, les images sauvegardées à ce niveau-là prennent en compte uniquement la dernière augmentation de données lancée par l'utilisateur. À la base, nous voulions qu'un

certain pourcentage d'image du jeu de données reçoive les différentes augmentations de données paramétrées par l'utilisateur. De plus, la rotation automatique, pour le jeu de données de l'entreprise, est fonctionnelle mais pas parfaite. En effet, certaines images possèdent des artefacts empêchant de détecter les contours des empreintes digitales correctement. Enfin, nous avons pris beaucoup du temps en début de projet à redéfinir le sujet pour qu'il corresponde aux attentes de l'unité d'enseignement en terme de complexité relatif à notre niveau d'étude. Ce temps nous a manqués en fin de projet, expliquant ainsi l'incomplétude d'un des scénarios principaux.

8 Retour sur les choix techniques

Au terme de ce projet, nous avons quelque peu remis en question notre choix initial à propos du langage de programmation à utiliser. En effet, bien que le projet nécessite la manipulation et le traitement de jeu de données conséquente, il était au final peu pertinent d'utiliser le langage C++ pour réaliser l'ensemble du logiciel. Pour le public visé, le langage python aurait été plus adapté, pour laisser la possibilité de rentrer dans le code et de paramétrer facilement et finement certaines parties de l'application. Enfin, il est également avéré que le C++ demande un investissement en temps plus conséquent pour le développement ou la maintenance d'un code, que le langage Python par exemple.

L'utilisation d'une branche pour coder chaque fonctionnalité, associée à une revue par les pairs avant chaque fusion avec la branche principale nous a beaucoup aidé pour produire du code qualitatif, respectant les conventions établies en début de projet et éviter de propager des bogues dans le projet. Cependant, il aurait été très utile de garder une trace des échanges à propos des corrections à effectuer dans les branches avant de les fusionner sur la branche principale, dans le même style que les requêtes de tirage sur GitHub ou les requêtes de fusion sur GitLab.

9 Perspectives d'améliorations

Avec plus de temps devant nous, nous aurions pu améliorer notre application sur les points suivants :

L'interfaçage avec python dans notre logiciel. D'une part, pour faciliter les expérimentations sur les pré-traitements, pouvoir en ajouter d'autres très facilement sans avoir à re compiler tout le projet ou modifier le code C++ et d'autre part pour avoir la possibilité d'accéder à des états ou objets de l'application en temps réel.

Pour la rotation automatique, sur le jeu de données de l'entreprise, il serait possible de nettoyer les images en amont pour s'assurer de l'absence d'artefacts gênants lors de la détection de contours des empreintes digitales.

La possibilité de pouvoir manipuler les chaînes de pré-traitement d'onglet en onglet, notamment pour pouvoir pré-traiter des images avant de les passer dans les méthodes de classification ; ou encore sauvegarder la configuration du logiciel, chaîne de pré-traitement, chaîne d'augmentation de données, pour pouvoir reprendre une expérience au point où ont l'a stoppé.

Pour la partie entraînement, pouvoir créer son modèle de classification directement depuis l'application. Par exemple, pour l'apprentissage profond, il serait confortable, d'indiquer dans un formulaire, le nombre, le type et les paramètres de couches de réseaux de neurones. Cela permettrait d'éviter les allers-retours entre l'application et le fichier contenant le modèle d'apprentissage automatisé.

9.1 Bogues connus

Dans le code que nous remettons avec ce rapport, nous avons connaissance de l'existence de quelques bogues, que nous n'avons pas réussis à résoudre par manque de temps :

- Dans l'onglet Data Augmentation, le comportement de l'application lors de la suppression des transformations d'images n'est pas celui attendu et entraîne des erreurs de segmentation.
- La sauvegarde dans l'onglet Data Augmentation concerne uniquement la dernière série d'augmentation de données affichée dans l'aperçu.
- Les images dans la pré-visualisation de l'onglet de pré-traitement ne reviennent pas à leur état initial si l'on supprime tout les pré-traitement courants.
- Lorsqu'on commence à essayer de charger un nouveau jeu de données après en avoir déjà chargé un, mais que l'on se ravise, il advient une erreur de segmentation.

10 Sitographie

Références

[1] Sujet de création de logiciels. https://masterinfo.emi.u-bordeaux.fr/wiki/lib/exe/fetch.php?media=wiki:iis:pfe2022_creation_de_logiciel_sante.pdf. Présentation de l'entreprise et démarche proposée.

[2] Imagerie macroscopique à effet couronne au service de l'eau et de la santé. https://www.revue-ein.com/actualite/les-proprietes-de-l-eau-vue-par-imagerie-macroscopique-d-effet-couronne. Documentation sur l'imagerie macroscopique à effet couronne.

[3] Graphics view framework. https://doc.qt.io/qt-5/graphicsview.html. Documentation officielle de Qt.

[4] Googletest. https://google.github.io/googletest/. Présentation de GoogleTest.

[5] Git. https://github.com/Florian33850/PFE_Classification.git. Git du projet.

[6] Pme. https://sarl-developpementdurable.fr/. SARL Developpement Durable.

[7] Gpl3. https://www.gnu.org/licenses/rms-why-gplv3.en.html. Why Upgrade to GPLv3.

[8] Opencv. https://docs.opencv.org/3.4/index.html. Documentation officielle d'OpenCV.

11 Annexe

FIGURE 14 – Exemple d'application du traitement "Mirror"

FIGURE 15 – Exemple d'application du traitement "Grayscale"

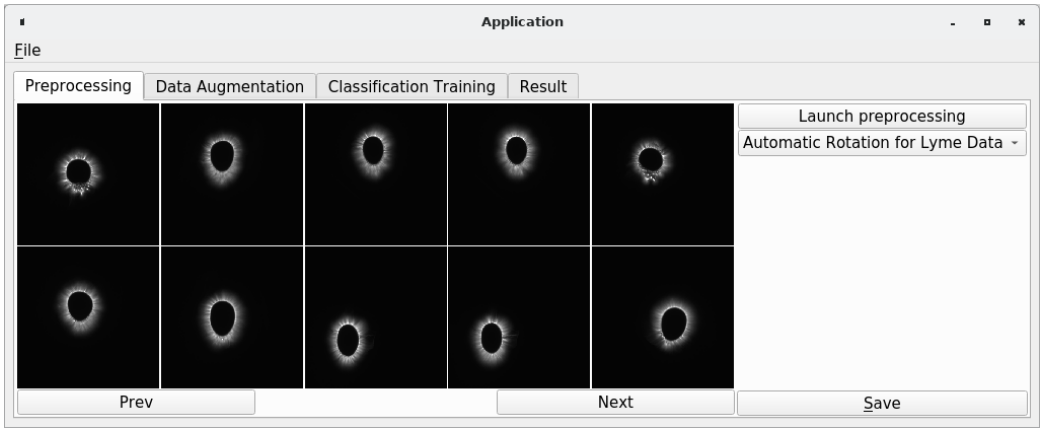

FIGURE 16 – Exemple d'un aperçu du jeu de donnée du client, sans le traitement "Automatic Rotation for Lyme Data"

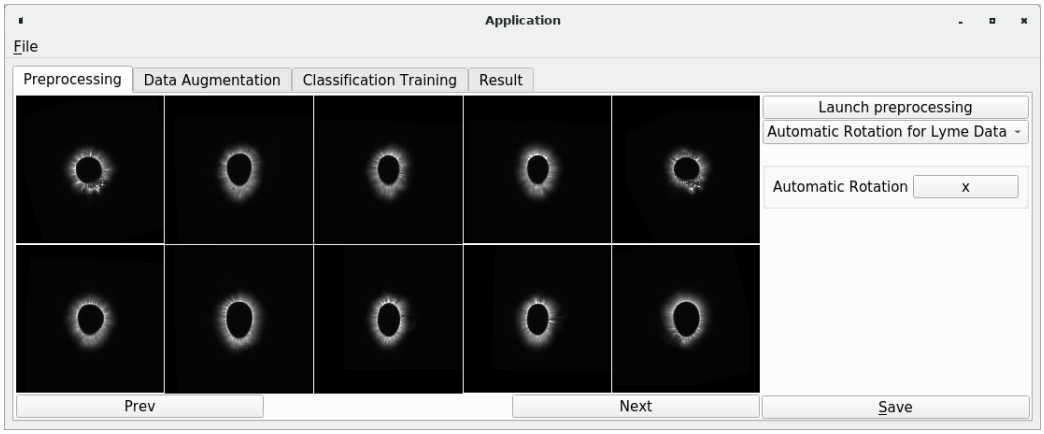

FIGURE 17 – Exemple d'un aperçu du jeu de donnée du client, avec le traitement "Automatic Rotation for Lyme Data"

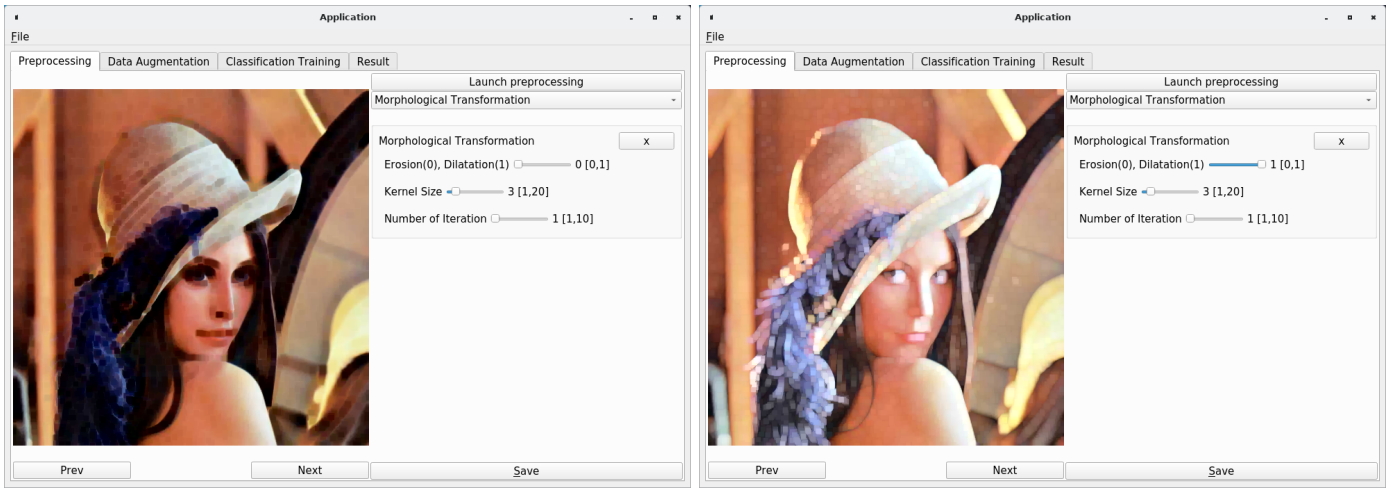

FIGURE 18 – Exemple d'application du traitement "Morphological Transformation"

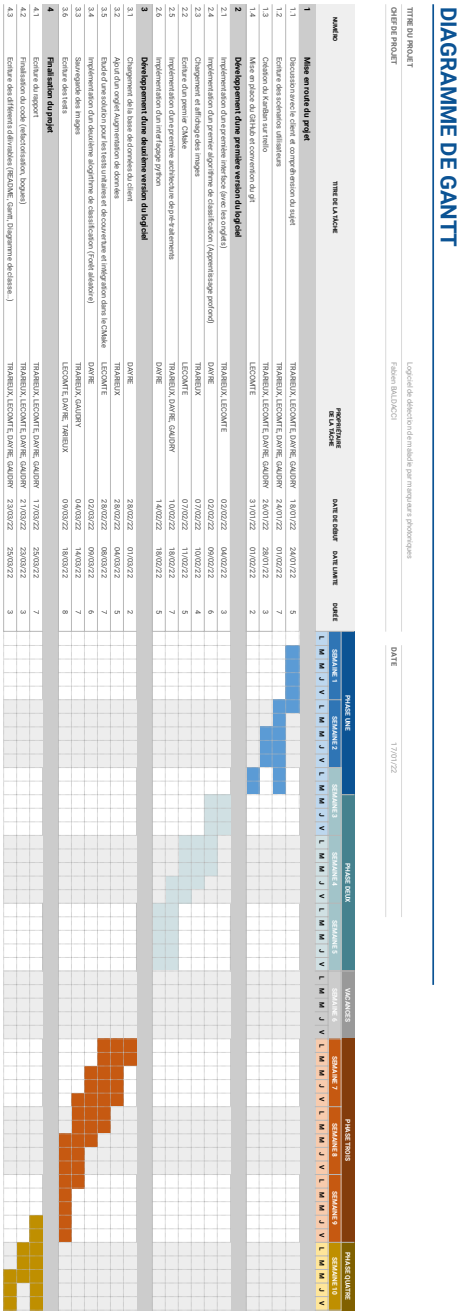

FIGURE 19 – Diagramme de Gantt effectif du projet

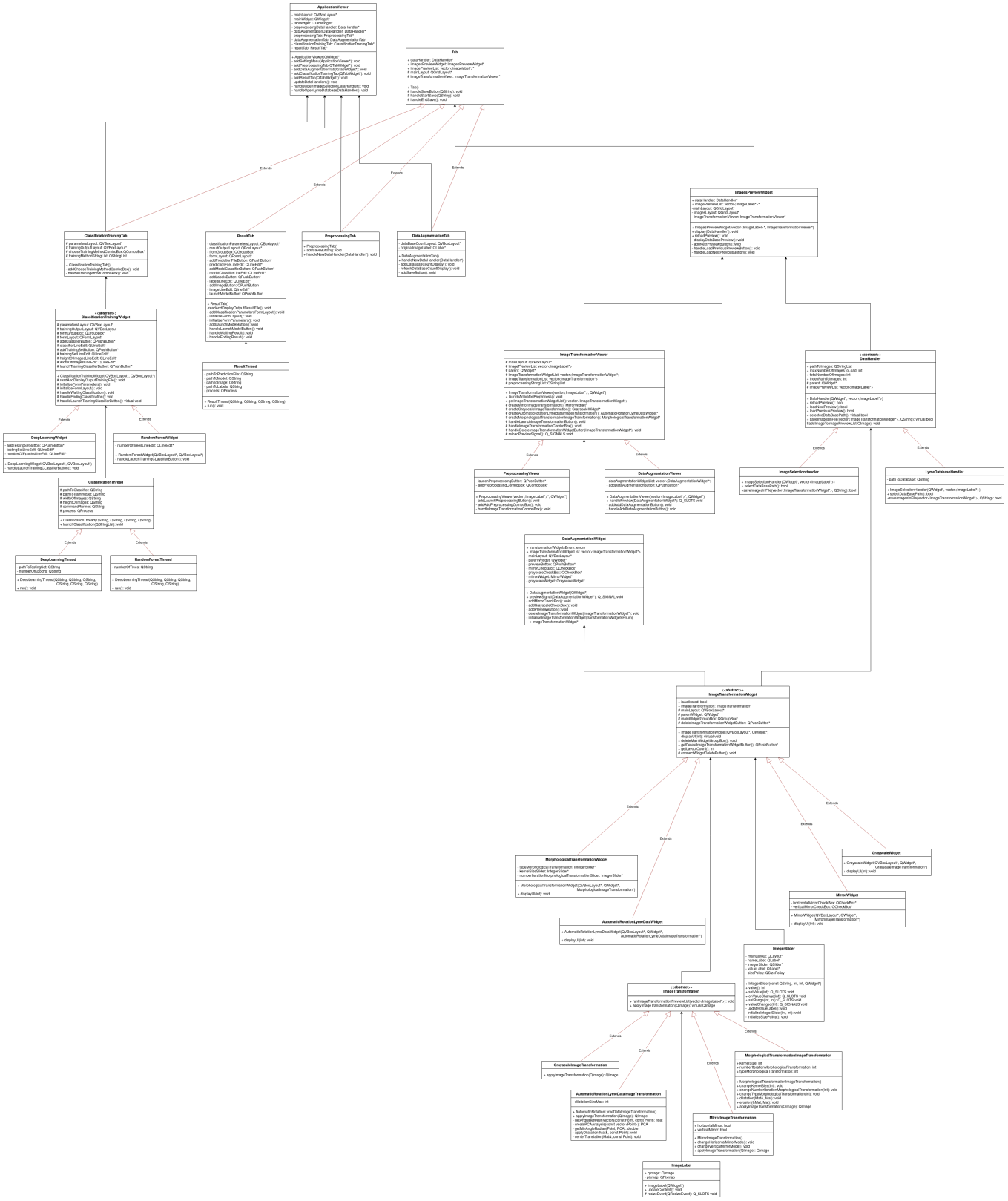

FIGURE 20 – Diagramme de classe du projet

```
Procédure de fusion des branches secondaire avec la branche principale:
  #Se positionner sur la branche principale et la mettre à jour par rapport au dépot git
  git checkout main
  git pull
  #Se positionner sur la branche à secondaire et la mettre à jour par rapport au dépot git
  git checkout maBrancheSecondaire
  git pull
  #Fusionner l'historique de la branche principale avec celui de la branche secondaire
  git merge main
  #Regler les conflits de fusion (dans VS code ou l'ide de votre choix)
  #/!\ ANALYSER LE CODE EN DETAIL!!!
  git diff main
  #/!\ TESTER EXHAUSTIVEMENT LE CODE!!!
  #communiquer les modifications au responsable de la branche, exiger qu'il prenne en compte les remarques et les commit sur sa
branche, revenir à la première étape
  #Commiter et envoyer les modification sur le dépot git
  git add des_fichiers des_fichiers des_fichiers
  git commit -m "MERGE: merge main into current branch"
  git push
  #Revenir sur la branche principale
  git checkout main
  #Verifier que la branche principale est bien à jour, si ce n'est pas le cas revenir à l'étape 1
  #Importer les modifications (et uniquement les modifications) dans la branche principale
  git merge --squash maBrancheSecondaire
  #Verifier et régler les conflits, TESTER LE CODE
  git commit -m "KEYWORD: EXHAUSTIVE DESCRIPTION OF THE SECONDARY BRANCH"
  git push
```

FIGURE 21 – Procédure de fusion des branches