Conception d'un prototype de microscope intelligent et autonome

Mael Balluet

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Automatique, Productique et Robotique – Signal, Image, Vision – Télécommunications*

Par

Maël BALLUET

Conception d'un prototype de microscope intelligent et autonome

Thèse présentée et soutenue à Rennes, le 22 janvier 2021
Unité de recherche : Institut de Génétique et Développement de Rennes / Inscoper SAS
Thèse N° :

Rapporteurs avant soutenance :

Loïc LE GOFF	Chercheur CNRS, Institut Fresnel, Marseille
Nadine PEYRIERAS	Directrice de recherche CNRS, FRE2039 Multilevel Dynamics in Morphogenesis

Composition du Jury :

Président :	Olivier DAMERON	Professeur, Université de Rennes
Examinateur :	Yannick GACHET	Directeur de recherches, Université Toulouse 3 Paul Sabatier
Dir. de thèse :	Jacques PÉCRÉAUX	Chercheur CNRS, IGDR
Co-dir. de thèse :	Marc TRAMIER	Ingénieur de Recherche HC CNRS, IGDR
Encadr. de thèse :	Otmane BOUCHAREB	Directeur général d'Inscoper SAS

Invité(s) :

Olivier CHANTEUX	Président d'Inscoper SAS
Sébastien LE NOURS	Maître de conférences, IETR, Université de Nantes

Remerciements

C'est avec beaucoup d'émotion que j'écris ces lignes dans le but de remercier toutes celles et ceux qui ont contribué à l'accomplissement de cette thèse, et qui ont rendu possible ces trois années inoubliables.

Je tiens dans un premier temps à remercier mes encadrants Jacques Pécréaux, Marc Tramier, Otmane Bouchareb et Olivier Chanteux pour m'avoir permis de vivre cette expérience enrichissante qu'est la thèse. Votre bienveillance et tous les échanges que nous avons eu ont été pour moi une source de motivation tout au long de ce projet très formateur.

Je voudrais remercier tous les membres des équipes CeDRE, MFQ et SPARTE de l'IGDR. Merci à Jacques, Hélène, Laurent, Youssef, Christophe, Sylvain, Nina, Yann, Marc, Giulia, Gilles, Angélique, Begüm, Seyta, Florian, Sébastien, Catherine, Rebecca, Audrey, Ostiane, Siham et Rajesh, pour votre accueil, votre investissement et votre bonne humeur. Plus généralement, je remercie aussi l'ensemble de l'IGDR pour ces trois super années. Un grand merci à tous les membres du Lunch Play pour les midi et soirées jeux de sociétés.

Je tiens aussi à remercier tous mes collègues d'Inscoper, Olivier, Otmane, Jérémy, Baptise, François-Eric, Célia, Elena et Bruno, pour m'avoir apporté leurs conseils et pour la super ambiance au sein de l'entreprise. Merci pour nos (très) longues discussions sur le projet Roboscope, et d'avoir remué sa conception dans tous les sens.

Je remercie aussi mon jury de thèse Loïc Le Goff, Nadine Peyriéras, Olivier Dameron, Yannick Gachet et Sébastien Le Nours d'avoir accepté de d'évaluer mes travaux de thèse.

Je remercie de nouveau Sébastien Le Nours, ainsi que Thomas Walter d'avoir fait partie de mon comité de thèse, et dont les conseils m'ont été très utiles.

Merci encore à Sébastien pour m'avoir donné l'occasion d'encadrer des travaux pratiques à Polytech Nantes. C'était pour moi une expérience très enrichissante et qui me tenait vraiment à cœur.

Je remercie très chaleureusement toute ma famille, et particulièrement ma mère Pascale, mon père Jean et mon frère Mehdi. C'est grâce à vos encouragements et à votre soutien que je me suis lancé dans cette aventure. Je n'aurais probablement pas parcouru tout ce chemin sans toute la motivation que vous m'avez apportée. Merci d'avoir fait de moi la personne que je suis aujourd'hui.

Je remercie tout aussi chaleureusement Tongxue pour m'avoir accompagné durant ce parcours. Merci pour tout ce que tu m'as apporté depuis notre rencontre. Tu es toujours

restée à mes côtés durant cette aventure, surtout dans les moments où j'en avais le plus besoin. Ton amour, ton soutien et tes encouragements ont énormément contribué à l'accomplissement de cette thèse. Merci à toi de faire partie de ma vie.

Je souhaite remercier tous ceux que j'ai suivis pendant cette thèse, Florian, Mathilde, Marie, Vincent, Lucie, Antoine, Guillaume, Lilian, Amanda, Solenne et Paul. Un très grand merci à vous pour tous les bons moments passés à l'IGDR, pour les soirées, les raclettes, les balades à vélo et toutes les chouettes sorties que nous avons pu faire. Vous avez fait de ces trois années une expérience inoubliable.

Je remercie aussi le club d'aïkido Tomoe de Villejean, et particulièrement Loïc, Nicolas et Nadège pour la grande qualité de leurs cours et pour la bonne ambiance au sein du club.

De manière plus générale, je remercie tous mes amis et toutes les personnes que j'ai eues la chance de rencontrer tout au long de mon parcours. Que ce soit lors de mes passages à Tours, à Nantes, en Irlande, en région parisienne ou à Rennes, vous avez été très nombreux à avoir rendu possible l'accomplissement de ce long périple. Un grand merci à vous tous.

Je voudrais conclure ces remerciements, bien que je n'ai pas eu la chance de les rencontrer, en remerciant tous les musiciens dont la musique m'a longuement accompagnée durant ces trois ans.

Résumé

Les systèmes de microscopie optique sont des outils complexes, indispensables à la compréhension du vivant par la recherche fondamentale et appliquée en biologie. Ils sont aussi un outil de choix dans la recherche de nouveaux médicaments par criblage dans l'industrie pharmaceutique. Leur automatisation est un champ d'étude en plein essor. Il s'agit d'analyser de manière autonome les images lors de leur acquisition afin de décider ce qui doit être imagé et dans quelles conditions. Cela permettra non seulement une accélération et une standardisation des expériences, mais ouvrira aussi la voie à l'étude de phénomènes biologiques complexes, jusqu'alors inaccessibles. Actuellement, la microscopie est soit hautement supervisée et utilisée par des experts en recherche, soit cantonnée à des approches rudimentaires lors des cribles.

Nous présentons dans cette thèse la conception et le développement d'un prototype de système embarqué analysant des images de microscopie en temps-réel et réalisant une boucle de rétroaction avec un microscope. Nous avons conçu ce système en confrontation permanente avec des applications biologiques théoriques plus ou moins complexes, ce qui nous a permis d'obtenir un modèle théorique générique et modulaire. Nous avons aussi entamé le test de ce système en conditions réelles. En particulier, notre système permet de modifier à la volée des modalités d'acquisition d'images d'un microscope en fonction des images analysées à la volée. De plus, il effectue un tri de ces images en ne retournant que celles représentant des objets d'intérêt pour les biologistes.

Un point critique de cette démarche est l'analyse des images en temps-réel. Nous nous sommes interrogés sur la manière de réaliser des classifications précises et rapides. Nous proposons une méthode d'optimisation d'un outil générique de classification d'images. Pour ce faire, nous identifions des caractéristiques pertinentes pour la classification des images dont le temps d'extraction reste faible. Dans cette démarche, nous avons mis en évidence une redondance des caractéristiques qui permet d'exclure les plus longues à calculer, quand bien même ce sont les plus discriminantes dans la classification. Ainsi, une dizaine de caractéristiques ainsi choisies permet de classer 14 cellules par seconde, avec une précision supérieure à 80 %, avec un algorithme de *random forest* ou un réseau simple de neurones. Ces travaux ouvrent des perspectives d'optimisation des systèmes de classification en apprentissage automatique, y compris profond.

En conclusion, nous avons mis en place les bases d'un microscope intelligent et autonome. Au-delà de l'accélération de la recherche, il ouvre la voie à une nouvelle génération de microscopie, à l'instar du séquençage, contribuant à l'émergence d'une médecine de précision basée sur la microscopie.

Mots clés : Système embarqué, Rétroaction, Temps-réel, Analyse d'images, Microscopie, Biologie cellulaire.

Abstract

Optical microscopy systems are complex tools, essential to investigate the living through fundamental and applied research. They are also first-rate tools to screen for new drugs in the pharmaceutical industry. Automating them is a vibrant field. It aims at autonomously analyzing images during their acquisition to decide on-the-fly what should be imaged and using which settings. Not only it will accelerate and standardize experiences, but also will pave the way to study biological processes still unreachable. Microscopy is currently either highly supervised and used by experts, or used simplistic setup for during screening.

We report in this PhD dissertation the design and development of a prototype of an embedded system performing real-time image analysis and feeding back to microscope control. We challenged the design of this system by confronting it to a panel of biological applications, considered theoretically. We obtained a generic and modular solution. We already started testing it in real conditions. In particular, our system can modify the microscope settings in real-time, depending upon simulated acquired images. Furthermore, it sorts out these images, returning only the ones relevant for biology.

Key to succeeding in this approach is real-time image processing. We sought an accurate and fast classification method. We offer a way to optimize a generic image classification. To do so, we identify the features relevant for classification and which extracting is not computationally intensive. In this process, we highlighted a feature redundancy, which allows us to exclude the most computationally-intensive one, even they are highly discriminant for classifying. Thus, about ten so-selected features are enough to classify 14 cells per second with an accuracy better than 80 %, using random forests or a simple neural network. This work paves the way to optimizing machine learning methods, including deep learning.

In conclusion, we developed the foundation of a smart and autonomous microscope. Beyond accelerating research, such an apparatus promises new microscopy, similar to next-generation sequencing, and will contribute to materializing a high precision medicine based on microscopy.

Key words : Embedded system, Servo-control, Real-time, Image analysis, Microscopy, Cell biology.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Glossaire

bagging Méthode d'apprentissage automatique consistant à entrainer plusieurs classificateurs dits "faibles" avec différents sous-ensembles du jeu d'apprentissage. 29

binning Fusion de pixels adjacents afin d'augmenter le rapport signal sur bruit au prix d'une réduction de la résolution spatiale. 13, 45, 60

boosting Méthode de d'apprentissage automatique consistant en l'utilisation de classificateurs dits "faibles" plusieurs fois de manière itérative afin d'obtenir un classificateur dit "fort". 29, 33, 111

bootstrap Répétition d'une expérience plusieurs fois avec différentes données en entrée pour chaque itération. 47, 49, 51, 52, 96, 118–120, 122, 123, 126, 127

pipeline Exécution en cascade de plusieurs instructions par un processeur. 29, 33, 35, 124

temps d'exposition Temps durant lequel le capteur d'une caméra photonique capture des photons. 12, 45, 60, 84

temps-réel En informatiques, un système est dit temps-réel lorsqu'il contrôle un procédé physique suffisamment rapidement par rapport à la vitesse d'évolution de ce dernier. 5, 16, 35, 40–42, 55, 56, 58, 61, 69, 70, 73, 75, 89–93, 95, 99, 101, 105, 123, 129, 135, 137–140

thread Portion d'un programme pouvant être exécutée indépendamment et de manière concurrente avec d'autres portions. 43, 70–72, 77, 78, 82, 85–89, 92

time-lapse Images acquises uniquement selon la dimension temps. 56, 65, 81, 83, 84, 86, 88–90, 96

Acronymes

ACP Analyse en Composantes Principales 22, 23, 25, 26, 28, 101, 102, 130

ANOVA Analysis of Variance (Analyse de la Variance) 47, 101, 102

API Application Programming Interface (Interface de Programmation Applicative) 45, 70, 72, 73, 84, 88, 92

AUC Area Under the ROC Curve (Aire Sous la Courbe ROC) 105, 106, 108, 110, 112, 114–119, 121, 122, 126, 127, *voir* ROC

CISC Complex Instruction Set Computer (Processeur à Jeu d'Instrutions Étendu) 69

CNN Convolutional Neural Network (Réseau de Neurones Convolutif) 36–38, 40, 41, 131, 136, 137, 140

CPU Central Processing Unit (Unité Centrale de Traitement) 29, 33, 35, 41, 44, 71, 88, 90, 92, 93, 100, 124

FCN Fully Convolutional Network (Réseau Purement convolutif) 38, *voir* CNN

FLIM Fluorescence Lifetime Imaging Microscopy (Imagerie de Durée de vie de Fluorescence) 3, 5

FPGA Field-Programmable Gate Array 92, 93

FRAP Fluorescence Recovery After Photobleaching 3, 4, 12, 16

FRET Förster Resonance Energy Transfer 3, 9

GPU Graphics Processing Unit (Processeur Graphique) 29, 33, 35, 38, 41, 44, 92, 93, 124, 129, 131, 137, 140

HCS Hight Content Screening (Criblage à haut débit) 4, 5, 8, 9, 12, 13, 55, 138, 139

IA Intelligence Artificielle 44, 92, 136

IPC Inter-Process Communication (Communication Inter-Processus) 61, 72

MSE Mean Squared Error (Erreur Quadratique Moyenne) 125

OOB Out-Of-Bag 48–50, 111, 112

OPS Opération Par Seconde 44

R-CNN Region-based CNN 38, 40, 41, 140, *voir* CNN

RISC Reduced Instruction Set Computer (Processeur à Jeu d'Instrutions Réduit) 69

ROC Receiver Operating Characteristic 104, 105, 107, 109, 112, 113, 115, 117, 126, 128

ROI Region Of Interest (Région d'intérêt) 20, 36, 38, 39

SNR Signal-to-Noise Ratio (Rapport Signal sur Bruit) 12, 13

SPIM Selective Plane Illumination Microscopy 12, 17

STED Stimulated Emission Depletion 17

SVM Support Vector Machine (Machine à Vecteurs de Support) 28

Instituts et équipes

CeDRE Cell Division Reverse Engineering 4, 9, 137

IGDR Institut de Génétique et Développement de Rennes 4, 61, 136, 137

MFQ Microscopie de Fluorescence Quatitative 4, 9, 137

Première partie

Introduction

Chapitre 1

Introduction générale

1.1 Contexte

La microscopie de fluorescence est depuis plusieurs décennies un outil incontournable de la recherche fondamentale et appliquée en biologie, que l'on retrouve sous différentes formes (Nketia et al. 2017). Cette technique est basée sur une émission lumineuse consécutive à l'excitation d'un fluorochrome, et permet l'observation et la quantification fine de composants spécifiques d'échantillons biologiques. Une révolution dans ce domaine a été le développement des protéines fluorescentes pour marquer de manières endogènes des molécules ou structures d'intérêt et les suivre dans des échantillons vivants (Tsien 1998 ; Jin Zhang et al. 2002). Au-delà de l'acquisition d'images, de nombreuses méthodes de microscopie ont été développées autours de ce principe de fluorescence. Nous retrouvons par exemple :

— des méthodes de quantification comme le Fluorescence Lifetime Imaging Microscopy (FLIM, Imagerie de Durée de vie de Fluorescence) et le Förster Resonance Energy Transfer (FRET) ;
— des méthodes de photo-manipulation comme le Fluorescence Recovery After Photobleaching (FRAP), la photo-conversion ou la photo-activation ;

Ces méthodes permettent d'observer et de quantifier des phénomènes biologiques tels que l'activité des protéines, leurs interactions, mais aussi leur dynamique et leur diffusion au sein des échantillons biologiques. Les outils de photo-manipulation et photo-ablation restent aujourd'hui des outils délicats à maîtriser. Leurs configuration et utilisation nécessitent une certaine expertise et sont largement manuelles. Cela implique donc un faible débit d'images.

Le domaine de la microscopie optique a aussi connu beaucoup d'avancés permettant la réalisation de ces expériences. Cela inclut la robotisation des microscopes et de leurs périphériques, leur permettant d'être pilotés par des signaux électroniques. Ainsi les microscopes sont aujourd'hui contrôlés par ordinateurs, au travers de différentes méthodes de pilotages ayant été développées, permettant déjà leur automatisation pour l'acquisition de grandes quantités d'images d'échantillons vivants.

Cette automatisation des microscopes a ainsi rendu possible la réalisation de manipulations de type Hight Content Screening (HCS, Criblage à haut débit). Celles-ci permettent d'observer l'effet d'un grand nombre de molécules sur le phénotype des cellules, de manière non-supervisée. Cette technique est utilisée par exemple dans la recherche pharmaceutique, pour tester l'effet de médicaments sur des cellules vivantes.

À l'heure actuelle, l'utilisation de la robotisation des microscopes se limite aux manipulations de type HCS pouvant être réalisées de manière non-supervisées. En revanche, les méthodes manuelles de photo-manipulation (telles que le FRAP) ne peuvent pas être réalisées de manière non-supervisées, et n'exploitent pas le potentiel qu'apporte cette robotisation.

D'autre part, lors de l'étude d'évènements rares et courts, la pertinence des images acquises peut être aléatoire. Les méthodes actuelles consistent soit à :

— trier les images *a posteriori* pour ne conserver que les images d'intérêt ;
— utiliser des méthodes de synchronisation chimique des échantillons afin d'observer un maximum d'évènements d'intérêt (Banfalvi 2017).

Or, le tri *a posteriori* peut nécessiter l'acquisition d'un grand nombre d'images en fonction de la rareté des évènements d'intérêt. De même, les méthodes de synchronisation chimique sont connues pour être très invasives.

1.2 Motivations

Cette thèse CIFRE est réalisée en partenariat avec les équipes Cell Division Reverse Engineering (CeDRE) et Microscopie de Fluorescence Quatitative (MFQ) de l'Institut de Génétique et Développement de Rennes (IGDR) (*IGDR* 2020), ainsi qu'avec l'entreprise Inscoper (*Inscoper* 2020).

L'équipe de recherche fondamentale CeDRE s'intéresse aux mécanismes rendant la division cellulaire robuste et fidèle, en utilisant l'analyse d'images pour étudier la dynamique des microtubules et du fuseau mitotique. Plus particulièrement, la transition entre la métaphase et l'anaphase est un évènement transitoire d'intérêt lors de l'étude de ces mécanismes. L'équipe s'intéresse donc à la microscopie intelligente afin de pouvoir observer cet évènement sur des cellules humaines, dont la durée est courte par rapport à la durée du cycle cellulaire, sans avoir recours à la synchronisation chimique. Sur le plus long terme, l'intérêt de la microscopie intelligente sera de réaliser par la suite des expériences de photo-manipulation en déterminant l'instant précis (comme à la transition métaphase-anaphase) auquel une coupe d'un composant du fuseau mitotique doit être réalisée par exemple. Cette manipulation ne nécessitera donc plus d'expertise pour être correctement réalisée.

L'équipe MFQ développe des outils de microscopie de fluorescence quantitative, et s'intéresse particulièrement aux HCS pour réaliser des expériences à haut débit. Comme mentionné précédemment, les expériences de photo-manipulation sont à l'heure actuelle hautement supervisées. L'équipe souhaite donc pouvoir automatiser ces manipulations en déterminant l'instant et la position où elles doivent être effectuées de

manière non-supervisée. Ainsi, son objectif est de pouvoir intégrer ces expériences de photo-manipulation dans un contexte HCS entièrement autonome.

Enfin, Inscoper est une start-up commercialisant une solution optimisée de pilotage de microscopes. Cette solution prend en charge un grand nombre de modèles de microscopes et de périphériques, pour la réalisation d'acquisition multi-dimensionnelles, de microscopie de fluorescence quantitative (FLIM), ou encore de photo-manipulation par exemple. Afin de répondre aux limitations technologiques auxquelles les biologistes font face aujourd'hui, l'objectif d'Inscoper est de proposer un "Roboscope", permettant le contrôle non-supervisé des microscopes en analysant à la volée les images acquises.

1.3 Problématiques

La réalisation d'un microscope intelligent nécessite donc de pouvoir contrôler les modalités d'acquisition d'images de microscopie au même rythmes que les échantillons biologiques étudiés évoluent, cela de manière non-supervisée. Ainsi, l'objectif de cette thèse est de réaliser un prototype permettant l'analyse à la volée d'images, et le contrôle en temps-réel d'un système de microscopie en fonction de cette analyse. Celui-ci doit aussi être capable de détecter des images d'intérêt, de les sélectionner et de les envoyer vers un ordinateur afin de les stocker. Il doit être en mesure de prendre des décisions suite à la détection de ces images d'intérêt, afin de modifier le protocole d'acquisition et de capturer ces images suivant d'autres modalités. Ce prototype constitue une première étape vers le marché, et sera commercialisé par la suite par Inscoper. Une attention particulière est donc prêtée à cet aspect d'industrialisation tout au long de l'étude et du développement de ce prototype.

Le premier objectif de ces travaux de thèse est de concevoir un système informatique permettant l'asservissement d'un microscope. Cette étape consiste à réaliser une étude fonctionnelle théorique détaillée de ce système d'asservissement. Ce système doit être :

générique : utilisable pour un grand nombre d'applications ;

modulaire : évolutif et facilement modifiable ;

rapide : respectant des contraintes de temps pour observer des évènement rares et/ou courts.

Ce modèle théorique doit être implémenté sur un support matériel afin d'y être exécuté et testé, en considérant toujours ces différentes contraintes lors du choix du support et de sa programmation.

Enfin l'analyse d'images étant un élément critique en termes de temps d'exécution, notre second objectif est de concevoir une méthode d'analyse à la volée d'images de microscopie, dans le but d'identifier des objets ou des évènements d'intérêt. Cette méthode devant s'intégrer au système complet d'automatisation, elle doit aussi respecter des contraintes de généricité et de temps d'exécution, tout en présentant une bonne qualité de détection des objets d'intérêt.

1.4 Étapes de développement

Cette thèse est organisée en trois parties. Nous présentons dans cette partie d'introduction un état de l'art dans le chapitre 2, présentant des technologies d'automatisation des microscopes, ainsi que des algorithmes de détection d'objets et des systèmes embarqués permettant l'analyse d'images existant aujourd'hui. Nous y présentons aussi dans le chapitre 3 le matériel et les méthodes que nous utilisons au cours de nos travaux.

Nous présentons ensuite deux chapitres détaillant nos contributions lors de ce projet. Nous proposons dans un premier temps dans le chapitre 4, une solution fonctionnelle théorique de microscopie intelligente, que nous implémentons sur un système embarqué. Nous présentons une application biologique nous permettant de tester notre implémentation. Dans un second temps, nous présentons dans le chapitre 5 une méthode de classification d'images de cellules en culture respectant des contraintes de temps d'exécution.

Enfin, nous concluons cette thèse dans le chapitre 6, avec une synthèse de nos résultats, une discussion générale sur ces derniers, ainsi qu'une présentation de perspectives qu'ouvre ce projet.

Chapitre 2

État de l'art

2.1 Automatisation de la microscopie

2.1.1 Robotisation des microscopes

La microscopie optique, et plus spécifiquement la microscopie de fluorescence, est depuis plusieurs dizaines d'années un outil essentiel de la biologie pour l'étude de spécimens vivants. Cette technique repose sur l'utilisation de fluorochromes pour marquer des structures d'intérêt au sein des cellules. Lorsque ces fluorochromes sont excités par une source lumineuse, ils émettent à leur tour des photons (principe de fluorescence), permettant ainsi de traquer et de quantifier ces structures d'intérêt parmi la multitude de composants des échantillons biologiques, de manière peu invasive. Il existe une grande variété de méthodes de microscopie utilisées dans différents domaines de la biologie (Nketia et al. 2017 ; Sanderson et al. 2014 ; Baroux et Schubert 2018), permettant par exemple l'étude *in vivo* d'échantillons biologiques. C'est pourquoi ces systèmes ont connu une forte évolution ces dernières décennies.

Les systèmes de microscopie de fluorescence sont des outils sophistiqués. Ils sont généralement composés de différentes parties, à savoir d'un microscope à proprement parler (statif) auquel sont connectés des périphériques tels que des sources de lumière, caméras, etc. Au-delà des évolutions qu'a connues la microscopie sur le plan optique, les acteurs principaux de ce milieu, que sont Leica, Nikon, Olympus et Zeiss proposent aujourd'hui des systèmes de microscopie dont les accessoires sont de plus en plus motorisés (condenseurs, roues de filtres, platine, focus, etc) (Abramowitz et al. 2020). Cette robotisation des microscopes et de leurs périphériques permet de les piloter via des signaux électroniques, tels que des signaux USB, série, des signaux analogiques, etc.

Des outils ont par conséquent été développés pour tirer parti de la motorisation des microscopes, afin de pouvoir programmer des séquences d'acquisition d'images (pouvant impliquer une longue suite de mouvements des périphériques, ne pouvant être réalisés à la fois manuellement et rapidement). Ils permettent d'acquérir chaque image avec des paramètres différents, définis par l'utilisateur, suivant des dimensions temporelles et spatiales. La plupart de ces outils logiciels sont actuellement exécutés sur des ordinateurs dont le rôle est de piloter les différents éléments du microscope en suivant la séquence

d'acquisition définie par l'utilisateur (voir figure 2.1) (Tischer et al. 2014). Au-delà des logiciels fournis par les fabricants de microscopes, ils existent aussi des solutions de sociétés tierces telles que MetaMorph [1] (*MetaMorph* 2020), ou µManager, ce dernier logiciel étant une solution *open source* (*Micro-Manager* 2020 ; Edelstein et al. 2014).

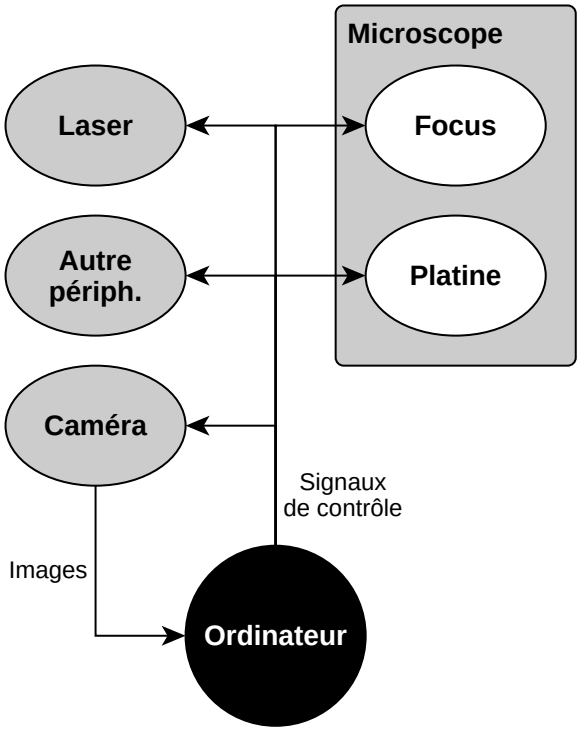


Figure 2.1 – Pilotage d'un système de microscopie par ordinateur générant différents signaux de contrôles (USB, série, signaux analogiques, etc). Le système est composé d'un statif (microscope) comprenant plusieurs composants motorisés (platine, focus, etc), ainsi que différents périphériques (*périph.*), tels que des sources de lumière, filtres, etc.

Dans un contexte d'acquisition rapide d'images (pour observer des objets dynamiques par exemple) ou de HCS, nous pouvons avoir des contraintes de temps importantes pour piloter le microscope. Certains évènements transitoires peuvent être par exemple de l'ordre de la minute, voir de la seconde. Or, les logiciels de pilotage exécutés sur ordinateur (voir figure 2.1) ne sont pas adaptés à un contrôle des microscopes avec des contraintes de temps de l'ordre de la seconde. En effet, les ordinateurs sont des appareils à usage général, fonctionnant souvent sous le système d'exploitation Windows, et ne sont pas dédiés uniquement au pilotage des microscopes. L'exécution de plusieurs tâches en parallèles (comme le gestionnaire de bureau Windows) autres que le logiciel de pilotage peut ralentir

1. `https://www.moleculardevices.com/products/cellular-imaging-systems/acquisition-and-analysis-software/metamorph-microscopy`

ce dernier. Cela créé des délais entre les différents mouvements du microscope, autres que les temps liés à ses contraintes mécaniques.

Suite au développement par les équipes CeDRE et MFQ, d'une méthode de contrôle efficace d'un ensemble de modules avec un système embarqué (microcontrôleur), ayant donné lieu au dépôt d'un brevet (Roul, Tramier et Pecreaux 2019), l'entreprise Inscoper a été créée dans le but de valoriser cette technologie en en faisant un produit industriel commercialisable (Roul, Bouchareb et al. 2017). L'entreprise commercialise un système dédié (système embarqué) à la génération efficace des signaux électroniques permettant de piloter les éléments d'un microscope, comme illustré dans la figure 2.2 (*Inscoper* 2020). Ce système embarqué peut être configuré pour prendre en charge différents modèles de microscopes. Avec cette solution, les rôles de l'ordinateur ne sont plus que d'afficher une interface utilisateur (permettant de configurer des expériences) et de stocker les images reçues depuis la caméra à laquelle il est relié. Ainsi, le système embarqué n'a pour tâche que le pilotage du microscope et des périphériques auxquels il est connecté, lui évitant d'être interrompu et ralentit par d'autres tâches, contrairement à un ordinateur. Cela a permis d'accélérer le débit d'image d'un facteur 3 par rapport aux logiciels exécutés sur ordinateur (*Inscoper* 2020).

Au-delà des aspects de rapidité, nous avons observé une meilleure stabilité de fonctionnement du microscope, lorsque le pilotage de ses périphériques est pris en charge par un système dédié. En effet, lors de l'utilisation d'un ordinateur, il est nécessaire d'utiliser des pilotes spécifiques à l'utilisation de chaque périphérique. Or, il est notoire dans la communauté des microscopistes que certains de ces pilotes présentent des problèmes de compatibilité. Ils sont aussi souvent dépendants de la version de Windows installée sur l'ordinateur. L'utilisation d'un système dédié pour gérer la génération des signaux de pilotages des éléments du microscope, permet de s'affranchir de ces contraintes (*Inscoper* 2020).

L'automatisation de l'instrumentation des microscopes a permis le développement des techniques de HCS. Il s'agit de techniques utilisées dans la recherche pharmaceutique, consistant à acquérir des images à moyen ou haut débit, dans différents contextes biologiques (Zanella, Lorens et Link 2010). Elles sont utilisées par exemple pour tester individuellement un grand nombre de composés chimiques, selon une expérience simple et standardisée pour une pathologie cible, en utilisant des plaques multi-puits (voir figure 2.3). Cela permet de générer une liste de candidats pour la création de médicaments (Perlman et al. 2004 ; S. Singh, Carpenter et Genovesio 2014 ; Thomas 2010 ; Esner, Meyenhofer et Bickle 2018). Le fait de pouvoir programmer de longues séquences d'acquisition suivant différentes dimensions (en coordonnées X et Y pour observer les puits d'un plaque multi-puits par exemple) se prête donc bien à ce genre d'application. Les progrès en termes de vitesse et d'efficacité de pilotage des microscopes ont par exemple permis le développement d'une méthode automatisée de criblage de l'activité de la protéine kinase AURKA avec un bio-senseur FRET (méthode développée au sein de l'équipe MFQ) (Sizaire et al. 2020).

En revanche, même si la robotisation des microscopes rend possible la réalisation de manipulation de type HCS de manière automatique et non-supervisée, la plupart des

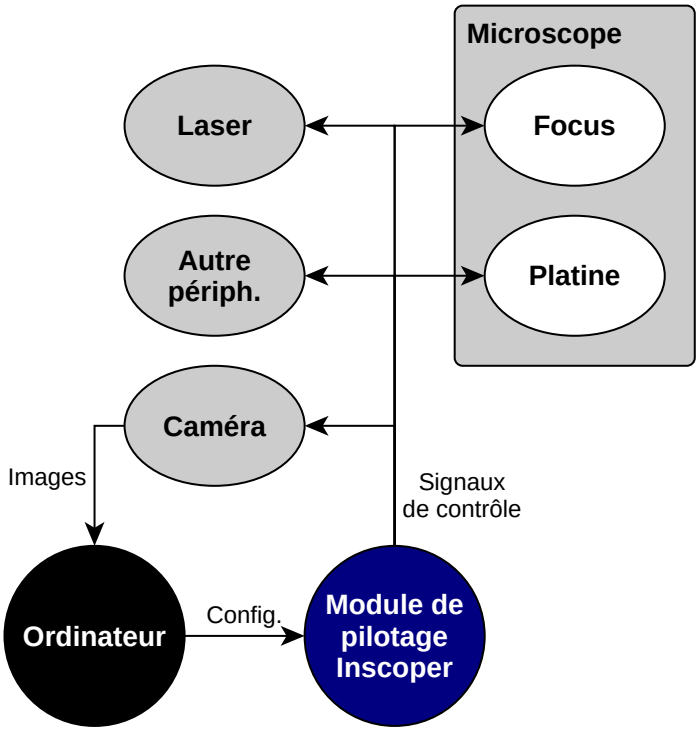


FIGURE 2.2 – Pilotage d'un système de microscopie via la solution Inscoper (*Inscoper* 2020), générant différents signaux de contrôles (USB, série, signaux analogiques, etc). Le système est composé d'un statif (microscope) comprenant plusieurs composants motorisés (platine, focus, etc), ainsi que différents périphériques (*périph.*), tels que des sources de lumière, filtres, etc.

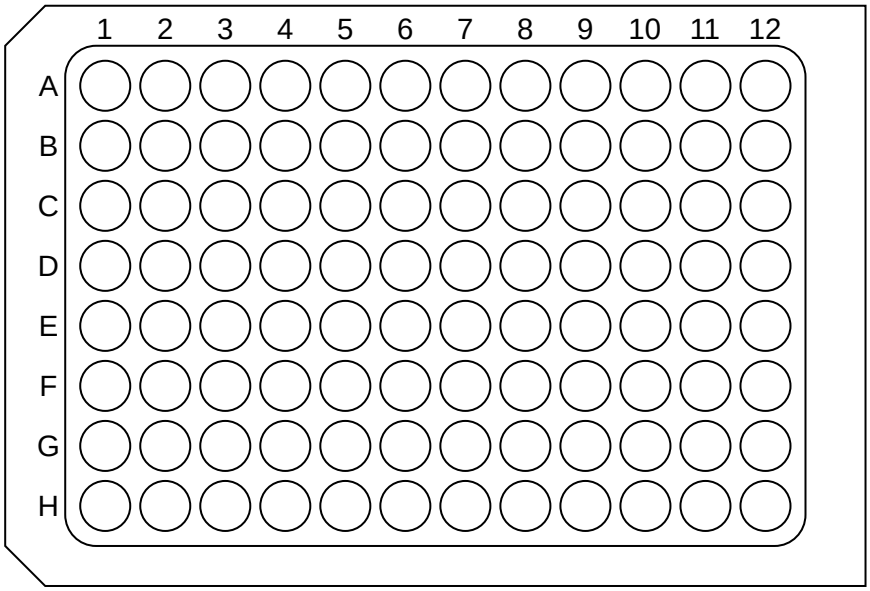


FIGURE 2.3 – Schéma de l'organisation d'une plaque 96 puits. Chaque cercle correspond à un puits contenant les échantillons à observer suivant un contexte biologique. Les puits sont organisés en lignes et colonnes afin de pouvoir être observé en déplaçant la plaque en X et Y.

techniques de photo-manipulation sont largement supervisées. C'est le cas par exemple des expériences de FRAP, avec lesquelles il faut choisir le moment et l'endroit où photo-blanchiment doit être réalisé. En effet, de telles expériences doivent impérativement être conduites de bout en bout par des experts. Ainsi, l'automatisation des expériences de photo-manipulation permettra d'envisager leur utilisation dans un contexte non-supervisé de HCS, et de donner la possibilité à des non-experts d'utiliser ces techniques, en laissant le microscope déterminer à leur place les moments et les endroits où les photo-manipulations doivent être réalisées.

2.1.2 Contraintes des méthodes traditionnelles de microscopie de fluorescence

Les marqueurs fluorescents sont aujourd'hui des éléments incontournables pour observer des structures spécifiques d'échantillons biologiques vivants, de manière peu invasive (Tsien 1998 ; Jin Zhang et al. 2002 ; Cole 2014). Comme tout fluorochrome, les protéines fluorescentes nécessitent une excitation lumineuse pour émettre un photon. Or le dosage de l'intensité de la source lumineuse d'excitation est un facteur non-négligeable lors de l'observation des échantillons. En effet, une trop forte dose de lumière peut conduire à l'annihilation de la capacité fluorescente des marqueurs (appelé photo-blanchiment), voir à la destruction de l'échantillon (Cole 2014 ; Esner, Meyenhofer et Bickle 2018).

Il existe aujourd'hui des solutions permettant d'acquérir des images en haute résolution (au-delà des limites qu'impose la microscopie en champ large, dues à la diffraction de la lumière visible), et exposant les échantillons à une illumination moins nocive que certaines autres méthodes de fluorescence (Baroux et Schubert 2018). Nous pouvons citer par exemple la méthode de microscopie de fluorescence à feuille de lumière, aussi appelée Selective Plane Illumination Microscopy (SPIM) (Voie, Burns et Spelman 1993 ; Huisken et al. 2004). Bien que cette méthode offre une solution au problème d'éclairage des échantillons, celle-ci peut générer une très grande quantité de données (plusieurs TB/jour) (Huisken et al. 2004 ; Scherf et Huisken 2015). Pour des applications générant une telle quantité de données, il est donc nécessaire de trier ces images *a priori*, dés leur acquisition, et non plus *a posteriori* comme il est d'usage de le faire actuellement.

En plus de la contrainte l'illumination, trois autres aspects techniques majeurs sont à prendre en compte lors de l'acquisition d'images :

- la résolution spatiale ;
- la résolution temporelle ;
- la qualité de l'image, définie par son Signal-to-Noise Ratio (SNR, Rapport Signal sur Bruit).

Ces trois aspects sont antagonistes, comme illustré dans la figure 2.4 (Scherf et Huisken 2015 ; Eisenstein 2020).

Un moyen d'augmenter le SNR d'une image consiste à capturer plus de photons en provenance de l'échantillon. Cela nécessite soit de l'éclairer de manière plus intense (ce qui peut être nocif), ou d'augmenter le temps d'exposition [2] du capteur de la caméra.

2. Le temps d'exposition correspond au temps durant lequel le capteur de la caméra capture des photons.

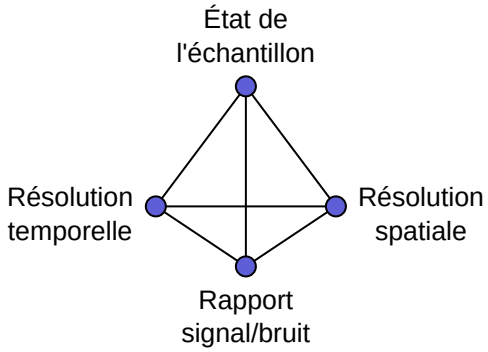


FIGURE 2.4 – Compromis entre les photo-dommages subits par l'échantillon (état de l'échantillon), les résolutions spatiale et temporelle, et le rapport signal sur bruit (SCHERF et HUISKEN 2015).

Cela implique donc d'augmenter le temps d'acquisition des images, et donc de dégrader la résolution temporelle.

Afin d'augmenter la résolution temporelle d'une acquisition, une pratique courante en microscopie consiste à réduire le nombre de pixels à acquérir, et donc de dégrader la résolution spatiale. Pour cela, deux méthodes consistent à :

— capturer uniquement une sous-région du champ de la caméra ;
— fusionner des pixels adjacents (binning) lors de la capture des photons.

Du fait de l'acquisition d'un plus grand nombre de photons par pixel, cette dernière méthode permet aussi d'augmenter le SNR.

Actuellement, ces différents compromis sont figés au début de l'expérience au moment du choix des modalités d'acquisition par l'utilisateur du microscope. En permettant la modification de ces modalités en cours d'acquisition grâce à un système de microscopie intelligent, il est possible d'adapter ces différents compromis aux besoins de l'expérience.

2.1.3 Vers la microscopie intelligente

Les limites des méthodes traditionnelles de microscopie optique, telles que le HCS ou les techniques de photo-manipulation supervisées, suscitent l'intérêt des chercheurs pour la microscopie intelligente, dans divers champs d'application. Des études montrent en effet que la complexité de la structure et du développement des spécimens vivants nécessite une évolution de notre manière de les observer, et que les méthodes traditionnelles d'acquisition ne suffisaient plus (SCHERF et HUISKEN 2015 ; EISENSTEIN 2020).

Les microscopes sont motorisés en X, Y et Z, de la même manière que la plupart des éléments qui les composent (roues de filtres, sélection des objectifs, etc). Cela permet leur pilotage par ordinateur et/ou par des modules dédiés, tels que celui développé par Inscoper, en suivant des séquences définies par l'utilisateur du microscope. En revanche, ces séquences sont aujourd'hui figées après leur configuration, ce qui ne permet pas de tirer pleinement parti du potentiel de cette motorisation des microscopes. L'idée de rendre

la microscopie intelligente consiste donc à analyser des échantillons, et à pouvoir modifier les modalités d'acquisition d'images au cours de l'expérience réalisée, en fonction de ces analyses. Cela revient à créer une boucle de rétrocontrôle, permettant d'adapter les compromis imposés par la microscopie (voir figure 2.4) en fonction des échantillons analysés (Scherf et Huisken 2015). Un premier exemple simple d'automatisation faisant intervenir une boucle de rétroaction est l'autofocus. Tout comme les caméras grand public que l'on retrouve par exemple sur les smartphones, les microscopes peuvent être équipés de fonctions d'autofocus (C.-Y. Chen, Hwang et Y.-J. Chen 2010). Cet outil permet d'éviter l'acquisition d'un grand nombre d'images floues devant être supprimées *a posteriori*.

Un des systèmes de microscopie intelligente se rapprochant le plus de ce que nous souhaitons réaliser a été développé et publié par Conrad et al. en 2011 (Conrad et al. 2011). Nous présentons un diagramme de flux décrivant le fonctionnement simplifié de Micropilot dans la figure 2.5a. Ce système consiste à analyser des images de cellules humaines HeLa Kyoto, acquises en basse résolution spatiale (acquisition plus rapide qu'en haute résolution spatiale, comme présenté dans la section 2.1.2), afin de détecter des cellules en prophase ou anaphase (deux phases transitoires de la division cellulaire) avec une sensibilité de 79,3 % et une précision de 87,9 %. Une fois une cellule d'intérêt détectée, le système peut modifier les modalités d'acquisition afin de réaliser des images en haute résolution et en 3 dimensions, ou réaliser de la photo-manipulation (imagerie complexe) (Conrad et al. 2011). L'intérêt d'un tel système est de pouvoir classifier les images dès leur acquisition. Les images de haute résolution sont acquises uniquement pour des objets d'intérêt, ce qui permet de largement réduire leur nombre et d'éviter une exposition trop importante des cellules aux sources de lumière.

En revanche, une limitation du système Micropilot est son fonctionnement séquentiel, que nous illustrons dans la figure 2.5b. En effet, dans la modalité à basse résolution, l'acquisition d'une nouvelle image n'est déclenchée qu'après avoir terminé l'analyse de la précédente. Ainsi, lors de cette analyse, le microscope est immobile. Il commencera son déplacement vers la prochaine position une fois l'analyse terminée, si aucune cellule d'intérêt n'a été détectée. Ce temps d'attente est plus important lors de l'utilisation de Micropilot avec un microscope de la marque Leica. Dans ce cas-ci, Micropilot attend d'avoir détecté un certain nombre de cellules d'intérêt, avant de déclencher la modalité complexe d'imagerie (Conrad et al. 2011).

Dans un contexte où nous étudions des évènements rares et courts, ces délais peuvent représenter une perte de temps significative, pouvant impacter la qualité des résultats obtenus. Le temps de réaction du microscope est aussi important que sa précision lors de la détection de ces évènements. Dans le cas d'évènements courts (quelques minutes par rapport à un cycle cellulaire pouvant durer 24 heures pour des cellules humaines par exemple), un temps trop important entre le moment où une image de cet évènement est capturée et le moment où le microscope change de modalité d'acquisition, peut provoquer une perte d'informations d'une partie voir de la totalité de l'évènement. Pour ce qui est des évènements rares (comme dans le cas de l'observation de défauts lors de la division cellulaire), il est nécessaire d'acquérir un nombre suffisamment important de

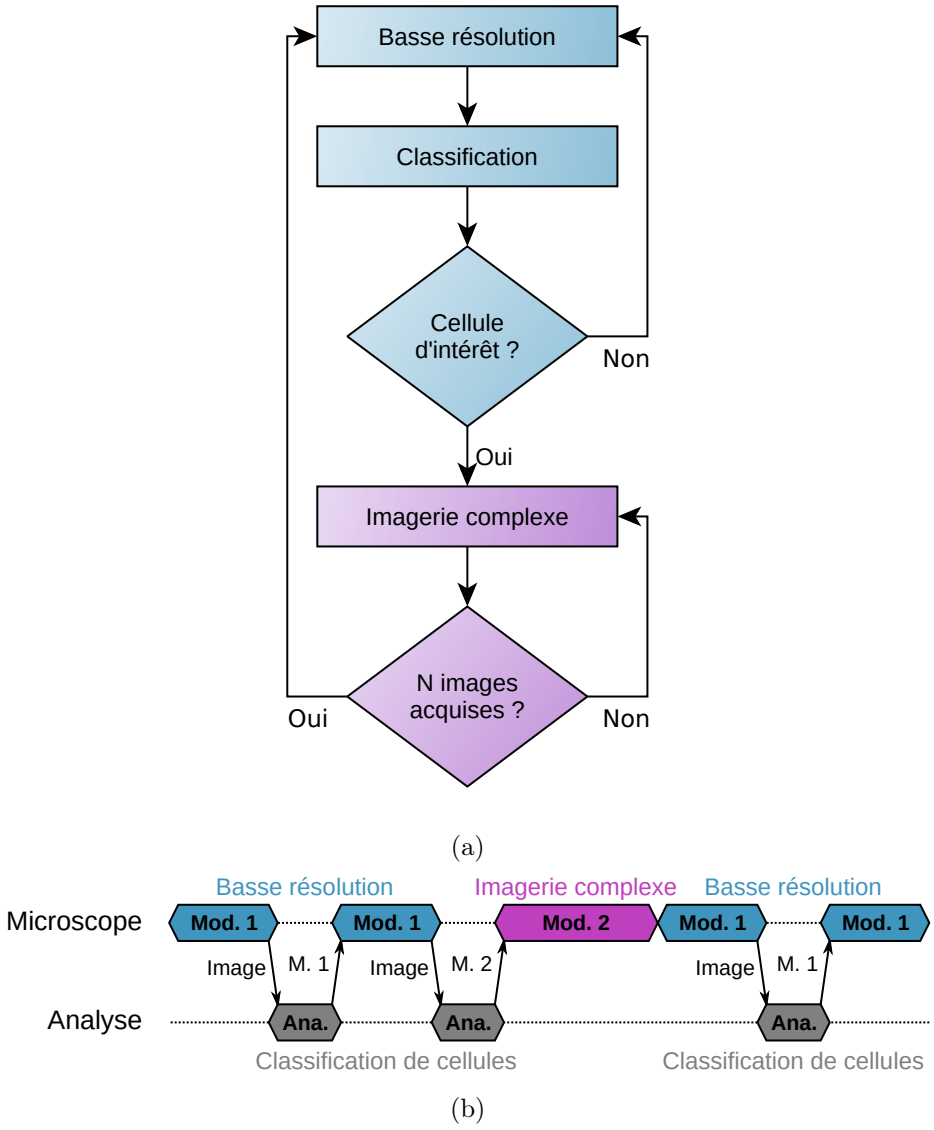


FIGURE 2.5 – Diagramme de flux schématisant le fonctionnement de Micropilot (a) (CONRAD et al. 2011), et diagramme temporel schématisant son fonctionnement séquentiel (b). Micropilot fonctionne suivant deux modalités d'acquisition des images : une modalité à basse résolution (Mod. 1 en bleu) et une modalité d'imagerie complexe (Mod. 2 en violet). Seules les images de la première modalité sont analysées (Ana.), afin de chercher des cellules d'intérêt (en prophase ou anaphase). La modalité d'imagerie complexe s'arrête après l'acquisition d'un certain nombre d'images (N) défini par l'utilisateur.

ces évènements, afin d'éviter tout biais statistique lors de leur étude. Une microscopie trop lente peut donc réduire le nombre d'évènements observés. L'étude de ces évènements nécessite ainsi une microscopie intelligente fonctionnant en temps-réel [3] (Stankovic 1988), devant analyser les images et contrôler les modalités d'acquisition au même rythme que les éléments biologiques que l'on souhaite observer.

D'autres projets de microscopie intelligente ont été développés ces dernières années en plus du projet Micropilot (Eisenstein 2020). Nous pouvons tout d'abord citer les travaux réalisés par Tischer et al., présentant deux solutions de microscopie intelligente, chacune développée pour un système de microscopie et une application (Tischer et al. 2014), que nous résumons dans la table 2.1. De manière générale, le principe de fonctionnement de ces solutions est le même que celui que nous présentons dans la figure 2.5a.

Application	Imagerie automatique en haute résolution de cellules infectées	Photo-blanchiment automatique sur des orgarnelles
Microscope	Leica SP5	Zeiss LSM 780
Logiciel de pilotage	Leica LASAF	Zeiss ZEN 2010
Logiciel d'analyse	CellProfiler	ImageJ

Table 2.1 – Microscope et logiciels utilisés pour les solutions de microscopie intelligente de Tischer et al. (Tischer et al. 2014).

La première solution réalisée par Tischer et al. a pour but d'observer des cellules infectées par un parasite (Plasmodium), en haute résolution, avec un microscope Leica SP5 piloté par son logiciel dédié Leica LASAF (*Leica-Microsystems* 2021). À l'échelle d'un tissu cellulaire, ces cellules sont rares, et difficiles à observer manuellement. Cette solution consiste donc à analyser automatiquement des images en basse résolution avec le logiciel CellProfiler (Carpenter et al. 2006), afin de détecter ces cellules infectées au sein d'un tissu. Ainsi, les images en haute résolution ne sont acquises que dans ces cas-ci, avec une fréquence d'environ 50 parasites par heure (Tischer et al. 2014).

Le deuxième système de microscopie intelligente réalisé a pour but d'effectuer des expériences de FRAP de manière non-supervisée sur des réticulums endoplasmiques, avec un microscope Zeiss LSM 780 et son logiciel de pilotage dédié Zeiss ZEN 2010 (*Zeiss* 2021). En microscopie classique supervisée, ces manipulations peuvent être difficiles à réaliser, du fait des mouvements de cette organelle, pouvant la faire sortir du champ du microscope. De plus, elle ne permet pas l'acquisition d'un grand nombre d'images. Le système de microscopie intelligente développé dans ce cas-ci permet de détecter automatiquement les réticulums endoplasmiques, en utilisant une plaque 96 puits, et le logiciel ImageJ (Schindelin et al. 2012) pour analyser les images. Finalement, celui-ci permet de réaliser 100 expériences de FRAP par heure, avec une qualité similaire à des expériences faites manuellement (Tischer et al. 2014).

3. En informatiques, un système est dit temps-réel lorsqu'il contrôle un procédé physique suffisamment rapidement par rapport à la vitesse d'évolution de ce dernier.

Ces deux systèmes de microscopie intelligente ont été développés chacun pour une application et un système de microscopie spécifique. En effet, des logiciels de pilotage de microscope et d'analyse d'images différents sont utilisés pour chaque application. Il n'est donc pas possible de les intervertir, ni de les utiliser pour d'autres applications et/ou systèmes de microscopie, pour lesquels un redéveloppement serait nécessaire (Tischer et al. 2014).

Nous avons observé ce phénomène de spécificité de la microscopie intelligente à une application avec d'autres solutions. Nous citons par exemple une technique développée par He et Huisken, visant à améliorer la couverture des échantillons sur des images multi-vues acquises avec une méthode de SPIM (He et Huisken 2020). Cette technique consiste à analyser des images à la volée selon plusieurs angles, afin d'estimer le ou les angles permettant une observation optimale de l'échantillon. Elle permet aussi une réduction de la quantité d'images et de la photo-toxicité. Cette solution est en revanche spécifique à l'utilisation d'un système multi-vues rotatif (He et Huisken 2020).

Nitta et al. ont développé un système de tri de cellules automatisé, basé sur la classification à la volée d'images avec un algorithme d'apprentissage profond. Le résultat de cette classification est utilisé pour contrôler un système de tri de cellules, pour les diriger dans différents récipients. Il permet de trier environs 100 cellules par seconde dans deux récipients différents (Nitta et al. 2018). Comme dans le cas précédent, cette méthode est proposée pour cette application très spécifique.

Une méthodologie proposée par Durand et al. consiste en une méthode d'optimisation de paramètres d'acquisition, tels que la puissance de l'illumination ou la vitesse d'acquisition, durant le déroulement d'une expérience, afin d'augmenter la qualité d'images en super-résolution de type Stimulated Emission Depletion (STED) (Hell et Wichmann 1994). Ces paramètres sont calculés à partir de l'analyse d'images avec des réseaux de neurones convolutifs (Durand et al. 2018). Cette solution est focalisée sur l'augmentation de la performance et de la qualité de l'acquisition des images. Elle ne permet pas une automatisation complète d'un microscope pour rechercher par exemple des évènements rares.

Enfin, lors de l'édition 2019 de la conférence Quantitative BioImaging, Mahecic et Manley nous ont présenté une approche permettant d'adapter le débit d'acquisition d'images de microscopie en fonction de l'intérêt des objets présents dans le champ d'une caméra (Mahecic et Manley 2019). Cela permet d'augmenter la quantité d'images considérées comme intéressantes et d'éviter l'exposition des échantillons à une trop grande quantité de lumière.

Finalement, ces techniques permettent d'automatiser une ou quelques expériences de microscopie, en intégrant une analyse d'images à la volée et un rétro-contrôle du microscope en cours d'acquisition (Tischer et al. 2014 ; Mahecic et Manley 2019 ; He et Huisken 2020 ; Durand et al. 2018 ; Nitta et al. 2018 ; Eisenstein 2020). Elles sont donc très spécifiques à une application ou un domaine de la microscopie, et tout changement d'application et/ou de microscope nécessite aujourd'hui de redévelopper un système de microscopie intelligente.

Cela a suscité l'intérêt de nombreux scientifiques, pour qui la microscopie intelligente peut présenter une vraie solution à des problèmes biologiques complexes (Eisenstein 2020). Malgré leur spécificité, les techniques que nous venons de présenter ont pour élément commun une analyse à la volée d'images de microscopie, dont les résultats permettent une modification des paramètres ou modalités d'acquisition au cours de l'expérience réalisée. Ainsi, plutôt que redévelopper des systèmes spécifiques à une application biologique et un microscope, l'étape suivante de cette automatisation consistera à développer un système de microscopie intelligente plus générique. En se basant sur ce même mécanisme d'analyse à la volée d'images et de modification de paramètres d'acquisition, celui-ci couvrira un plus large panel d'applications biologiques, et pourra être utilisé peu importe le système de microscopie.

2.2 Analyse d'images pour la recherche d'évènements ou d'objets d'intérêt

En microscopie, les images représentent l'élément de mesure principale que retourne la caméra associée au microscope. De la même manière que les biologistes étudient les informations contenues dans ces images, il s'agit de l'élément d'entrée des outils existants de microscopie intelligente. Ici, un enjeu majeur consiste à détecter des événements ou objets d'intérêt au sein des images. Au même titre que la microscopie intelligente, de nombreuses méthodes d'analyse ont été développées pour faciliter la tâche fastidieuse de recherche manuelle de ces objets ou événements par les biologistes. Des algorithmes permettant d'analyser ces images ont donc été développés pour la microscopie mais aussi dans de nombreux autres domaines (Ali et al. 2016).

Nous pouvons citer l'exemple du milieu médical, dans lequel des outils de détection d'objets peuvent être utilisés comme aide au diagnostic de certaines maladies en radiologie (Chartrand et al. 2017), ou du cancer du sein suivant différentes modalités comme la mammographie, l'imagerie ultrasons, l'imagerie par résonance magnétique, la microscopie (histologie) ou l'imagerie infrarouges (Yassin et al. 2018). Nous pouvons aussi retrouver ce genre d'outil dans le domaine de la vidéo-surveillance, lors de l'analyse des activités humaines dans le but de détecter des situations à risques, des comportements dangereux ou des intrusions (Sargano, Angelov et Habib 2017 ; S. Zhang et al. 2017). Comme dans le cas de la biologie, ces domaines peuvent présenter des situations nécessitant de détecter des évènements rares (situations anormales) parmi un grand nombre d'objets ou d'évènements.

Une image peut contenir plusieurs objets. Le principe de la détection de ces objets repose donc sur l'obtention de deux informations :

— déterminer où se situe l'objet sur l'image ;
— déterminer la catégorie de l'objet.

Nous présentons dans cette section les concepts les plus utilisés aujourd'hui pour détecter et classifier les objets présents dans des images.

2.2.1 Méthode usuelle de détection d'objets

Une approche usuelle de détection d'objets consiste à décomposer le processus en trois étapes (voir figure 2.6) :

Segmentation : séparation des différents objets que contient l'image ;

Extraction de caractéristiques : calcul d'un ensemble de valeurs caractérisant l'objet ;

Classification : identification de chaque objet à partir de ses caractéristiques en l'assignant dans une catégorie (classe).

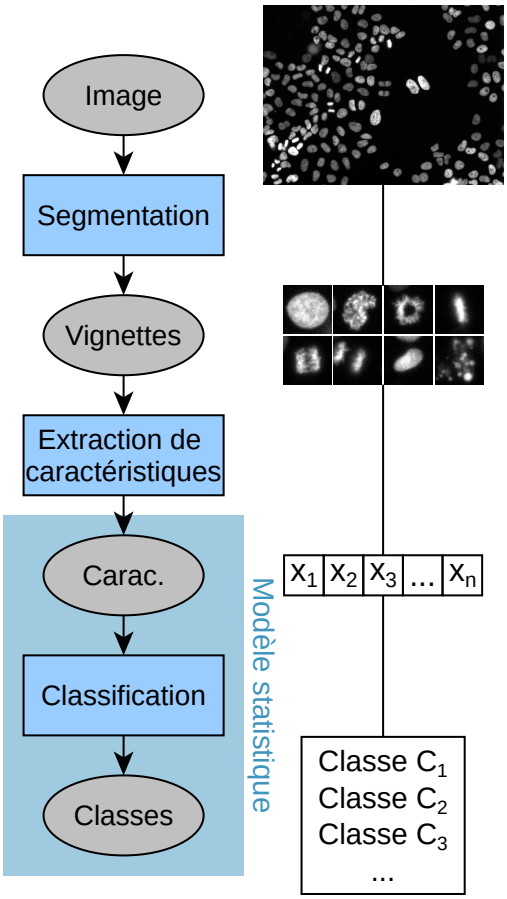


FIGURE 2.6 – Méthode usuelle de détection d'objets, incluant les étapes de segmentation d'une image en vignettes, d'extraction de caractéristiques décrivant chaque vignette et de classification de ces vignettes à partir de leurs caractéristiques.

Ce processus est utilisé dans un grand nombre d'applications biologiques comme la détection automatique de cellules en mitose :

— Harder et al. ainsi que Held et al. présentent des méthodes de détection, de traçage et de classification de cellules dans différentes phases du cycle cellulaire sur des images de fluorescence (Harder et al. 2009 ; Held et al. 2010) ;
— Haibo Wang et al. ainsi que Logambal et Saravanan présentent deux méthodes de détection et de comptage de cellules en mitoses pour le diagnostic de cancer (Haibo Wang et al. 2014 ; Logambal et Saravanan 2015).

La solution de microscopie intelligente Micropilot présentée dans la section 2.1.3 utilise cette même approche pour identifier des cellules d'intérêt (Conrad et al. 2011).

2.2.1.1 Segmentation

Lorsque des images contiennent plusieurs objets, l'objectif de la segmentation est de les séparer sous forme de régions que nous appelons Region Of Interest (ROI, Région d'intérêt), ou en détectant leurs bords. Un grand nombre de méthodes de segmentation ont été développées jusqu'à aujourd'hui (Nketia et al. 2017 ; Meijering 2012). La méthode la plus simple de segmentation consiste à utiliser des seuils afin de créer une image binaire séparant le premier plan de l'image de son fond. Cette solution convient à la segmentation d'images présentant une forte différence d'intensités entre le fond et le premier plan, les rendant facilement utilisables en microscopie de fluorescence (objets clairs sur un fond noir). Elle présente aussi l'avantage d'être rapide à exécuter (Nketia et al. 2017 ; Meijering 2012). En revanche, cette méthode présente des limitations lors de la segmentation d'images bruitées et/ou contenant des objets accolés. Elle est donc appropriée pour le premier développement d'une solution de détection d'objets.

2.2.1.2 Extraction de caractéristiques

Une fois une image contenant plusieurs objets segmentée en plusieurs images (vignettes) contenant un seul objet, dont on connaît les coordonnées sur l'image originale, il est possible de classifier chaque vignette dans différentes catégories afin d'identifier ce qu'elles représentent. Or, en fonction de leur taille, elles peuvent contenir un grand nombre de données (pixels), porteuses de peu d'information. Une approche usuelle consiste dans un premier temps à réduire le nombre de dimensions du problème de classification, en calculant un ensemble de valeurs permettant de décrire chaque vignette. Nous appelons cette étape extraction de caractéristiques. Il existe différentes manières de décrire ces vignettes :

intensité des pixels : calcul de statistiques (moyenne, minimum, maximum, median, etc) et/ou d'histogrammes sur l'intensité des pixels de l'images ;

morphologie de l'objet : calcul d'informations sur les dimensions et/ou la forme de l'objet contenu dans l'image ;

gradient et bords de l'objet : calcul de statistiques (moyenne, minimum, maximum, median, etc) et/ou d'histogrammes sur le gradient de l'image ;

textures : calcul d'informations sur la texture de l'image telles que les textures de Haralick (Haralick, Shanmugam et Dinstein 1973) et/ou les textures de Tamura (Tamura, Mori et Yamawaki 1978) ;

approximations polynomiales : calcul de coefficients de décomposition polynomiale de l'image tels que les moments de Zernike (Teague 1980).

Cette liste n'est pas exhaustive et présente les types de caractéristiques les plus utilisées pour classifier des images en biologie (Harder et al. 2009 ; Held et al. 2010 ; Haibo Wang et al. 2014 ; Logambal et Saravanan 2015 ; Orlov et al. 2008). Celles-ci peuvent être calculées directement sur des images ou vignettes brutes, mais aussi sur des images transformées comme dans la méthode proposée par Orlov et al. (Orlov et al. 2008).

Nous n'avons actuellement pas de méthode permettant d'estimer quelles caractéristiques sont les mieux adaptées à notre application. Il peut donc être intéressant de combiner plusieurs types de caractéristiques dans un même vecteur pour générer un grand nombre d'informations (Orlov et al. 2008).

2.2.1.3 Classification

Dans le contexte de la détection d'objets, l'étape classification des méthodes usuelles consiste à attribuer une classe ou une catégorie à une vignette, à partir des caractéristiques lui étant associées, comme illustré dans la figure 2.7. Cela permet ainsi de trier ces caractéristiques (et les vignettes à partir desquelles elles sont calculées) en fonction de leur contenu. En biologie, la classification de caractéristiques d'images permet par exemple d'identifier à quelle phase du cycle cellulaire est une cellule (Harder et al. 2009 ; Held et al. 2010).

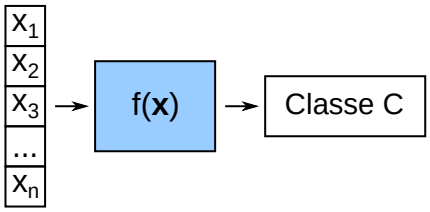


Figure 2.7 – Principe de la classification, où f est une fonction attribuant à un vecteur $\mathbf{x}$ (avec $\mathbf{x} = \{x_1; x_2; x_3; ...; x_n\}$ contenant n variables) une classe C.

La méthode la plus simple et la plus rapide pour résoudre un problème de classification consiste à utiliser une seule variable (ici une seule caractéristique d'image). En effet, il est possible d'attribuer différentes classes, à différentes valeurs ou intervalles de cette variable, à l'aide de seuils permettant de séparer les classes. Nous illustrons cette classification avec un exemple présenté dans la figure 2.8. Cet exemple consiste en la classification d'une variable selon trois classes. Pour chaque classe, cette variable suit une loi normale dont la moyenne dépend de la classe à laquelle elle appartient.

Cette méthode n'est efficace que si les valeurs de la variable sélectionnée sont différentes pour chaque classe. Nous disons dans ce cas que la variable est discriminante. Dans l'exemple de la figure 2.8, la variable permet de discriminer la classe 1 des deux autres classes. En revanche, si cette variable peut avoir un même ensemble de valeurs pour différentes classes, avec une probabilité élevée (comme les classes 2 et 3 dans la figure 2.8,

dont les densités de probabilités se chevauchent), l'attribution d'une classe devient alors difficile, et engendrer un grand nombre d'erreurs. Dans ce cas, l'utilisation d'une seule variable n'est pas suffisante.

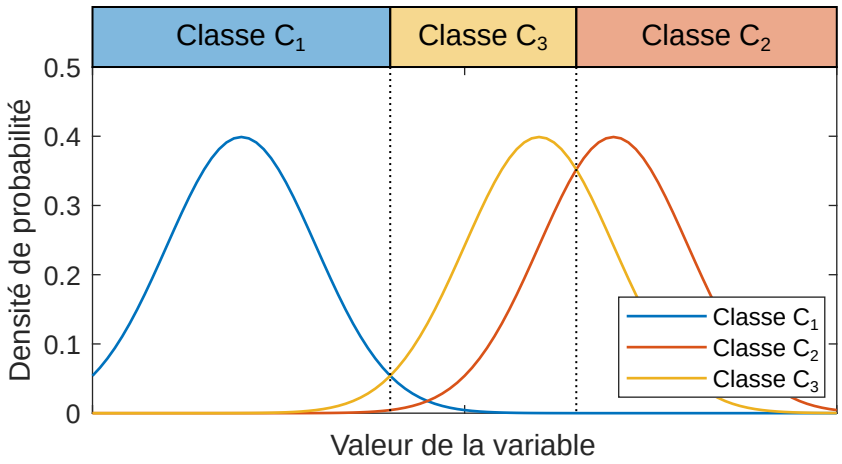


FIGURE 2.8 – Exemple théorique de classification avec une seule variable, suivant une loi normale dont la moyenne dépend de la classe réelle à laquelle elle est rattachée. Dans cet exemple, la déviation standard pour chaque classe est identique.

Lorsqu'une seule caractéristique d'image ne suffit pas à obtenir une classification suffisamment précise pour une application donnée, nous pouvons avoir recours à plusieurs caractéristiques sous forme d'un vecteur. Or, plus le nombre de caractéristiques utilisées est élevé, plus il difficile de les visualiser graphiquement et de créer des modèles théoriques permettant de les classifier. C'est pourquoi des méthodes statistiques d'apprentissage automatique se sont démocratisées ces dernières décennies, afin de traiter des problèmes comportant un grand nombre d'entrées. Ces méthodes reposent sur l'utilisation d'un jeu de données, appelé jeu d'apprentissage, pour créer un modèle statistique permettant de réaliser une tâche de classification ou de régression. Leur mise en œuvre comporte deux phases :

apprentissage : phase durant laquelle le modèle statistique est construit en "l'entraînant" sur le jeu de données d'apprentissage ;

utilisation : phase durant laquelle de nouvelles données sont fournies en entrée du modèle afin qu'il prédise une sortie en fonction de ce sur quoi il a été entraîné.

Il s'agit donc d'outils génériques, n'étant pas liés à une application particulière. Il peuvent ainsi être adaptés à une utilisation spécifique, en les configurant et en les entraînant avec des données similaires à l'application ciblée.

Parmi ces méthodes statistiques, l'Analyse en Composantes Principales (ACP) est un modèle très utilisé, dont le but principal est de réduire la dimension d'un jeu de données. Cette méthode repose sur le calcul de nouvelles variables (appelées composantes principales), à partir du jeu de données, et des vecteurs propres et valeurs propres de la matrice de covariance de ce dernier. Ces composantes principales sont orthogonales

(non-corrélées). De plus, la variance du jeu de données initial est maximisée sur un sous-ensemble de ces composantes. Ainsi, l'ACP permet de réduire la dimension d'un jeu de données, augmentant son interprétabilité, tout en minimisant la perte d'informations statistiques (Jolliffe 2002 ; Jolliffe et Cadima 2016).

Dans un contexte de détection à la volée d'objets, les outils de réduction de dimension sont utiles pour réduire le temps d'extraction des caractéristiques et de la classification. L'ACP peut être utilisée à cet effet, en utilisant un nombre réduit de composantes principales pour réaliser la classification (voir figure 2.9). De plus, une analyse des vecteurs propres obtenus permet d'estimer la contribution de chaque variable du jeu de données dans le calcul d'une composante principale. Les variables ayant une contribution faible (pour les composantes principales dont la variance est la plus importante) peuvent être supprimées.

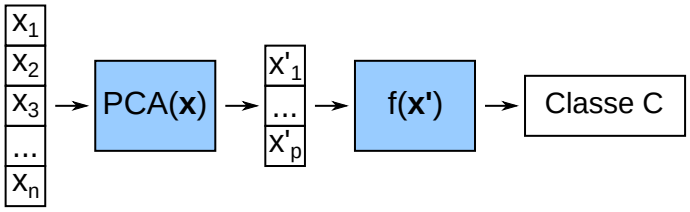
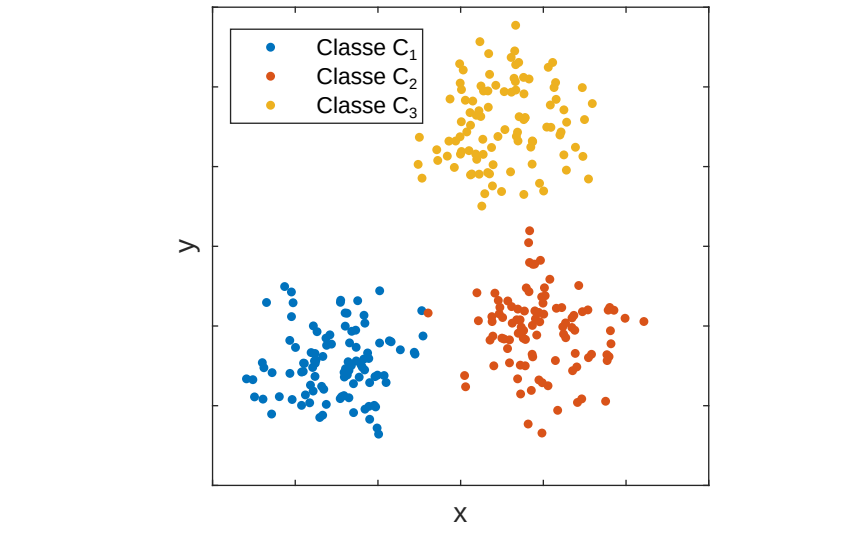


Figure 2.9 – Réduction de dimension avec une ACP. La fonction de classification f est appliquée sur un vecteur $\mathbf{x}'$ de dimension plus faible que celle de $\mathbf{x}$, afin de lui attribuer une classe C.

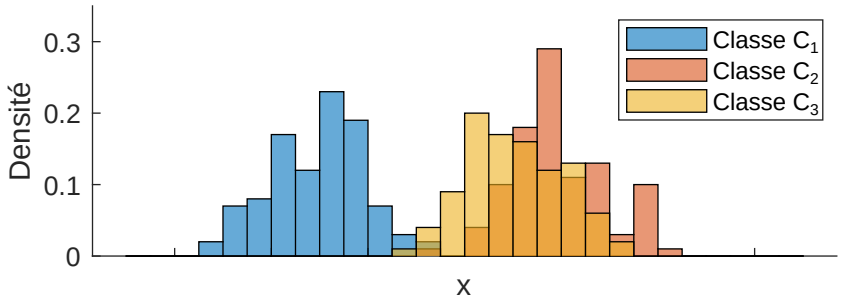
Du fait que l'ACP maximise la variance sur la première composante principale, il peut être envisagé de l'utiliser pour réduire le problème de classification à un problème à une variable. Nous présentons un exemple dans la figure 2.10, dans lequel deux variables sont utilisées pour résoudre un problème de classification à trois classes. De manière similaire à l'exemple de la figure 2.8, chaque variable suit une loi normale dont la moyenne dépend de la classe à laquelle elle est rattachée. De même, l'utilisation d'une seule de ces variables n'est pas suffisante pour réaliser une classification précise. Les figures 2.10b et 2.10c montrent en effet une forte probabilité que les variables aient la même valeur pour différentes classes. C'est pourquoi une ACP peut être utilisée pour réduire le problème de classification à une variable tout en conservant la variance entre les classes (voir figure 2.11).

L'ACP peut donc être une solution rapide pour classifier plusieurs variables. En revanche, ne s'agissant pas d'un outil de classification à l'origine, l'obtention d'une composante principale discriminante pour toutes les classes d'un problème n'est pas garantie. L'ACP étant calculée uniquement à partir de la covariance des variables, celle-ci ne prend pas en compte la présence de classes dans le jeu de données. Des variables ayant une variance plus élevée au sein des classes, par rapport à la variance entre celles-ci, peuvent rendre l'ACP non-utilisable pour directement résoudre un problème de classification.

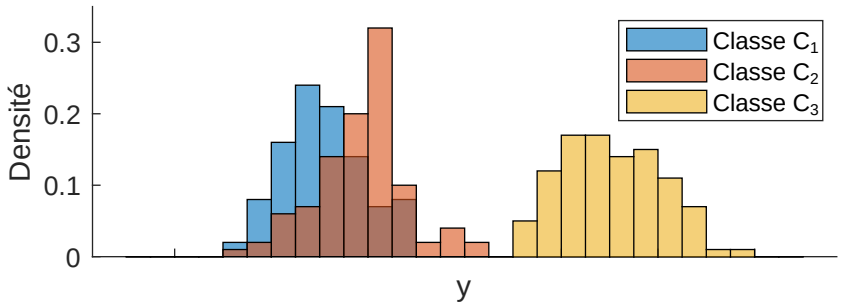
D'autres méthodes statistiques ont été proposées dans le but d'identifier des groupes, aussi appelés *clusters*, au sein d'un jeu de données. Dans le contexte de la classification



(a)

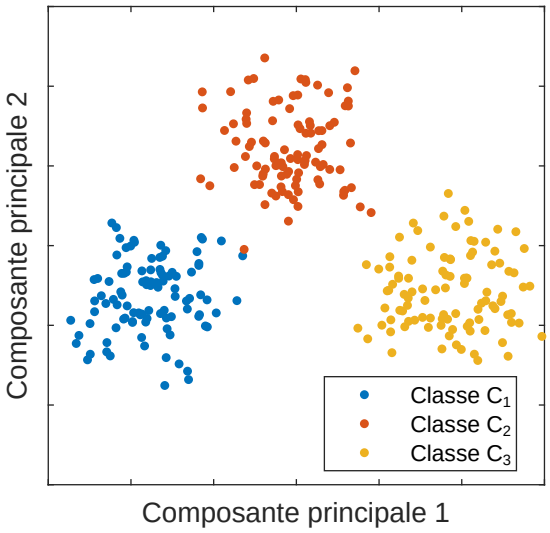


(b)

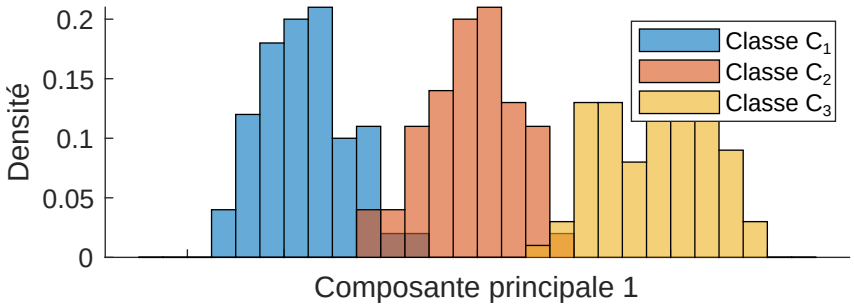


(c)

FIGURE 2.10 – Exemple de classification avec deux variables x et y suivant une loi normale, dont la moyenne dépend de la classe à laquelle elles sont rattachées. Dans cet exemple, la déviation standard pour chaque classe est identique. Les classes sont présentées sous la forme d'un nuage de points (a) et d'un histogramme pour x (b) et y (c).



(a)



(b)

FIGURE 2.11 – Exemple de classification de deux variables en utilisant l'ACP (a) suivant une loi normale, dont la moyenne dépend de la classe à laquelle elles sont rattachées. Dans cet exemple, la déviation standard pour chaque classe est identique. La meilleure composante principale (b) est utilisée pour la classification.

d'images, ces méthodes de *clustering* sont utilisées pour discerner les images appartenant à un même *cluster*, en utilisant leurs caractéristiques (Mann et Kaur 2013).

L'ACP ainsi que les méthodes de *clustering* sont des méthodes dites d'apprentissage non-supervisé. Ces modèles statistiques sont construits sans connaissance *a priori* des classes réelles auxquelles appartiennent les images et les caractéristiques leur étant associées. Seules les variables d'entrées (caractéristiques) sont nécessaires. Il existe aussi d'autres méthodes dites d'apprentissage supervisé, consistant à créer des modèles statistiques à partir d'un jeu de données annoté [4]. L'annotation des images et de leurs caractéristiques est réalisée manuellement, et peut être chronophage et laborieuse. L'accès à un grand nombre de données annotées peut donc être difficile. Malgré cela, du fait de la connaissance *a priori* des sorties de chaque donnée d'apprentissage, les méthodes d'apprentissage supervisé permettent aujourd'hui d'obtenir des résultats plus précis de classification que les méthodes d'apprentissage non-supervisé. Nous présentons dans la figure 2.12 une liste d'algorithmes d'apprentissage les plus utilisées à l'heure actuelle en classification.

De manière comparable à l'ACP, il existe des méthodes linéaires d'apprentissage automatique supervisé. Celles-ci consistent à utiliser une combinaison linéaire de l'ensemble des caractéristiques pour réaliser une classification, ou bien à calculer des hyperplans permettant de séparer les différentes classes.

Parmi ces méthodes, nous pouvons citer par exemple le discriminant linéaire. Son utilisation la plus simple consiste à résoudre un problème de classification à deux classes, en calculant une combinaison linéaire des caractéristiques en entrée. Celle-ci est représentée par l'équation 2.1, où $\mathbf{x}$ représente le vecteur de caractéristiques d'entrée, $\mathbf{w}$ représente un vecteur de poids et w_0 représente un biais (Bishop 2006, pp. 181-182). $y(\mathbf{x})$ est donc utilisée pour classifier le vecteur $\mathbf{x}$ dans une classe C_1 si $y(\mathbf{x}) \geq 0$, ou dans une classe C_2 si $y(\mathbf{x}) < 0$. L'objectif de l'apprentissage de ce modèle consiste donc à déterminer les poids $\mathbf{w}$ et w_0, de telle sorte que l'hyperplan obtenu avec l'équation $y(\mathbf{x}) = 0$ permette de séparer au mieux les classes.

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 \tag{2.1}$$

Le discriminant linéaire peut aussi être généralisé à des problèmes de classification avec plus de deux classes. Il existe aujourd'hui plusieurs méthodes se basant sur celle avec deux classes. L'une d'entre elles consiste par exemple à calculer une combinaison linéaire pour chaque classe (voir équation 2.2) (Bishop 2006, pp. 182-184). $\mathbf{x}$ est attribué à une classe C_k si l'équation $y_k(\mathbf{x}) > y_j(\mathbf{x})$ est vérifiée pour tout $k \neq j$.

$$y_k(\mathbf{x}) = \mathbf{w}_k^{\mathrm{T}}\mathbf{x} + w_{k0} \tag{2.2}$$

Afin de déterminer les paramètres $\mathbf{w}_k$ et w_{k0} permettant de séparer les différentes classes du problème, une méthode consiste à quantifier la discriminance [5] de $y_k(\mathbf{x})$.

4. Dans un jeu de données annoté, la classe réelle à laquelle appartient un ensemble de variables (ici les caractéristiques d'une image) est associée à ces dernières.

5. La discriminance est définie comme étant le rapport entre la variance inter-classes d'une variable, et la somme des variances intra-classes de cette variable (Bishop 2006, pp. 186-192).

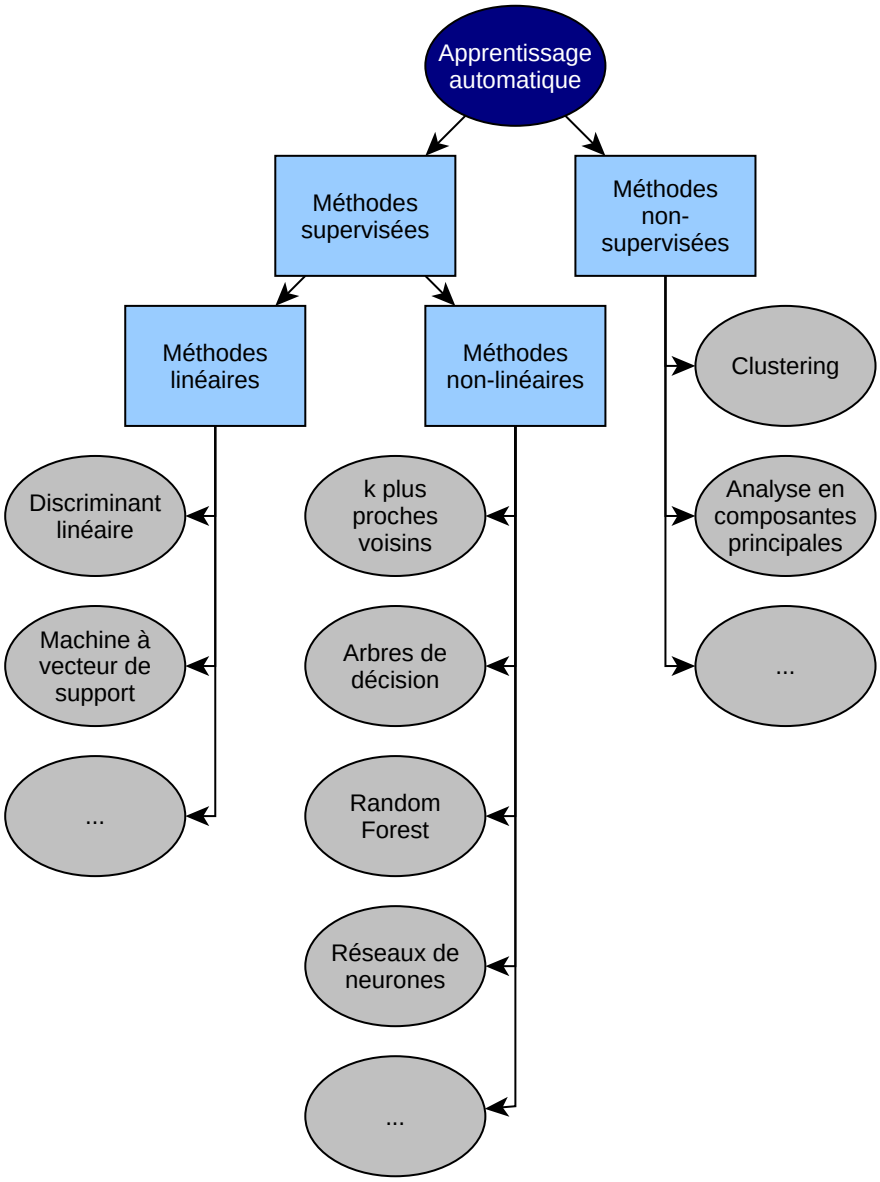


FIGURE 2.12 – Liste de méthodes populaires de classification (KOTSIANTIS, ZAHARAKIS et PINTELAS 2006).

Maximiser ce facteur en augmentant la variance entre les classes, tout en diminuant la variance à l'intérieur de chaque classe, permet d'éviter le recouvrement de plusieurs classes comme dans l'exemple que nous avons présenté précédemment dans la figure 2.8 (Bishop 2006, pp. 186-192).

Cette approche est similaire à l'ACP, dans le sens où la variance est utilisée comme critère pour calculer les paramètres de la combinaison linéaire des caractéristiques d'images. En revanche, contrairement à l'ACP, le discriminant linéaire prend en compte la classe à laquelle appartiennent les caractéristiques, permettant ainsi de les classifier avec une meilleure précision. Dans un contexte où le temps de classification des images est aussi important que la précision, ce critère de discriminance peut être utilisé dans un but de réduction de dimension, en identifiant les caractéristiques apportant moins d'informations pour la classification.

La Support Vector Machine (SVM, Machine à Vecteurs de Support) est une autre méthode permettant de déterminer un hyperplan de séparation des différentes classes d'un jeu de donnée. Elle consiste à calculer les paramètres de cet hyperplan, de telle sorte que la marge entre celui-ci et les points du jeu de données soit maximale. Cela permet à la SVM de réduire le risque de mauvaise classification lors de son utilisation avec de nouvelles caractéristiques. Elle a ainsi une meilleure capacité de généralisation d'un problème que le discriminant linéaire (Gokcen et Peng 2002).

Le discriminant linéaire et la SVM sont des méthodes de classification linéaires, dont le but est de séparer le jeu de données en différentes classes grâce à des hyperplans. Malgré leur simplicité en termes de calcul, pouvant rendre leur exécution rapide, l'utilisation de ces méthodes nécessite que les données soit linéairement séparable. Or, pour un grand nombre d'applications, les caractéristiques d'images peuvent représenter des phénomènes non-linéaires. Dans ce telles situations, les classes ne pouvant pas être séparées par des hyperplans, les méthodes linéaires ne permettent pas une classification précise. Une approche peut être de transformer un problème non-linéaire en un problème linéaire, en projetant les caractéristiques dans un espace dans lequel elles sont linéairement séparables. Cela nécessite en revanche de connaître *a priori* la fonction non-linéaire permettant cette transformation.

L'objectif de la classification étant d'attribuer une classe à ensemble de caractéristiques (ou une image), parmi un nombre de classes fini, les arbres décisions sont un outil très utilisé à cet effet. Dans un arbre de décision, les variables sont seuillées individuellement les une après les autres (Breiman et al. 1984). Les variables sélectionnées, ainsi que leur seuil, sont déterminés par apprentissage. Cela permet ainsi de diviser le jeu de données en différentes sous-régions, chacune appartenant à une classe.

Nous présentons dans la figure 2.13 un exemple de classification avec deux variables dans deux classes. Les données représentées dans la figure 2.13a ne sont pas linéairement séparables. Il est en revanche possible de diviser le jeu de données en sous-régions ne contenant que des points d'une même classe. Lorsque le seuillage d'une variable ne suffit pas à obtenir que des échantillons provenant d'une même classe (dans la figure 2.13a, lorsque $y \geq y_1$), une autre variable peut être utilisée pour diviser la sous-région

correspondante en deux autres sous-régions. Cela donne ainsi l'arbre de décision de la figure 2.13b dans le cas de notre exemple.

Contrairement aux méthodes linéaires présentées précédemment, qui utilisent uniquement des opérations arithmétiques linéaires, les arbres de décision utilisent des opérations conditionnelles. Dans un contexte d'optimisation en temps d'exécution, ces opérations sont difficiles à paralléliser (sur un Graphics Processing Unit (GPU, Processeur Graphique) par exemple). De plus, lors de leur utilisation sur un Central Processing Unit (CPU, Unité Centrale de Traitement) avec un système de pipeline [6], les conditions peuvent fréquemment provoquer un vidage de ce pipeline. Cela peut ainsi ralentir le CPU. Cette contrainte est importante lorsque l'arbre de décision comporte beaucoup de nœuds, ce qui peut être le cas lors de l'utilisation d'un grand nombre de variables. Ce phénomène peut être limité en imposant par exemple une limite de profondeur [7] de l'arbre de décision, lors de sa construction.

Les arbres de décision sont considérés comme une méthode de classification faible, du fait qu'ils présentent des problèmes de sur-apprentissage [8]. Il existe donc aujourd'hui des méthodes de classification consistant à combiner des algorithmes de base, appelées méthodes d'agrégation de modèles. Nous pouvons citer par exemples les méthodes de boosting telles que Adaboost (Freund et Schapire 1996 ; Schapire 2013), ou les méthodes de bagging, la plus populaire étant *Random Forest* (Breiman 2001). En revanche, du fait de leur utilisation de plusieurs algorithmes basiques, ces méthodes sont plus longues en termes de temps d'exécution.

Random Forest est une méthode de classification (et de régression) constituée d'un ensemble d'arbres de décision (Breiman 2001). Chacun d'entre eux a pour entrée le vecteur $\mathbf{x} = \{x_1; x_2; x_3; ...; x_n\}$ (qui est le même dans chaque cas), et a pour but de lui attribuer une classe. Ainsi, l'ensemble génère K classes (où K correspond au nombre d'arbres de décision de l'ensemble, qui est un paramètre défini par l'utilisateur avant l'apprentissage). Un système de vote est finalement utilisé par *random forest* pour attribuer une classe au vecteur $\mathbf{x}$, en sélectionnant celle ayant été attribuée par le plus d'arbres de décision. Nous schématisons ce principe dans la figure 2.14.

Chaque arbre de décision dans un *random forest* est entraîné avec différents sous-ensembles du jeu de données d'apprentissage. En effet, entraîner deux arbres de décision avec un même jeu d'apprentissage générerait deux arbres identiques, donnant chacun les mêmes résultats de classification. C'est pourquoi lors de l'apprentissage d'un *random forest*, certains échantillons et caractéristiques sont aléatoirement supprimées du jeu d'apprentissage, pour chaque arbre de décision. Ainsi, chaque arbre est entraîné avec un sous-ensemble différent du jeu de données (Breiman 2001). Nous schématisons ce procédé dans la figure 2.15.

6. Les CPU avec un pipeline d'instructions permettent l'exécution en cascade de plusieurs instructions.

7. La profondeur d'un arbre de décision représente le plus grand nombre de seuillages qu'il peut être amené à effectuer pour attribuer une classe à une entrée.

8. Le sur-apprentissage signifie qu'un modèle statistique d'apprentissage automatique échoue à généraliser le problème qu'il est censé résoudre, en apprenant les détails du jeu d'apprentissage. Par conséquent, il ne sera pas capable de classifier de nouvelles données avec une bonne précision.

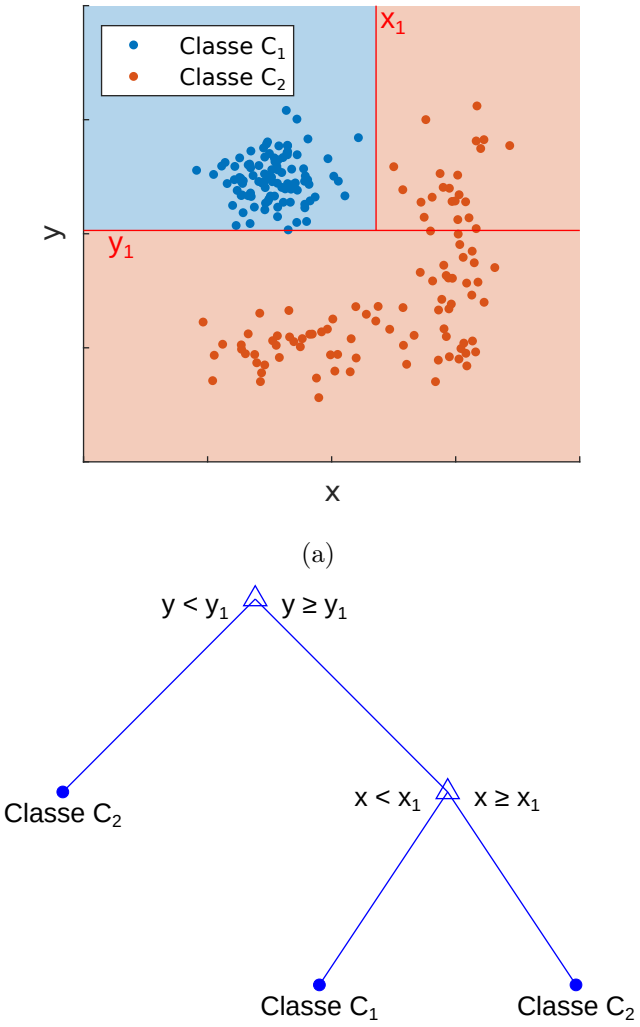


FIGURE 2.13 – Exemple de classification avec un arbre de décision. Le jeu de données comprend deux classes C_1 et C_2, représentées par deux variables x et y (a), non-linéairement séparables, chacune utilisée pour construire un arbre de décision dont les nœuds utilisent les constantes x_1 et y_1 comme seuil de décision sur les variables x et y respectivement (b).

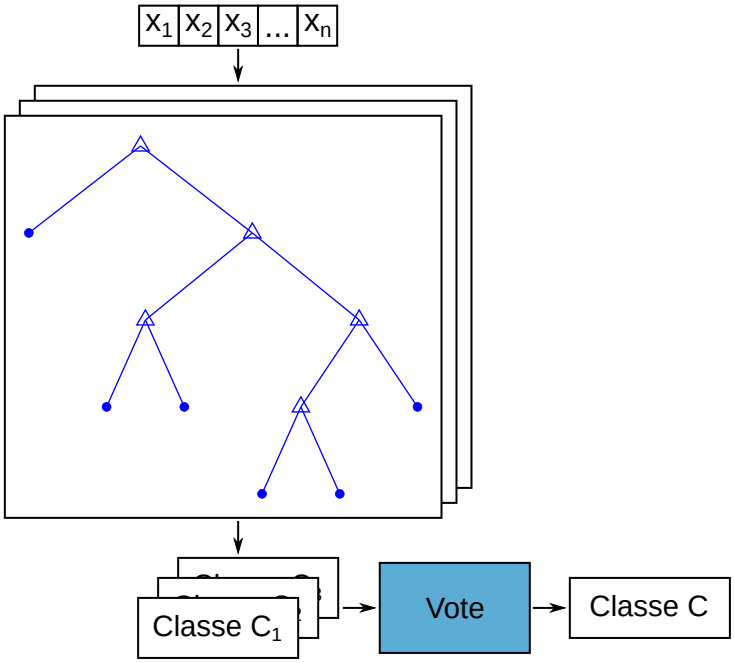


FIGURE 2.14 – Principe de fonctionnement d'un *random forest*, constitué d'un ensemble d'arbres de décision. Chaque arbre a pour entrée le vecteur $\mathbf{x}$, et lui attribue une classe C_k. La décision finale de *random forest* repose sur un système de vote par rapport à l'ensemble des classes attribuées par les arbres de décision, afin d'attribuer une classe C au vecteur $\mathbf{x}$.

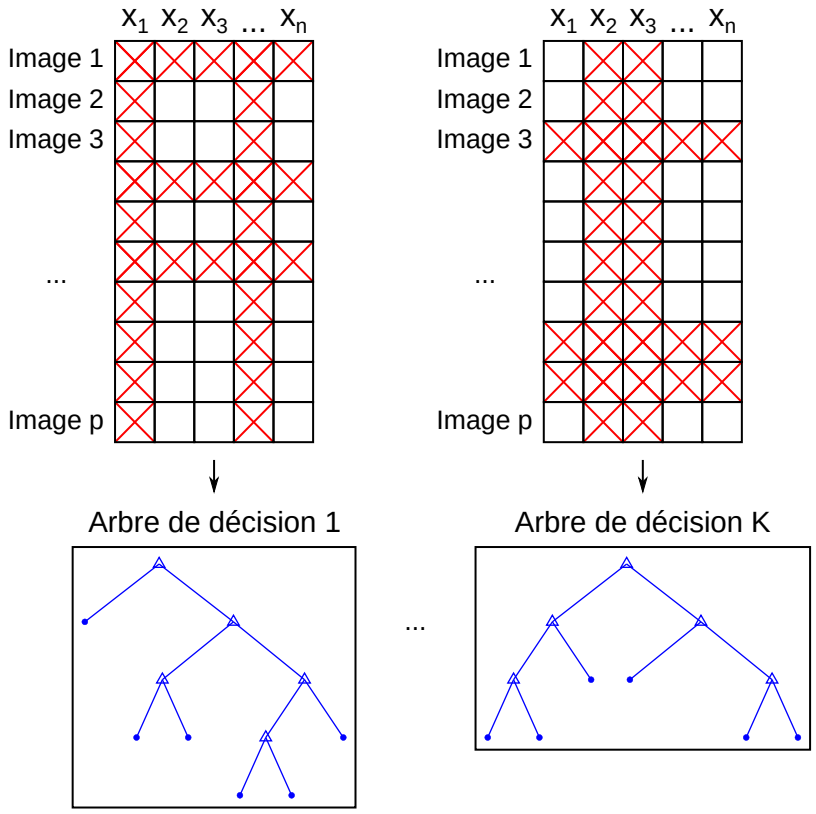


FIGURE 2.15 – Principe de l'entraînement d'un *random forest*, au cours duquel chaque arbre de décision est entraîné avec un sous-ensemble différent du jeu de données d'apprentissage. Pour chaque sous-ensemble, un certain nombre de caractéristiques x_i et d'images (à partir desquelles les caractéristiques sont calculées) sont aléatoirement supprimées.

Random Forest est une méthode très utilisée en classification du fait qu'elle soit généralement plus précise et rapide que les méthodes de boosting, en plus de ne pas souffrir de problème de sur-apprentissage (Breiman 2001). Le fait d'utiliser un ensemble d'arbres de décision, entraînés sur différents sous-ensembles du jeu d'apprentissage, lui confère aussi une meilleure capacité de généralisation que les arbres de décisions eux-mêmes. Bien que les arbres puissent être exécutés en parallèle sur plusieurs CPU, *random forest* souffre des mêmes limitations que les arbres de décision. En effet, il est difficilement envisageable d'accélérer une telle méthode sur un GPU, du fait de son utilisation intensive de conditions. Ces conditions peuvent aussi provoquer un grand nombre de vidages du pipeline des CPU.

Enfin, les réseaux de neurones sont une méthode de classification et de régression, ayant gagnée en popularité de manière très importante ces dernières années. Comme *random forest*, ils peuvent être considérés comme une méthode combinant plusieurs algorithmes de base du même type : les perceptrons (ou neurones). Nous illustrons le principe de fonctionnement des réseaux de neurones dans la figure 2.16. Chaque neurone d'un tel réseau consiste à réaliser une combinaison linéaire de ses éléments d'entrée. Une fonction non-linéaire, appelée fonction d'activation, est ensuite appliquée au résultat de cette combinaison linéaire. Les neurones sont organisés en couches, chacune pouvant contenir en contenir plusieurs. Le nombre de neurones et de couches est fixé avant l'entraînement du réseau. Les neurones d'une couche sont tous connectés à chaque neurone de la couche suivante. Enfin, le rôle de la couche de sortie est de donner la probabilité (entre 0 et 1) d'appartenance du vecteur d'entrée à une classe. Cette probabilité est généralement calculée avec une fonction *softmax* (voir l'équation 2.3, où la probabilité d'appartenance à une classe c est calculée à partir des valeurs $\mathbf{y} = \{y_1; ...; y_K\}$ des neurones de la couche de sortie). C'est pourquoi cette couche contient finalement autant de neurones que de classes.

$$\sigma(\mathbf{y})_{C_k} = \frac{e^{y_k}}{\sum_{i=1}^{K} e^{y_i}} \tag{2.3}$$

L'entraînement d'un réseau de neurones, pour résoudre un problème de classification (ou de régression), consiste à adapter les différents poids des liaisons entre les neurones, afin de minimiser l'erreur de classification. Nous utilisons pour cela des méthodes d'optimisation, dans le but de minimiser une fonction de coût basée sur cette erreur. Une méthode aujourd'hui très utilisée pour les réseaux de neurones est la descente de gradient. Il s'agit d'une méthode itérative, consistant à calculer de nouveaux poids $\mathbf{w}_n$ à partir de ceux de l'itération précédente ($\mathbf{w}_{n-1}$), et du gradient de la fonction de coût par rapports à ces poids. Le principe de fonctionnement de la descente de gradient est donné dans l'équation 2.4. Le gradient est pondéré par un paramètre α, appelé taux d'apprentissage, avant d'être soustrait aux poids de l'itération précédente. Ainsi, un gradient nul signifie qu'un minimum local a été atteint.

$$\mathbf{w}_n = \mathbf{w}_{n-1} - \alpha \times \frac{dL(\mathbf{w}_n)}{d\mathbf{w}_n} \tag{2.4}$$

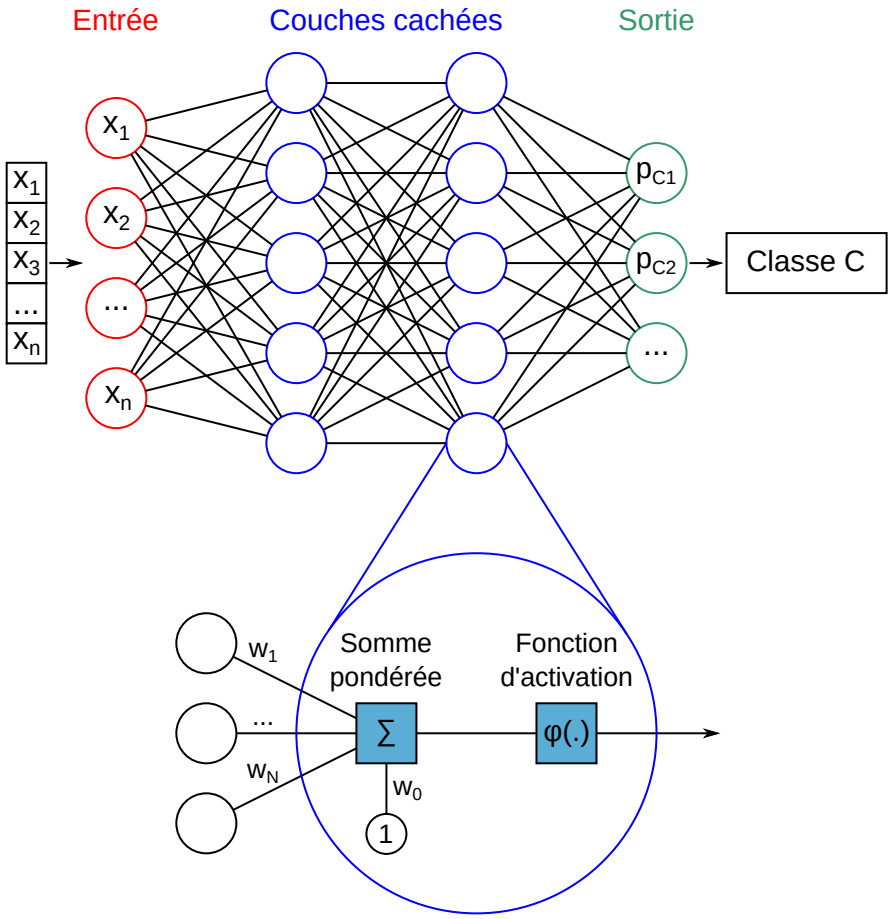


FIGURE 2.16 – Principe de fonctionnement d'un réseau de neurones. Les neurones sont organisés en couches, chacun d'entre eux étant connectés aux neurones de la couche suivante. Chaque neurone effectue une combinaison linéaire de la sortie des neurones de la couche précédente, puis y applique une fonction d'activation non-linéaire φ. Les éléments du vecteur d'entrée $\mathbf{x}$ sont connectés à la première couche cachée. Chaque neurone de la couche de sortie donne la probabilité p_{C_i} de l'appartenance du vecteur d'entrée à une classe C_i. La classe finale C attribuée au vecteur d'entrée est finalement celle dont la probabilité est la plus élevée.

Le taux d'apprentissage permet de paramétrer l'amplitude des pas entre les itérations, et est souvent déterminé de manière empirique. Une valeur trop importante peut engendrer l'oscillation de l'entraînement autour d'une solution, sans pouvoir converger, alors qu'une valeur trop faible peut rendre l'entraînement plus lent. Celui-ci pouvant être difficile à déterminer, et ayant un impact important sur la convergence de l'apprentissage, il existe des algorithmes permettant d'ajuster ce taux en cours de l'apprentissage en fonction de l'évolution de la fonction de coût.

Contrairement à *random forest*, les réseaux de neurones peuvent souffrir de problèmes de sur-apprentissage. Une première raison de ce problème est que, en fonction du nombre de neurones et de couches utilisés au sein d'un réseau, le nombre de paramètres (poids) à optimiser lors de l'apprentissage peut être très important, par rapport au nombre d'échantillons d'apprentissage (ce qui est souvent le cas en biologie). Les méthodes de régularisation permettent de réduire le sur-apprentissage. Elles consistent à ajouter un membre au calcul de la fonction de coût, comme indiqué dans l'équation 2.5. Ce membre calcule la norme de l'ensembles des poids $\mathbf{w}_n$ du réseau à chaque itération n. Cette norme est pondérée par un taux de régularisation λ, puis est ajoutée à la fonction d'erreur de classification $\epsilon(\mathbf{w}_n)$. Ainsi, l'ajout de ce paramètre permet de minimiser à la fois l'erreur de classification et la valeur des poids du réseau afin de réduire leur norme. Cela force donc l'algorithme d'apprentissage à attribuer des poids de très faible valeur aux neurones ayant peu d'importance pour la classification (Bishop 2006, pp. 144-145, pp. 256-257).

$$L(\mathbf{w}_n) = \epsilon(\mathbf{w}_n) + \lambda \times \|\mathbf{w}_n\|_2 \tag{2.5}$$

Une autre méthode fréquemment utilisée, permettant d'éviter le sur-apprentissage des réseaux de neurones, consiste à stopper l'apprentissage lorsque celui-ci commence à apprendre les détails du jeu d'apprentissage (empêchant la généralisation). Cette méthode est appelée arrêt anticipé (Bishop 2006, pp. 259-261). En plus de l'utilisation d'un jeu de données d'apprentissage, cette méthode consiste à utiliser un second jeu de données, appelé jeu de validation. Contrairement au jeu d'apprentissage, celui-ci n'est pas utilisé lors de l'optimisation des poids du réseau. En revanche, la fonction de coût est utilisée sur les deux jeux de données. Ainsi, si la fonction de coût augmente plusieurs itérations d'affilées avec le jeu de validation, cela signifie que le réseau est en train de sur-apprendre, et que l'apprentissage peut être stoppé.

Finalement, malgré les problèmes de sur-apprentissage des réseaux de neurones, ceux-ci ne sont constitués que d'opérations arithmétiques, telles que des combinaisons linéaires (multiplications de matrices) et des fonctions d'activation non-linéaires. Contrairement à *random forest*, les réseaux de neurones n'utilisent pas de condition, les rendant plus efficace lors d'une exécution sur un CPU (pas de vidage de pipeline). De plus, il est possible d'exécuter les réseaux de neurones sur des GPU afin de paralléliser les calculs des différents neurones, et de les accélérer par rapport à une exécution sur un CPU. Ils sont ainsi plus enclin à une exécution en temps-réel pour la détection d'évènements rares et courts.

2.2.2 Détection d'objets avec des méthodes d'apprentissage profond

Les méthodes usuelles consistent en trois étapes que nous avons présentées dans la section 2.2.1 (segmentation, extraction de caractéristiques et classification). Ces méthodes de segmentation et d'extraction de caractéristiques sont figées, contrairement à la classification qui peut être entraînée avec un jeu d'apprentissage. Il existe aussi des méthodes permettant par exemple de classifier directement des images, en entraînant l'extraction de pseudo-caractéristiques en même temps que la classification. C'est ce qu'on appelle l'apprentissage profond (Xing et al. 2018 ; Hongkai Wang et al. 2018). Nous pouvons distinguer deux types de méthodes ayant pour entrée une image et pour sortie une classification ou une détection d'objets.

Une méthode d'apprentissage profond a aussi été développée permettant de créer une fonction fusionnant les trois points de segmentation, d'extraction de pseudo-caractéristiques et de classification. Cette méthode, appelée segmentation sémantique, permet d'extraire des ROI directement depuis une image contenant plusieurs objets, en leur attribuant une classe (Xing et al. 2018 ; Hongkai Wang et al. 2018).

2.2.2.1 Réseaux de neurones convolutifs pour la classification

Les réseaux que l'on appelle Convolutional Neural Network (CNN, Réseau de Neurones Convolutif) permettent entre autres de classifier des images. Ils sont composés de deux parties, illustrées dans la figure 2.17 :

— des couches de convolutions permettant d'extraire des pseudo-caractéristiques spécifiques au type des images ;
— des couches connectées de classification.

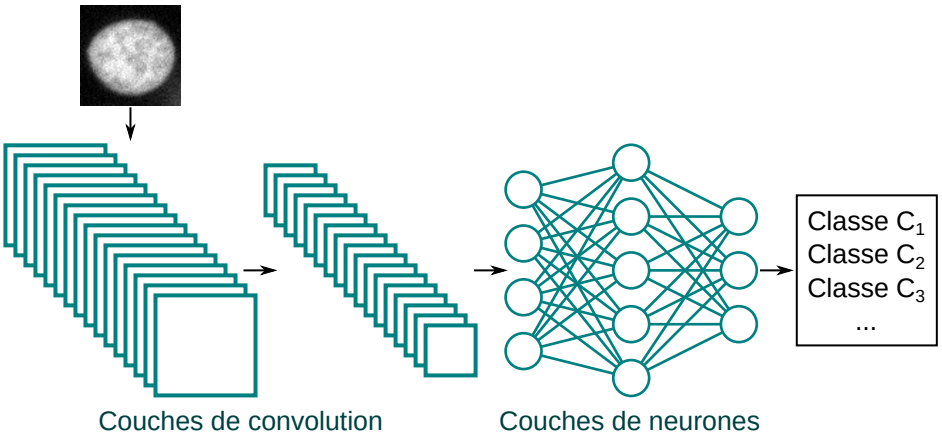


Figure 2.17 – Schéma d'un réseau de neurones convolutif de classification. Chaque couche de convolution génère plusieurs images de sortie, servant d'entrée à la couche suivante. Les pixels des images de la dernière couche convolutive sont arrangées dans un vecteur, représentant les pseudo-caractéristiques, et servant d'entrée d'un réseau de neurones classique.

Ils prennent en entrée les images brutes pour les classifier directement. En effet, les couches de convolution permettent d'obtenir un extracteur de pseudo-caractéristiques, entraîné et adapté aux images devant être classifiées (Xing et al. 2018 ; Hongkai Wang et al. 2018). Gao et al. utilisent par exemple un CNN pour classifier des cellules humaines afin de faciliter la détection de maladies auto-immunes (Gao et al. 2017). Il est donc possible d'utiliser ce genre de réseau pour fusionner les méthodes d'extraction de caractéristiques et de classification que nous avons présentées dans la section 2.2.1, comme illustré dans la figure 2.18 (Haibo Wang et al. 2014).

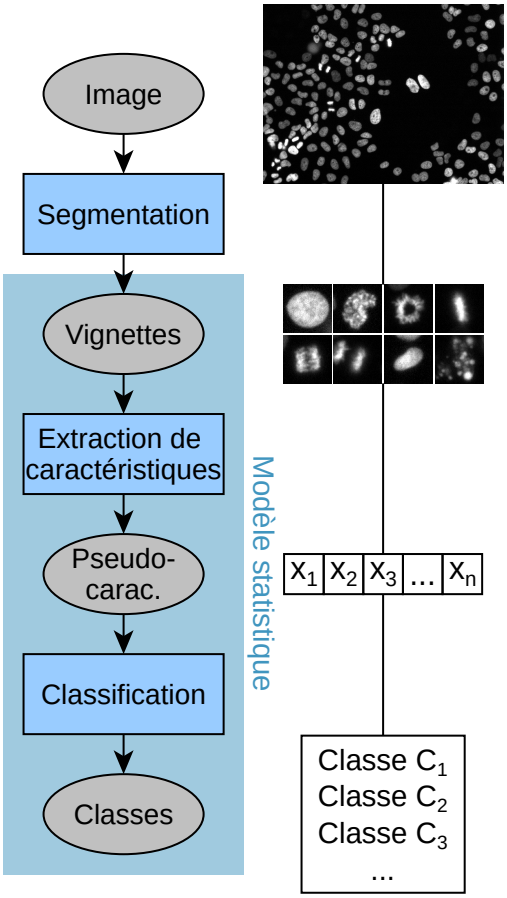


Figure 2.18 – Méthode de détection d'objets avec un réseau de neurones convolutif pour la classification, fusionnant les étapes d'extraction de caractéristiques et de classification. L'élément d'entrée du modèle statistique est une vignette contenant un objet, à laquelle une classe est directement attribuée.

Comme mentionné précédemment, les réseaux de neurones convolutifs apprennent à calculer des pseudo-caractéristiques, en même temps qu'ils apprennent à classifier des images à partir de ces pseudo-caractéristiques. De la même manière que les couches de neurones utilisent un ensemble d'éléments de bases (perceptrons), les couches convolutives

générant les pseudo-caractéristiques, utilisent un ensemble de filtres de convolutions, dont les poids sont adaptés lors de l'apprentissage. Ainsi, l'ajout ou la modification de pseudo-caractéristiques ne nécessitent pas de développer des fonctions, contrairement à l'approche usuelle pour lesquelles les extracteurs de caractéristiques sont figés. Cela permet donc d'adapter plus facilement les réseaux de neurones convolutifs à différentes applications.

De plus, les filtres de convolution, au même titre que les perceptrons, n'utilisent que des opérations arithmétiques. Cela rend donc possible l'accélération leur calcul, en les exécutant sur GPU. Dans le cas des extracteurs de caractéristiques utilisés dans les approches usuelles, cela n'est pas toujours le cas, et dépend des fonctions utilisées.

En revanches, contrairement aux caractéristiques des approches usuelles, les pseudo-caractéristiques générées par les couches convolutives n'ont pas de sens physique, ce qui rend leur interprétation difficile par des développeurs. Cet aspect de "boite noir" et l'interconnexion entre les différentes couches, rend les réseaux de neurones convolutifs difficiles à optimiser en termes de temps d'exécution.

2.2.2.2 Réseaux de neurones convolutifs pour la détection d'objets

Il est aussi possible d'utiliser des CNN pour détecter des objets directement sur des images non-segmentées. Comme présenté dans la figure 2.19, ces modèles prennent directement en entrée une image contant plusieurs objets, et détecte les différentes ROI englobant les objets en leur attribuant une classe. Certaines architectures de CNN permettent par exemple de localiser des ROI sous forme de rectangles encadrant chaque objet sur l'image, auxquels est associé une classe. L'architecture la plus populaire à l'heure actuelle est appelé Faster Region-based CNN (R-CNN) (Hongkai Wang et al. 2018 ; Ren et al. 2015). Celle-ci a par exemple été utilisée par J. Zhang et al. dans pour détecter des cellules cancéreuses en microscopie à contraste de phase (J. Zhang et al. 2016 ; Hongkai Wang et al. 2018).

Ce type de réseaux de neurones possède les mêmes avantages et inconvénients que les CNN présentés dans la section 2.2.2.1, de manière plus importante. En effet, les CNN permettant une détection des objets, directement à partir d'une image pouvant en contenir plusieurs, permet de s'affranchir à la fois des fonctions de segmentation et d'extraction de caractéristiques. Celles-ci sont réalisées et entraînées avec les couches convolutives du réseau, en fonction du type d'images utilisées. En revanche, l'ensemble du processus de détection d'objets est une "boite noire", rendant l'interprétation du réseau et son optimisation plus difficile que les méthodes présentées précédemment.

2.2.2.3 Réseaux de neurones convolutifs pour la segmentation sémantiques

Enfin, les méthodes de segmentation sémantique peuvent aussi être utilisées pour détecter des objets directement sur une image en contenant plusieurs. Cette méthode permet de classifier chaque pixel de l'image, afin d'obtenir un masque représentant chaque objet segmenté sous forme de régions classifiées. Les réseaux de type Fully Convolutional Network (FCN, Réseau Purement convolutif) sont conçus pour ce genre de problèmes

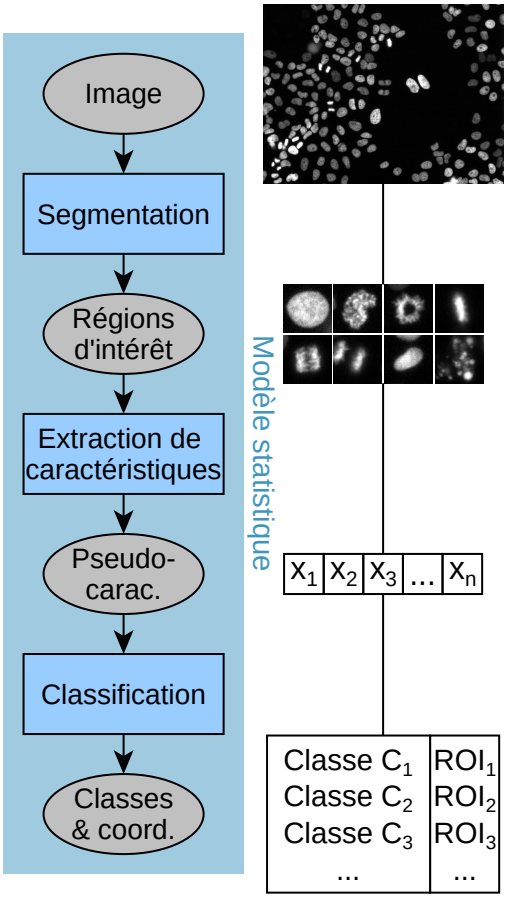


FIGURE 2.19 – Méthode de détection d'objets avec un réseau de neurones convolutif, fusionnant les étapes de segmentation, d'extraction de caractéristiques et de classification. L'élément d'entrée du modèle statistique est l'image contenant tous les objets, à partir de laquelle des ROI sont détectées et classifiées.

(Hongkai Wang et al. 2018 ; Dai et al. 2016). Feng Ning et al. utilisent par exemple ce genre de réseaux pour segmenter différentes parties d'embryons de C. elegans (Feng, Zhou et Dong 2019).

2.2.2.4 Discussion sur l'utilisation des réseaux de neurones convolutifs

Finalement, ces différentes méthodes de détection d'objets, basées sur des algorithmes d'apprentissage profond, offrent une grande flexibilité, et permettent de s'affranchir des étapes d'extraction de caractéristiques, voire de segmentation présentées dans la section 2.2.1 (Hongkai Wang et al. 2018). Le fait de pouvoir entraîner ce genre d'algorithmes avec ses propres images les rendent adaptés à la création d'outils génériques de détection d'objets.

S'agissant de méthodes basées sur des réseaux de neurones, ils peuvent être en proie à des problèmes de sur-apprentissage, du fait du très grand nombre de leurs paramètres libres (poids des filtres de convolution et des neurones). Des méthodes ont été développées ces dernières années pour limiter ce problème, telles que l'arrêt anticipé de l'apprentissage lorsque le modèle commence à sur-apprendre. Nous pouvons aussi citer les méthodes de régularisation et de *dropout*[9] (Srivastava et al. 2014).

Le temps d'exécution des réseaux de neurones est un point important lorsque nous nous penchons sur des questions de détection d'objets en temps-réel. Certains réseaux comportant un grand nombre de couches convolutives peuvent représenter un temps d'exécution important. Il existe tout de même des architectures de réseaux adaptées pour une exécution en temps-réel comme les réseaux YOLO (Redmon et Farhadi 2017), Faster R-CNN (Ren et al. 2015) ou RetinaNet (Lin et al. 2017), qu'il est possible d'utiliser sur des systèmes embarqués par exemple (Waithe et al. 2020).

En revanche, un inconvénient de ces réseaux est leur aspect "boite noire", les rendant plus difficile à étudier et à optimiser en temps d'exécution que les méthodes usuelles d'apprentissage automatique telles que *Random Forest* (Breiman 2001).

2.3 Systèmes embarqués pour l'instrumentation et l'analyse d'images de microscopie

En microscopie et en biologie, les ordinateurs à usage général sont à l'heure actuelle les outils les plus utilisés pour l'analyse d'images et le pilotage de microscope. En effet, la majorité des outils d'analyse et de pilotage sont développés pour ce genre de support matériel. Or, leur utilisation n'est pas adaptée pour des applications temps-réel, les systèmes d'exploitation comme Windows et les différents pilotes utilisés pouvant créer des latences importantes. De ce fait, les ordinateurs présentent des temps d'exécution de leur fonctions très variables, et peuvent présenter des instabilités s'ils sont amenés à exécuter un grand nombre de tâches en parallèle.

9. Le dropout est une méthode utilisée sur des CNN pour éviter le sur-apprentissage, en désactivant aléatoirement des neurones durant l'apprentissage (Srivastava et al. 2014).

Par opposition aux ordinateurs, les systèmes embarqués sont des supports électroniques dédiés, destinés à réaliser une ou quelques tâches très spécifiques. Ceux-ci peuvent être équipés de microcontrôleurs (comprenant un CPU, de la mémoire, des entrées/sorties, etc), les rendant ainsi programmables. Ils sont souvent utilisés pour l'exécution de fonctions en temps-réel, et le temps d'exécution de leurs tâches est plus stable et reproductible. Ceux-ci sont présents dans quasiment tous les domaines, de l'électro-ménager à l'aéronautique, en passant par l'automobile, la téléphonie ou le milieu médical. La méthode de pilotage développée par Inscoper a justement montré une rapidité accrue (trois fois plus rapide que sur un ordinateur), du fait de l'exécution de ces fonctions de pilotage sur un système embarqué (*Inscoper* 2020). En effet, ce système ne souffre pas des latences provoquées par le système d'exploitation Windows, et ne peut pas être interrompu par d'autres tâches non-liées aux fonctions de pilotage.

Étant donné la forte activité de recherche sur les techniques d'apprentissage profond, les systèmes embarqués intègrent de plus en plus de puissance de calcul. En effet, certaines cartes à microcontrôleurs hybrides, basées sur ARM, embarquent aussi des GPU. Parmi les systèmes embarqués grand public, nous pouvons citer :

— Raspberry Pi et ses dérivées (*Raspberry Pi* 2020) ;
— certains modèles Odroid (comme la Xu4) (*ODROID* 2020) ;
— les cartes embarquées NVIDIA Jetson (*NVIDIA Embedded Systems* 2020).

Castillo-Secilla et al. ont par exemple réalisé un système d'autofocus, en accélérant l'analyses des images sur le GPU d'une NVIDIA Jetson TX1 (Castillo-Secilla et al. 2017). Aussi, Waithe et al. comparent l'exécution de différents réseaux de neurones convolutifs tels que Faster R-CNN, YOLO v2 et v3 et RetinaNet pour détecter des cellules sur une NVIDIA Jetson TX2 (Waithe et al. 2020). Cette solution propose un tri automatique de cellules à la volée, mais ne présente pas de boucle de rétro-action permettant le pilotage du microscope.

Nous trouvons aussi des exemples de systèmes d'acquisition d'images à bas coût, comme le microscope multimodal proposé par Watanabe et al., basé sur l'utilisation d'une Raspberry Pi avec un module caméra de la même marque (Watanabe et al. 2020).

Certains systèmes embarquent aussi des ressources matérielles permettant d'accélérer les calculs de réseaux de neurones de type CNN, comme la NVIDIA Jetson AGX Xavier (*NVIDIA Embedded Systems* 2020) ou la Rock Pi 4 (*ROCK Pi* 2020).

2.4 Conclusion

Finalement, la question de la microscopie intelligente reste très ouverte à l'heure actuelle. En effet, la littérature montre aujourd'hui un fort intérêt des biologistes pour l'automatisation des systèmes de microscopie, qui permettrait de résoudre plusieurs limites biologiques, optiques et technologiques actuelles (Scherf et Huisken 2015 ; Eisenstein 2020). Diverses solutions ont été proposées jusqu'à aujourd'hui permettant d'analyser des images à la volée, et de créer une boucle de rétro-action, avec un ou quelques périphériques d'un microscope. Or ces méthodes sont actuellement très spécifiques à quelques types de microscopie et quelques expériences biologiques (Conrad et al. 2011 ; Mahecic et

Manley 2019 ; He et Huisken 2020 ; Durand et al. 2018 ; Nitta et al. 2018 ; Scherf et Huisken 2015 ; Eisenstein 2020). C'est pourquoi notre objectif est d'aller vers un système de microscopie intelligente générique, afin de pouvoir automatiser un maximum d'expériences.

Bien qu'étant proche de ce que nous souhaitons réaliser dans ce projet, la solution Micropilot (Conrad et al. 2011) peut être améliorée d'un point de vue temps d'exécution. En effet, son exécution séquentielle sur un ordinateur induit des latences, pouvant être incompatibles avec l'étude d'évènements rares et/ou courts. C'est pourquoi nous optons pour une solution embarquée, permettant l'exécution en parallèles des fonctions de pilotage et d'analyse d'images.

Notre objectif est de créer un système capable de détecter des évènements rares et courts en biologie en temps-réel. Il existe aujourd'hui une large panoplie d'algorithmes permettant de résoudre ce genre de problème, constituée d'outils fréquemment utilisés en analyse d'images et d'apprentissage automatique profond. L'utilisation de méthodes d'apprentissage automatique est en adéquation avec notre objectif de créer un système générique. En effet, elles peuvent être spécialisées pour une application lors d'une phase d'entraînement, sans avoir à être redéveloppées. Les tendances actuelles allant vers l'usage de réseaux de neurones convolutifs dans un grand nombre de domaines, impliquant de la détection d'objets, ceux-ci peuvent être des candidats intéressants pour notre problématique. Or, de par leur aspect de "boite noire", la compréhension de leur résultats d'apprentissage est difficile par rapport à certaines approches usuelles. C'est pourquoi nous nous intéressons aux méthodes usuelles de détection d'objets, constituées d'étapes de segmentation, d'extraction de caractéristique d'images et de classification, dans le but de pouvoir les optimiser pour une exécution en temps-réel.

Chapitre 3

Matériel et méthodes

3.1 Support hardware

3.1.1 Système embarqué

Nous souhaitions exécuter notre application sur un système embarqué. Nous avons sélectionné la NVIDIA Jetson AGX Xavier [1], dont les caractéristiques techniques sont présentées dans la table 3.1 (*NVIDIA Embedded Systems* 2020). Cette carte embarque aussi un Linux de type Ubuntu, nous permettant entre autres d'exécuter des tâches en multi-thread (*NVIDIA Embedded Systems* 2020).

En termes de connectique, la NVIDIA Jetson AGX Xavier comporte :

— deux ports USB3.1 ;
— un port Ethernet Gigabit ;
— un port x8 PCIe Gen4.

3.1.2 Chaîne de compilation

Afin de programmer la NVIDIA Jetson, nous avons construit une chaîne de compilation croisée à partir de la collection de compilation GNU (GCC) (*GCC* 2020). Cette chaîne nous permet de compiler des programmes C et C++ sur une cible *aarch64* (ARM 64 bit) embarquant un système d'exploitation Linux. Nous avons choisi des versions d'outils et de librairies compatibles avec ceux de la NVIDIA Jetson (voir table 3.2). La version de GCC que nous avons utilisée permettait l'utilisation des librairies standards C++ stables jusqu'à la version C++14.

3.2 Caméra de microscopie

Seul le fabricant de caméra scientifique Hamamatsu nous a fourni les pilotes adéquats pour utiliser leurs caméras sur un système ARM. C'est pourquoi nous nous sommes

1. Description complète de la NVIDIA Jetson AGX Xavier : `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/`

CPU	Type	NVIDIA Carmel ARM®
	Cœurs	8 cœurs v8.2 64 bits
	Cache	8 MiB L2 + 4 MiB L3
Mémoire	Type	LPDDR4x
	Taille	32 GiB
	Bus	256 bit
	Vitesse	136,5 GB/s
GPU	Type	NVIDIA Volta™
	Cœurs	512 cœurs CUDA 64 cœurs Tensor
Accélérateur apprentissage profond	Nb.	2 × moteurs NVDLA
Performances d'IA	Vitesse	32 TOPS

TABLE 3.1 – Caractéristiques techniques de la NVIDIA Jetson AGX Xavier (*NVIDIA Embedded Systems* 2020).

Outil	**Version**
GCC	8.4.0
Binutils	2.30
Noyau Linux	4.9.140
GlibC	2.27
MPFR	4.0.1
GMP	6.1.2
MPC	1.1.0
ISL	0.18
Cloog	0.18.1
GDB	8.1

TABLE 3.2 – Version des éléments de la chaîne de compilation.

tourné vers un modèle de cette marque, plus précisément la Hamamatsu ORCA-Flash4.0 C11440-22C (*Hamamatsu* 2020). Les images de cette caméra sont envoyées via un port USB3 ou *Camera Link*. Le seul port compatible avec les cartes NVIDIA Jetson est l'USB3. Nous présentons certaines des caractéristiques techniques de cette caméra dans la table 3.3. Cette caméra comporte aussi trois entrées/sorties numériques permettant la synchronisation et le déclenchement de la caméra.

Technologie	CMOS
Nb. pixels	$2\,048 \times 2\,048$
Taille pixel	6,5 µm × 6,5 µm
Capacité des puits	30 000 électrons
Débit d'images	30 images/s
Binning	2×2, 4×4
Temps d'exposition	38,96 µs à 10 s

TABLE 3.3 – Caractéristiques techniques de la Hamamatsu ORCA-Flash4.0 C11440-22C (*Hamamatsu* 2020).

Cette caméra est utilisable via une Application Programming Interface (API, Interface de Programmation Applicative) compilée sous deux formats :

— pour les architectures x86, pour les environnements Windows ;
— pour les architectures ARM, pour les environnements Linux.

Elle peut être utilisée entre autres avec les cartes NVIDIA Jetson.

3.3 Microscope et périphériques

Afin de tester notre prototype, nous avons travaillé sur un microscope équipé de plusieurs périphériques. Le choix de ce matériel a été fait par Inscoper indépendamment de ce projet de thèse. Il est composé des périphériques suivants :

Statif Nikon Ti2e ;

Laser Lumencore SpectraX ;

Module de photomanipulation Starscan.

Ces types de statif et de périphériques sont couramment utilisés en recherche. En effet, ils sont équipés de composants motorisés comme la platine, permettant de déplacer les échantillons biologiques en X et Y, le focus permettant un réglage en Z, ainsi que la sélection des différents objectifs ou filtres de lumière.

3.4 Suite d'acquisition d'images et de contrôle Inscoper

Afin de piloter le microscope et ses périphériques de manière efficace, nous avons utilisé la solution de pilotage et d'acquisition d'images développée par Inscoper (*Inscoper* 2020). Cette solution est constituée de deux sous-ensembles :

— un système dédié (système embarqué) pilotant le microscope et ses périphériques ;
— une interface graphique installée sur un ordinateur permettant à un utilisateur de paramétrer l'expérience qu'il souhaite réaliser.

3.4.1 Module de pilotage

Le module de pilotage Inscoper est un système embarqué dont la fonction est de contrôler le microscope ainsi que ses différents périphériques (lasers, platine, etc), en suivant des séquences d'acquisition [2] définies par l'utilisateur. Ce module donne la possibilité d'exécuter plusieurs séquences. Elles peuvent être exécutées les unes après les autres dans un ordre défini par l'utilisateur. Une séquence peut aussi en interrompre une autre, mettant cette dernière en pause. Celle-ci reprend son cours une fois la séquence l'ayant interrompue terminée.

Chaque séquence d'acquisition permet l'exécution de fonctions de pilotages (déplacement de la platine à une position donnée, vérification de sa position, sélection d'un filtre, etc) dans un ordre précis réalisant ainsi l'expérience souhaitée. Chaque fonction génère des signaux destinés aux différents périphériques du microscope (platine, lasers, focus, objectifs, etc), via des signaux numériques, analogiques ou série (RS232, USB, Ethernet), afin de pouvoir les contrôler. Ce modules est aussi capable de vérifier et d'attendre qu'un périphérique soit bien d'en l'état souhaité. Ceci permet de déterminer de manière efficace si un périphérique comme la platine ou le focus à bien été déplacé. Les fonctions sont donc organisées dans un ordre défini par l'utilisateur selon plusieurs dimensions au cours d'une séquence :

— la dimension temps permet de définir l'intervalle de temps séparant l'acquisition des images ;
— la dimension position permet de définir les coordonnées en XY de chaque image ;
— la dimension focus permet de définir les coordonnées en Z de mise au point ;
— la dimension canal permet de définir et configurer les périphériques utilisé (lumière, lasers, filtres, etc) ;
— ...

La configuration de ces dimensions se fait via une interface graphique (voir section 3.4.2).

Enfin, une autre fonctionnalité du module de pilotage consiste à envoyer une confirmation après l'exécution d'une fonction, permettant de connaître la dernière fonction ayant été exécutée (en indiquant aussi la dimension à laquelle elle appartient). Cette fonctionnalité nous est utile pour connaître le contexte (état du microscope et de ses périphériques) de l'acquisition d'une image. Lorsqu'une fonction consiste à interroger un périphérique sur son état (récupération des coordonnées absolues en XY de certaines platines par exemple), la réponse est intégrée à la confirmation d'exécution de la fonction correspondante. Cela permet de connaître directement l'état d'un élément du système de microscopie. En revanche, cette fonctionnalité n'est pas disponible sur tous les périphériques.

2. Une séquence d'acquisition représente une liste d'instructions exécutées par le module de pilotage, définissant les différents états dans lequel doit être le microscope avant d'acquérir une image.

3.4.2 Interface graphique

L'interface graphique permet à l'utilisateur de la solution de pilotage de programmer son expérience. Cette configuration se fait en trois étapes :

— configuration des périphériques du microscope ;
— configuration de la séquence d'acquisition ;
— acquisition et visualisation des images.

La première étape de configuration permet à l'utilisateur de régler les différents périphériques du microscope, en les mettant dans un certain état avant de lancer son expérience. Cela correspond, par exemple, à choisir un grossissement en sélectionnant un objectif disponible sur le microscope.

C'est lors de la deuxième étape que l'utilisateur peut programmer une séquence d'acquisition selon plusieurs dimensions (acquisition multidimensionnelle). Ces dimensions sont, par exemple, la position de la platine (coordonnées en X et Y), la position du focus (coordonnées en Z permettant la mise au point), la dimension de temps, ou encore les lasers d'excitation utilisés. Pour chaque dimension, l'utilisateur peut choisir plusieurs points qui sont exécutés les uns après les autres. L'ordre dans lequel les dimensions sont parcourues est aussi défini par l'utilisateur. Ainsi, le déroulement de chaque point dans chaque dimension, définie ce que nous appelons la séquence d'acquisition. Chaque séquence est donc envoyée au module de pilotage pour être exécutée.

Contrairement à d'autres logiciels de pilotage de microscope tels que MetaMorph (*MetaMorph* 2020), l'interface utilisateur proposée par Inscoper présente une meilleure ergonomie. En effet, celle-ci est composée d'une unique fenêtre et comporte peu de menus déroulants, rendant son utilisation plus simple et intuitive.

3.5 Outils de statistiques

Au cours de ce projet, nous utilisons différents outils permettant de détecter et classifier des cellules sur des images, dans le but de détecter des évènements rares et courts. Nous utilisons donc des modèles statistiques pour analyser ces images. La détection d'évènements rares et courts nécessite de respecter des contraintes de temps d'exécution. Nous choisissons d'utiliser des outils de réduction de dimension pour optimiser en temps d'exécution, des fonctions d'analyse d'images existantes. Aussi, afin de valider le fonctionnement et la reproductibilité de cette optimisation, nous avons recours à une méthode de bootstrap.

3.5.1 Outils de réduction de dimension

3.5.1.1 Score de Fisher

Une première méthode de réduction de dimension que nous utilisons est la méthode linéaire du discriminant de Fisher (Fisher 1936). Elle permet de calculer un score quantifiant la discriminance d'une variable (caractéristique d'image), à partir du ratio entre la variance inter-classes et les variances intra-classes, de manière similaire à l'Analysis of Variance (ANOVA, Analyse de la Variance) (Larson Martin G. 2008). Dans notre

cas où les problèmes de classification de cellules comprennent plus de deux classes, nous réutilisons l'équation 3.3, utilisée dans le programme WND-CHARM développé par Orlov et al., utilisant la moyenne des classes μ_{C_k} et la moyenne globale μ (voir équations 3.1), ainsi que la variance des classes $\sigma^2_{C_k}$ (voir équation 3.2). Dans cette équation :

- $FS(x)$ représente le score de Fisher de la variable x ;
- N_{C_k} représente le nombre d'échantillons compris dans la classes C_k ;
- N représente le nombre d'échantillons total ;
- x_n représente la valeur de la variable x pour l'échantillon n.

Dans ces travaux, le score de Fisher est utilisé pour pondérer des variables, en fonction de leur discriminance pour un problème de classification de différents types de cellules (Orlov et al. 2008). Ce score est aussi utilisé lors de l'estimation des paramètres de discriminants linéaires, pour maximiser la discriminance entre les classes (Bishop 2006, pp. 186-192).

$$\mu_{C_k} = \frac{1}{N_{C_k}} \sum_{n \in C_k} x_n \qquad \mu = \frac{1}{N} \sum_{n=1}^{N} x_n \tag{3.1}$$

$$\sigma^2_{C_k} = \frac{1}{N_{C_k}} \sum_{n \in C_k} (x_n - \mu_{C_k})^2 \tag{3.2}$$

$$FS(x) = \frac{\sum_{C_k} (\mu - \mu_{C_k})^2}{\sum_{C_k} \sigma^2_{C_k}} \tag{3.3}$$

Ce score étant calculé directement à partir des valeurs de la variable dont la discriminance est mesuré (moyennes et variances), il peut être calculé une seule fois au avant de réaliser la classification.

3.5.1.2 Importance pour la classification avec *Random Forest*

Afin de réduire la dimension de notre problème de classification, nous avons aussi recours à une méthode non-linéaire, permettant de quantifier la contribution des caractéristiques d'images dans leur classification. Nous utilisons le modèle statistique de classification *random forest*, basé sur un ensemble d'arbres de décision, qui permet intrinsèquement d'estimer l'importance des variables utilisées lors de la classification (Breiman 2001).

Comme nous le présentons dans la section 2.2.1.3, les arbres de décision constituant un *random forest* sont entrainés avec différents sous-ensembles du jeu de données d'apprentissage : tous les échantillons et toutes les variables ne sont pas utilisées lors de l'apprentissage de chaque arbre de décision. Ainsi, pour un arbre donné, les échantillons ne figurant pas dans son jeu d'apprentissage sont dits Out-Of-Bag (OOB).

L'importance d'une caractéristique repose sur la variation de l'erreur de classification des arbres de décision du *random forest*, lorsque la valeur cette caractéristique change de manière aléatoire (voir figure 3.1). Pour chaque arbre de décision, les échantillons OOB correspondants sont classifiés et le nombre d'échantillons mal classifiés est mesuré. Cela

donne ainsi une erreur de classification pour chaque arbre. Pour une caractéristique donnée dont nous mesurons l'importance, les valeurs des échantillons OOB pour cette variable sont mélangées les unes avec les autres de manière aléatoire. Cela permet ainsi de donner des valeurs erronées à cette variable. L'erreur de classification des échantillons OOB est ensuite de nouveau mesurée, et comparée avec l'erreur précédemment mesurée. Cette différence est normalisée par rapport au nombre d'échantillons OOB. Ainsi, plus l'augmentation de cette erreur de classification est importante, plus la variable en question contribue au résultat de classification de l'arbre de décision. Finalement, cette augmentation de l'erreur est moyennée sur l'ensemble du *random forest*, pour chaque variable, donnant ainsi un score d'importance des caractéristiques pour la classification (Breiman 2001).

Contrairement au score de Fisher présenté dans la section 3.5.1.1, l'importance des caractéristiques lors de la classification avec *random forest* n'est pas liée uniquement aux valeurs de ces caractéristiques, mais aussi au modèle résultant de l'apprentissage. Cette importance doit donc être recalculée à chaque fois qu'un *random forest* est entraîné.

3.5.2 Validation de modèles statistiques par bootstrap

Du fait que nous utilisons des modèles statistiques pour classifier des images, il est nécessaire de valider la reproductibilité des résultats obtenus. Nous vérifions donc, lors de l'utilisation de ces modèles, que les résultats ne sont pas un cas particulier obtenu avec un jeu d'images spécifique, ou un état spécifique des modèles utilisant des paramètres aléatoires (comme *random forest* ou les réseaux de neurones, présentés dans la section 2.2.1.3).

Nous utilisons une méthode de bootstrap, pour valider que les résultats obtenus ne sont pas spécifique à un jeu de données et à un modèle. Le bootstrap consiste à répéter l'apprentissage et le test d'un modèle statistique un certain nombre de fois, dans le but d'analyser la variation des différents résultats obtenus (précision de la classification, importance des caractéristiques avec *random forest*). Comme présenté dans la figure 3.2, chaque modèle statistique est entraîné avec un sous-ensemble différent du jeu complet d'images. Les images de chaque sous-ensemble, dont nous extrayons des caractéristiques pour la classification (voir 2.2.1), sont sélectionnées aléatoirement. Ainsi, une faible variation des résultats obtenus avec les différents modèles montre leur reproductibilité.

Dans notre cas, nous souhaitons détecter des évènements rares et courts. Cela implique donc l'utilisation de bases d'images déséquilibrées [3] en termes de classes. L'utilisation de modèles statistiques nécessite d'équilibrer ces classes. De plus, il est possible que plusieurs images différentes représentent un même objet à différents instants. Les doublons doivent être supprimés. Afin d'intégrer le bootstrap à l'équilibrage et la suppression des doublons, nous réalisons les étapes suivantes :

— l'objet (cellule) présent sur l'image est identifié par un index (les images dont les coordonnées sont proches à différents instants, sont considérées comme présentant le même objet, et ont donc un index identique) ;

3. Une base d'images déséquilibrée contient une ou plusieurs classes avec un nombre d'images beaucoup plus important que les autres classes.

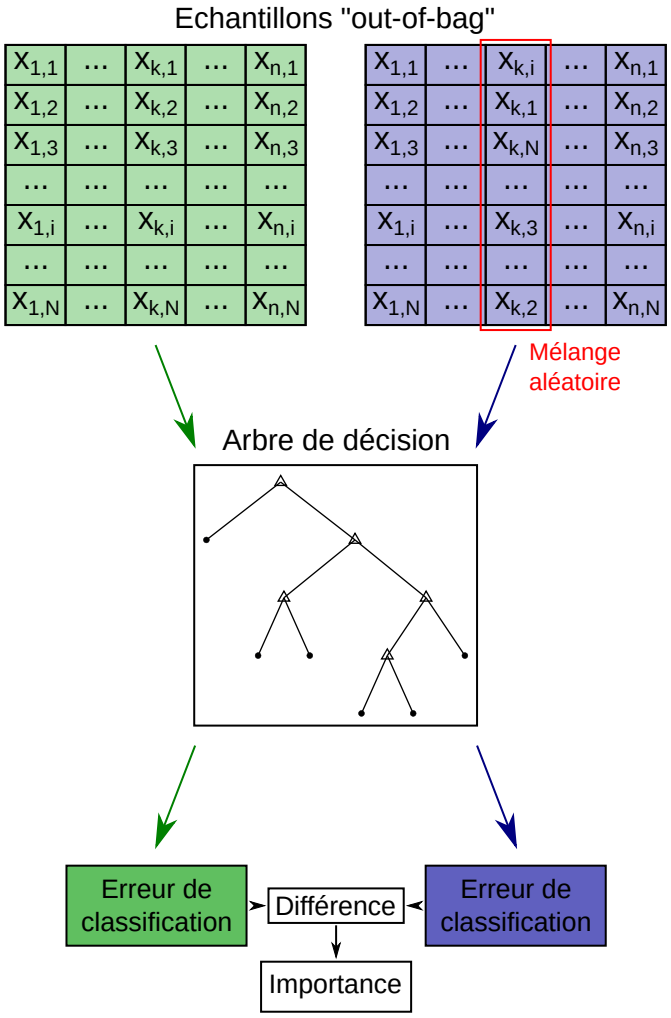


FIGURE 3.1 – Principe de l'importance d'une caractéristique dans un arbre de décision. Celui-ci classifie les caractéristiques OOB, avant et après avec mélangé les valeurs d'une caractéristique de manière aléatoire. Chaque classification donne un taux d'erreurs, dont la différence permet de quantifier l'importance de la caractéristique en question pour la classification.

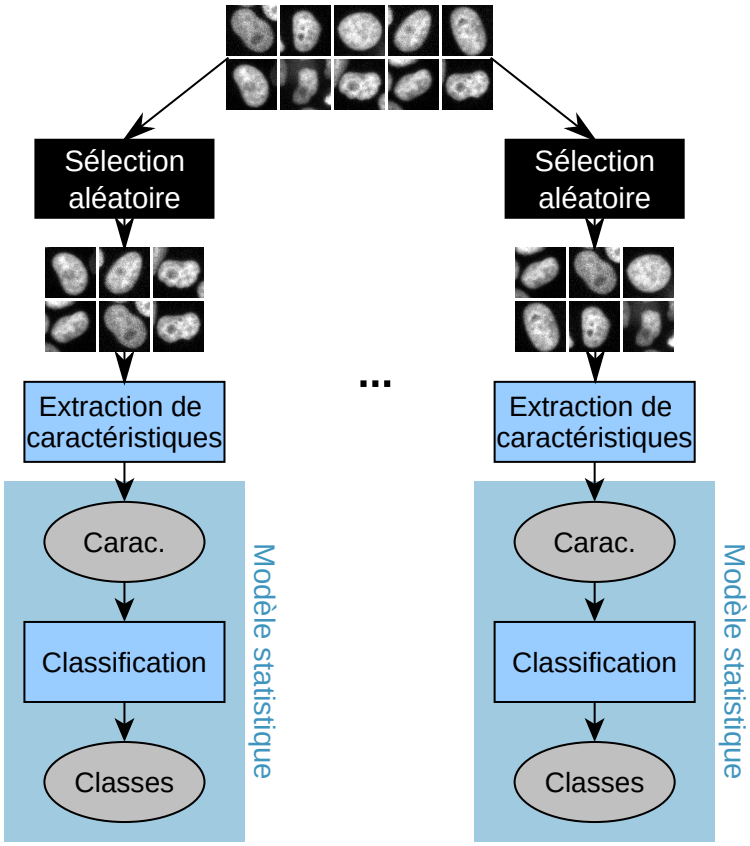


FIGURE 3.2 – Principe de fonctionnement du bootstrap. À partir du jeu d'images complet, plusieurs sous-ensembles sont créés pour en extraire des caractéristiques et entraîner un modèle statistique pour chacun d'entre eux. Les images de chaque sous-ensemble sont sélectionnées de manière aléatoire.

— les doublons d'objets sont supprimés aléatoirement, afin d'avoir une seule instance de chaque objet sur l'ensemble du jeu d'images ;
— des images sont aléatoirement supprimées dans les classes afin d'obtenir le même nombre d'images pour chaque classe.

Finalement, ce processus est répété à chaque itération du bootstrap afin de créer des jeux d'images différents pour chaque modèle statistique entraîné.

Deuxième partie

Contributions

Chapitre 4

Asservissement d'un système de microscopie

4.1 Objectifs du travail

Les systèmes de microscopie optique ont connu d'importantes avancées technologiques tant sur le plan optique que sur les plans électronique et mécanique, donnant les systèmes motorisés que nous connaissons aujourd'hui. Ce sont des outils incontournables pour la recherche en biologie (Nketia et al. 2017). Cette motorisation des microscopes a permis leur pilotage par ordinateur ou par des systèmes dédiés, automatisant ainsi les acquisitions d'images de biologie du vivant. Cela a rendu possible par exemple l'application de méthode de criblage (HCS) sur des échantillons biologiques vivants. Les méthodes actuelles impliquent en revanche l'acquisition d'un grand nombre d'images (dont la pertinence est aléatoire), qui sont ensuite traitées *a posteriori*. Rendre un microscope intelligent, en lui donnant la capacité d'analyser les images, de modifier leur contenu (transformation) et de modifier la ou les modalités d'acquisition en temps-réel, permettrait d'augmenter le ratio entre le nombre d'images pertinentes et le nombre total d'images acquises. Le contenu d'intérêt des méthodes de criblage en serait donc augmenté. Comme discuté dans la section 2.1, certaines méthodes d'automatisation de microscopes existent aujourd'hui, développées pour des applications très spécifiques, alternant acquisition d'images et analyse.

Notre objectif est de concevoir un système capable de :

— acquérir et stocker des images retournées par une ou plusieurs caméras scientifiques ;
— analyser et/ou transformer ces images à la volée ;
— modifier la ou les modalités d'acquisition en temps-réel en fonction des images traitées ;
— retourner vers un ordinateur les images d'intérêt traitées ou non, préalablement sélectionnées.

Contrairement aux solutions d'automatisation existantes suivant un modèle d'exécution séquentiel (Conrad et al. 2011 ; Tischer et al. 2014 ; Nitta et al. 2018), nous souhaitons éviter au maximum les temps d'inactivité de nos différentes tâches. La modification de

modalité d'acquisition en temps-réel nécessite de pouvoir interrompre une acquisition en cours pour en lancer une autre grâce à une méthode d'exécution événementielle (voir figure 4.1).

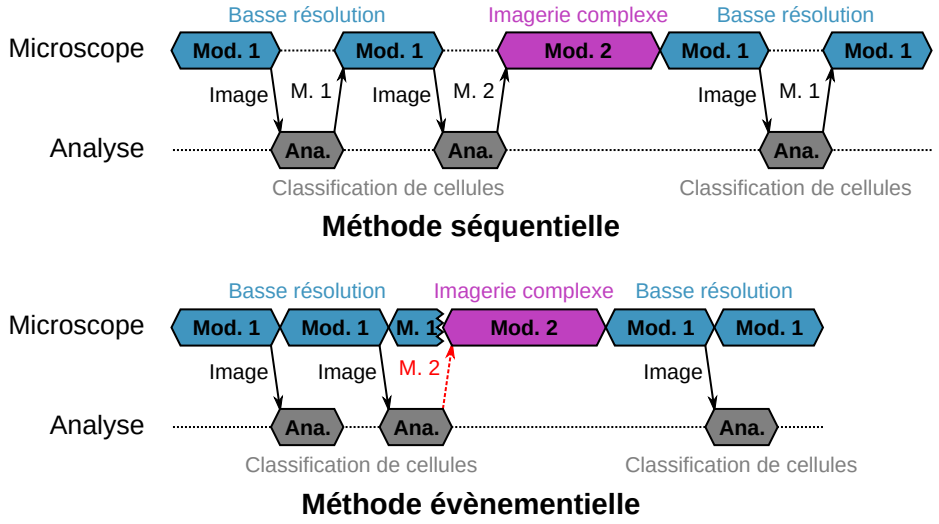


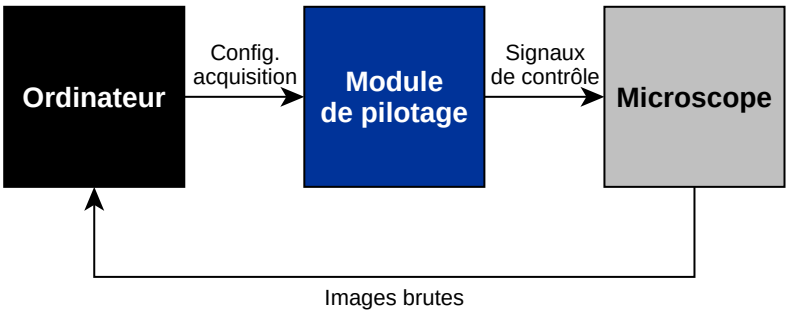
FIGURE 4.1 – Exemple de modification des modalités d'acquisition d'images avec un mécanisme d'interruption. Celui-ci représente l'acquisition d'images suivant de deux modalités (*Mod. 1* en basse résolution en bleu et *Mod. 2* en imagerie complexe en violet) sur un microscope. Les images sont analysées (*Ana.*) et les modalités déclenchées (*M. 1* et *M. 2* pour les modalités en basse résolution et en imagerie complexe respectivement).

Le système que nous concevons a pour vocation d'être industrialisé par Inscoper. Il est donc nécessaire de concevoir un système pouvant être modifié et maintenu facilement, et couvrant un maximum d'application biologique. Nous nous sommes donc imposés des contraintes de généricité et de modularité. Ceci étant, nous avons tout de même suivi une application spécifique afin de cadrer ce projet de thèse pour aller vers une preuve de concept. Cette application consiste à chercher des cellules en début de mitose ou à la transition métaphase-anaphase, que nous appelons cellules d'intérêt, sur une lamelle ou une plaque multi-puits, afin de réaliser un time-lapse [1] à haute cadence d'acquisition uniquement sur ces cellules d'intérêt.

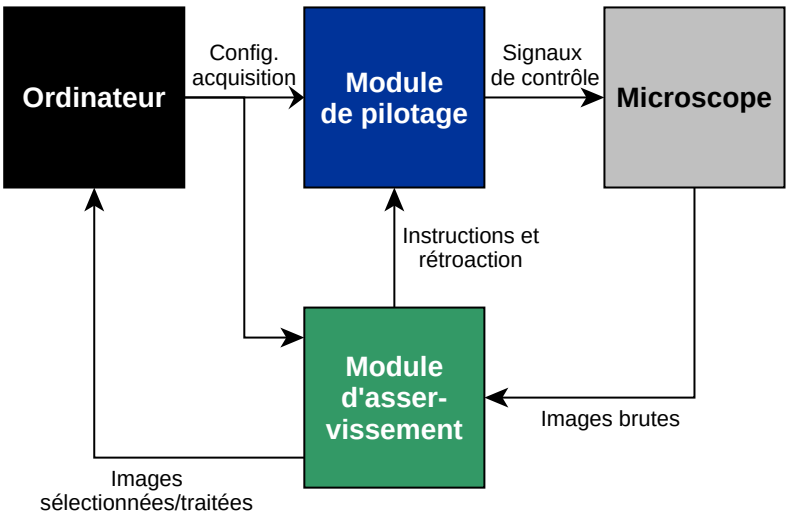
Inscoper commercialisant aujourd'hui une solution de pilotage de systèmes microscopes (voir figure 4.2a et section 3.4) (ROUL, TRAMIER et PECREAUX 2019), nous avons délimité ce projet en nous focalisant sur la réalisation l'asservissement du microscope. Nous réutilisons donc ce qui a déjà été développé par Inscoper en termes de pilotage (voir figure 4.2b). Toutefois, pour des raisons de simplicité d'utilisation, nous souhaitons laisser la possibilité d'utiliser le système complet présenté dans la figure 4.2b de la même manière que celui présenté dans la figure 4.2a, en laissant le choix à un potentiel utilisateur

1. Images acquises uniquement selon la dimension temps.

de réaliser une acquisition intelligente (avec asservissement) ou classique sans avoir à déconnecter physiquement aucun module.



(a)



(b)

FIGURE 4.2 – Schéma du pilotage d'un microscope avec la solution Inscoper (a) et de l'insertion de notre solution d'asservissement (b).

Ainsi, nous suivons une démarche de conception visant à étudier les différentes tâches que doit réaliser notre système, avant de l'implémenter sur un support matériel et de le tester en suivant l'application que nous avons sélectionnée.

4.2 Conception du système

Nous présentons dans cette section les différentes étapes de conception de notre système. Cette phase nécessaire nous permet de définir les entrées/sorties liant notre système à son environnement, et de structurer et diviser le système en fonctions élémentaires.

4.2.1 Définition des entrées/sorties et de l'environnement

La première étape de conception de notre système consiste à définir l'environnement avec lequel il doit communiquer et les différentes entrées/sorties le constituant. Il est destiné à interagir avec un environnement constitué du microscope et de ses périphériques, d'une caméra scientifique capturant des images, et d'une interface utilisateur permettant de configurer l'expérience. Plus précisément, l'interface utilisateur a pour rôle de configurer les périphériques du microscope, de paramétrer l'expérience devant être réalisée, ainsi que de recevoir les images traitées et sélectionnées. Le microscope quant à lui est contrôlé via un module de pilotage dédié avec lequel notre système doit communiquer. Cette interface et ce module sont des éléments de la solution de pilotage Inscoper.

À l'origine, le module de pilotage et l'interface utilisateur communiquent par un système de commandes mis au point par Inscoper. Afin d'éviter toute déconnexion des modules pour réaliser différentes expériences, il est nécessaire de laisser la possibilité à l'interface utilisateur et au module de pilotage de communiquer dans le cas où l'utilisateur du système souhaite réaliser une acquisition classique sans asservissement par exemple. Or, le module de pilotage ne présente qu'un seul canal permettant l'envoi et la réception de commandes. Nous avons donc décidé d'interfacer notre système entre l'interface utilisateur et le module de pilotage. En effet, afin de pouvoir modifier les modalités d'acquisition en temps-réel, il est nécessaire que notre système puisse communiquer directement avec le module de pilotage. Ainsi notre système devait être capable de transmettre des informations entre l'interface utilisateur et le module de pilotage.

Il en va de même pour la caméra, qui possède aussi un seul canal de transmission des images et de configuration. Notre système devant analyser les images retournées par la caméra, cette dernière doit être directement connectée au système.

Finalement, nous modélisons les interconnections entre les éléments constituant notre système (que nous avons appelé module *Roboscope*) et son environnement comme schématisé dans la figure 4.3.

4.2.1.1 Connexions avec l'interface utilisateur

L'interface utilisateur exécutée sur un ordinateur a deux rôles principaux :
— configurer l'expérience que l'utilisateur souhaite réaliser ;
— recevoir et stocker les images traitées par le système.

Configuration de l'expérience Les commandes utilisées pour la communication entre l'interface utilisateur et le module de pilotage utilisent un mécanisme pouvant facilement être adapté. Ainsi, plutôt que de créer un protocole spécifique à la communication entre

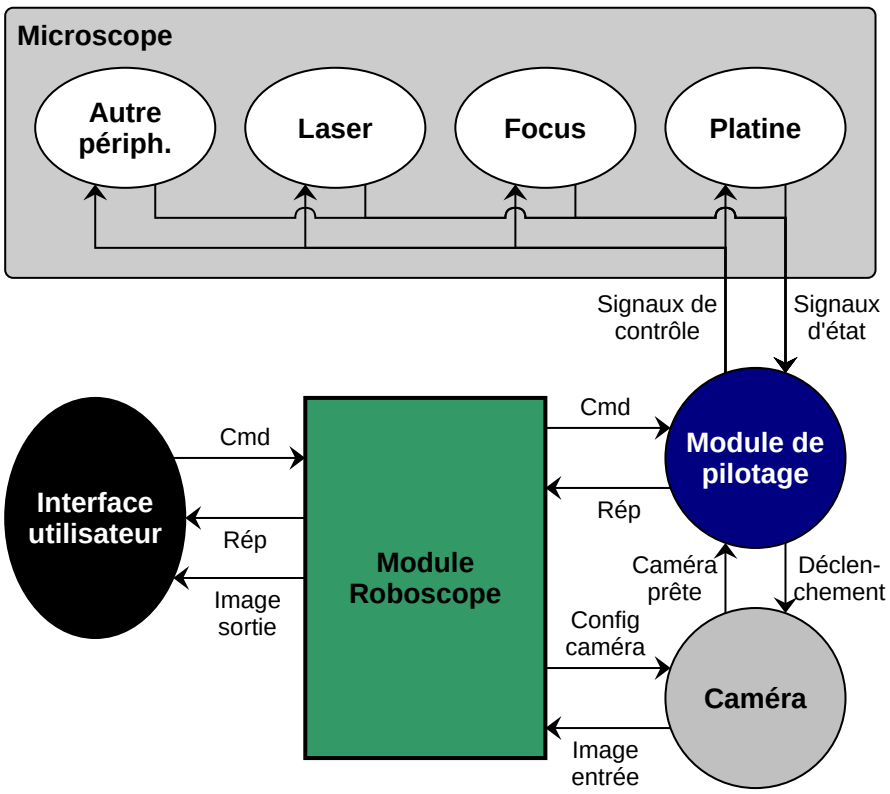


FIGURE 4.3 – Inter-connections entre le module *Roboscope* et son environnement. L'interface et le module de pilotage communiquent avec le module Roboscope par envoie de commandes (*Cmd.*) et réponses (*Rép.*).

l'interface utilisateur et notre système, nous avons décidé d'utiliser le même protocole qu'avec le module de pilotage afin d'éviter l'ajout d'une fonction de traduction de commandes. Cela est particulièrement utile lorsque des commandes doivent directement être envoyées de l'interface utilisateur au module de pilotage. Les rôles respectifs de *Cmd* et *Rép* sont donc de recevoir et d'envoyer des commandes en provenance ou vers l'interface utilisateur.

Envoi des images traitées Une fois les images traitées, celles devant être stockées (images d'intérêt) sur l'ordinateur par l'interface utilisateur sont envoyées via la sortie *Image sortie*.

4.2.1.2 Connexions avec la caméra

La caméra est le périphérique en charge d'acquérir les images de microscopie représentant les cellules ou les objets que nous souhaitons analyser. Celle-ci n'étant pas configurée par le module de pilotage, elle nécessite aussi d'être paramétrée par le système.

Configuration de la caméra Nous configurons la caméra via la sortie *Config caméra*, permettant entre autres de paramétrer son mode d'acquisition (déclenchement interne ou externe), son temps d'exposition, si elle doit effectuer un binning, etc.

Réception des images Le système reçoit les images brutes retournées par la caméra via l'entrée *Image entrée*.

4.2.1.3 Connexions avec le module de pilotage

Enfin, le système que nous développons est connecté au module de pilotage afin d'interagir avec le microscope et ses différents périphériques. Le module de pilotage communique avec les autres modules en utilisant le mécanisme de commandes conçu par Inscoper. Ainsi, la sortie *Cmd* et l'entrée *Rép* ont pour buts respectifs d'envoyer et de recevoir des commandes vers et depuis le module de pilotage. Les principales commandes utiles à l'asservissement du microscope sont :

— paramétrer, modifier, exécuter et interrompre des séquences d'acquisition ;

— obtenir des informations sur l'état de la séquence d'acquisition en cours d'exécution.

Du fait de la réalisation d'un asservissement, ce dernier point est important pour connaître le contexte dans lequel les images ont été acquises. Ce paramètre, en plus des images en tant que telles, nous sert d'élément de mesure.

Une fois les entrées/sorties du système définies, ainsi que leurs interactions avec son environnement, nous spécifions les différentes fonctions constituant le système.

4.2.2 Étude fonctionnelle du système

Lors de cette étape, nous divisons notre système en différentes fonctions élémentaires. Nous souhaitons concevoir un système modulaire, facilement modifiable, afin de permettre son maintient et son industrialisation. Aussi, nous voulons le rendre utilisable dans un grand nombre d'applications biologiques (contrainte de généricité). En effet, si une fonction vient à être modifiée ou ajoutée, cela ne doit pas impacter de manière significative le reste du système. C'est pourquoi, même si les tâches peuvent communiquer les unes avec les autres, celles-ci sont dimensionnées pour être exécutées de manière autonomes.

Dans un premier temps, nous divisons le système en macro-fonctions et fonctions élémentaires. Cela nous permet d'identifier les différents rôles du système et d'avoir une vision globale sur son comportement. Chaque macro-fonction regroupe plusieurs fonctions élémentaires. Nous avons identifié six tâches principales nous permettant de mettre en place le schéma fonctionnel présenté dans la figure 4.4. Ces tâches permettent de répondre théoriquement aux éléments du cahier des charges en offrant :

— une fonction d'acquisition et de stockage des images en provenance de la caméra (*Réception images*) ;
— des fonctions de traitement d'images (*Traitement images*) ;
— une macro-fonction permettant de modifier et exécuter les séquences d'acquisition en temps-réel (*Gestion séquences*) ;
— une fonction retournant les images d'intérêt vers l'interface utilisateur (*Envoi images*).

Afin de ne pas figer notre système à une application, nous le concevons pour être configurable pour plusieurs applications grâce à sa fonction *Gestion application*, nous permettant ainsi de répondre théoriquement à notre objectif de généricité. Enfin, la fonction *Gestion commandes* gère la réception et l'envoi des commandes depuis et vers l'interface utilisateur et le module de pilotage. Elle crée aussi un lien entre ces deux modules, leur permettant d'échanger des commandes en travers de notre système.

Aussi, ces fonctions communiquent entre elles et avec l'environnement à travers différents types de connexions (voir figure 4.4) :

Signal lien permettant de synchroniser des fonctions (lien synchrone) ;

Transfert de données lien permettant de synchroniser des fonctions avec des données (lien synchrone) ;

Partage de données données partagées entre plusieurs fonctions accessibles à tout moment (lien asynchrone).

Ces trois types de canaux de communication sont fréquemment utilisés dans la conception de systèmes informatiques et électroniques. Nous les retrouvons par exemple dans le mécanisme de Inter-Process Communication (IPC, Communication Inter-Processus) sous Linux (sémaphores, files de messages et mémoire partagée). Ils nous permettent ici de lier les fonctions de notre systèmes tout en s'abstrayant des technologies utilisées.

Nous sommes arrivés à cette structure fonctionnelle en suivant une démarche itérative, consistant à confronter de manière théorique chaque modèle que nous avions conçu à des cas d'utilisation plus ou moins complexes. Des cas concrets d'utilisation nous ont aussi été proposés par des biologistes de l'IGDR. Ainsi nous avons conçu et modifié notre système jusqu'à ce qu'il puisse théoriquement réaliser toutes ces expériences.

4.2.2.1 Description de la fonction *Réception images*

Chaque image capturée par la caméra est prise dans un contexte particulier défini par l'état dans lequel se trouve le microscope lors de son acquisition. Nous avons donc deux éléments de mesure à prendre en compte pour asservir le microscope :
— les images devant être analysées ;
— l'état du microscope lorsqu'une image est acquise.

En plus de l'image en tant que telle, cette dernière donnée est une information importante de l'asservissement. En effet, il est nécessaire de savoir dans quel état sont les périphériques du microscope (platine, focus, etc) lors de l'acquisition de l'image afin de déterminer leurs nouveaux paramètres dans le cas de l'acquisition d'une image d'intérêt. Nous avons donc décidé d'associer l'état du microscope à chaque image acquise.

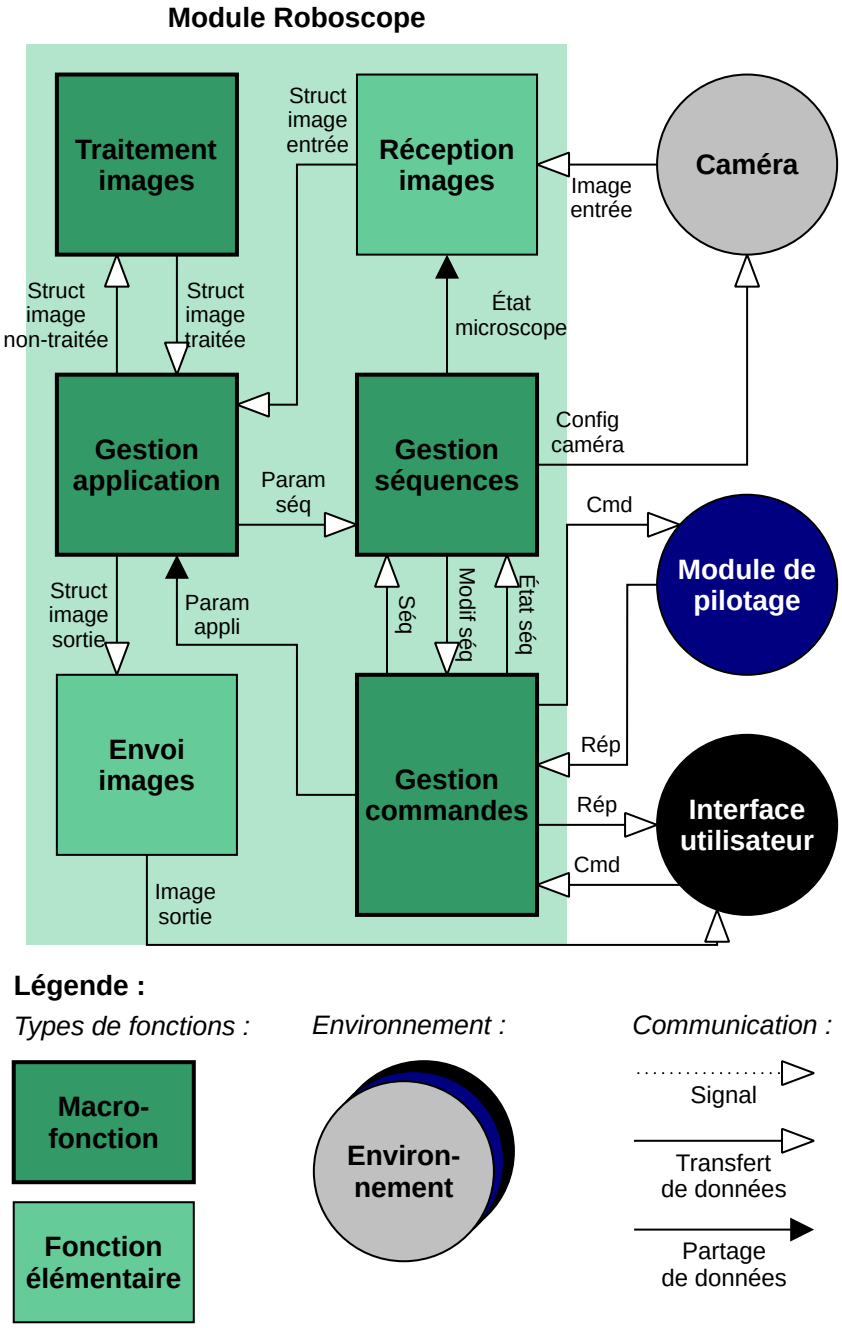


FIGURE 4.4 – Schéma fonctionnel du module *Roboscope* et légende correspondante. Les macro-fonctions sont des ensembles contenant plusieurs fonctions élémentaires. Les signaux *Cmd* et *Rép* signifient respectivement commandes et réponses. Les abréviations *séq*, *struct*, *config*, *modif*, *appli* et *param* signifient respectivement séquence, structure, configuration, modification, application et paramètre.

Ainsi, la fonction *Réception images* est une fonction élémentaire du système, permettant dans un premier temps de recevoir les images brutes retournées par la caméra via l'entrée *Image entrée* et d'y associer l'état dans lequel est le microscope à son acquisition via l'entrée *État microscope*. Cette dernière est mise à jour par *Gestion séquences* que nous décrivons dans la section 4.2.2.4. Cette image à laquelle est associé son contexte d'acquisition est ensuite envoyées à la fonction *Gestion application* via la sortie *Struct image entrée*[2].

4.2.2.2 Description de la fonction *Traitement images*

Afin d'asservir le microscope, les images retournées par la caméra doivent être analysées et/ou transformées. Du fait de notre volonté de créer un système générique, il est nécessaire de pouvoir réaliser différents types d'analyse. C'est pourquoi la fonction *Traitement images* est en réalité un ensemble de fonctions, laissant ainsi la possibilité à un potentiel utilisateur d'y intégrer ses propres fonctions.

Pour des raisons de compatibilité avec les autres fonctions du système, nous imposons un format que toutes les fonctions de *Traitement images* doivent suivre :

- l'entrée *Struct image non-traitée* doit contenir l'image à traiter avec son contexte d'acquisition ;
- de la même manière que l'état du microscope, les résultats de traitement doivent être stockés à la structure de l'image sur laquelle ils ont été obtenus ;
- la sortie *Struct image traitée* doit être au même format que *Struct image non-traitée*, contenant les résultats de traitement.

Une décision est ensuite prise par la fonction *Gestion application* (plus précisément la fonction *Prise décision* présentée dans la section 4.2.2.3), en fonction des résultats d'analyse stockés dans la structure d'image *Struct image traitée*.

L'intérêt de ce mécanisme est de répondre à notre objectif de généricité en permettant le traitement d'une image simplement. Il laisse aussi la possibilité d'enchaîner les traitements sur une même image tout en conservant le contexte et les résultats des traitements précédents. L'ordre dans lequel sont exécutées ces fonctions de traitement est géré par la fonction *Gestion application*.

4.2.2.3 Description de la fonction *Gestion application*

Les fonctions de *Traitement images* ont pour but d'analyser les images en provenance de *Réception images*. Il est donc nécessaire de lier ces fonctions pour créer une chaîne de traitement. Aussi, afin de réaliser l'asservissement, notre système doit être capable de prendre des décisions sur les images traitées pour modifier les modalités d'acquisition. C'est donc le rôle de la macro-fonction *Gestion application* (voir figure 4.5) d'assurer :

- la redirection des images au sein du système vers une fonction d'analyse d'image ou l'ordinateur (fonction élémentaire *Aiguillage images*) ;

2. Nous appelons structure d'image (ou *Struct image*) les images auxquelles des données telles que l'état du microscope ou les résultats d'analyse sont stockés.

— la prise de décision du changement de modalité d'acquisition en fonction des résultats d'analyse d'images [3] (fonction élémentaire *Prise décision*).

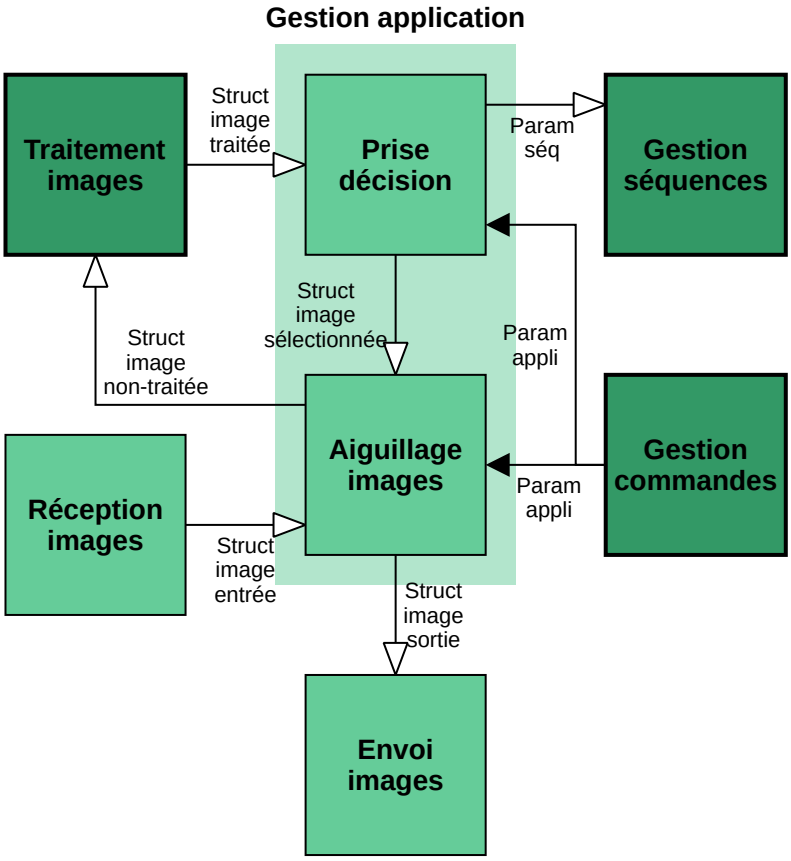


FIGURE 4.5 – Schéma fonctionnel de la macro-fonction *Gestion application*. Les abréviations *séq*, *struct* et *param* signifient respectivement séquence, structure et paramètre.

En fonction de l'expérience devant être réalisée, l'ordre dans lequel sont exécutées les fonctions de traitement et le type de décision devant être prise peuvent être différents. *Aiguillage images* et *Prise décision* sont donc des fonctions centrales de notre système, permettant de définir une expérience de microcopie intelligente. C'est pourquoi nous avons choisi de les grouper dans une même macro-fonction.

Aiguillage des images Le rôle de la fonction élémentaire *Aiguillage images* est donc de diriger les images vers les fonctions *Traitement images* afin de les analyser ou vers *Envoi images* afin de les envoyer à l'interface utilisateur. Cet aiguillage des images

3. Ces résultats d'analyse sont générés par la fonction *Traitement images*. Ils peuvent la représenter des caractéristiques de l'image, des probabilités d'avoir observé un objet d'intérêt, etc.

prend en compte le contexte d'acquisition de l'image (dans quelle modalité elle a été prise par exemple), ainsi que les analyse et/ou transformées qui lui ont été appliquées par *Traitement images*. Ainsi, sur cette base, l'utilisateur peut théoriquement réaliser différentes expériences en configurant cet aiguillage (via l'entrée *Param appli*).

Prise de décision Enfin, les images traitées par les fonctions *Traitement images* (via *Struct image traitée*) doivent être envoyées à la fonction élémentaire *Prise décision* dans le but de prendre deux décisions :

— l'image doit-elle être renvoyée vers la fonction *Gestion application* pour un autre traitement ou pour être envoyée vers l'interface utilisateur ;
— une autre séquence d'acquisition doit-elle être exécutée.

Cette décision est prise sur la base des résultats de l'analyse de chaque image. Si ces résultats remplissaient des critères particulier (probabilité de détection d'une cellule d'intérêt supérieure à un seuil donné par exemple), le rôle de cette fonction est donc de déclencher l'exécution d'une nouvelle modalité d'acquisition avec des nouveaux paramètres (par exemple, modification des coordonnées de la platine pour centrer la cellule d'intérêt puis lancement d'un time-lapse), et/ou de retourner l'image traitée vers *Aiguillage images*.

Finalement, ces deux fonctions élémentaires sont configurables (aiguillage des images, paramètres de prise de décision, paramètres à modifier dans les séquences) via l'entrée *Param appli*, permettant ainsi d'adapter le comportement du système à un grand nombre d'applications.

4.2.2.4 Description de la fonction *Gestion séquences*

Nous avons jusqu'ici mentionné les fonctions permettant de manipuler les images au sein du système et de prendre des décisions quant aux résultats de leur analyse. Ces fonctions visent à :

— exécuter et paramétrer une nouvelle modalité d'acquisition lorsqu'un événement d'intérêt est détecté (*Gestion application*) ;
— connaître l'état du microscope lors de la capture d'une image par la caméra (*Réception images*).

Nous pilotons le microscope grâce au module de pilotage Inscoper. Celui-ci contrôle les acquisitions en les organisant sous forme de séquences (voir section 3.4.1). C'est pourquoi nous avons conçu une fonction dans notre système, que nous avons appelé *Gestion séquences* permettant d'interpréter ces séquences d'acquisition et de les modifier. Nous présentons le schéma fonctionnel de cette fonction dans la figure 4.6.

Modification des modalités d'acquisition Nous avons dans un premier temps créé une fonction élémentaire au sein de *Gestion séquences* pour gérer la modification des modalités d'acquisition, que nous appelons *Modification séquences*. Cette fonction a donc pour but d'intégrer les paramètres calculés par la fonction *Gestion application* à la séquence correspondante, qui peut être ensuite exécutée. Ces modifications sont envoyées via la sortie *Modif séq* vers la fonction *Gestion commandes*, et seront ensuite envoyées

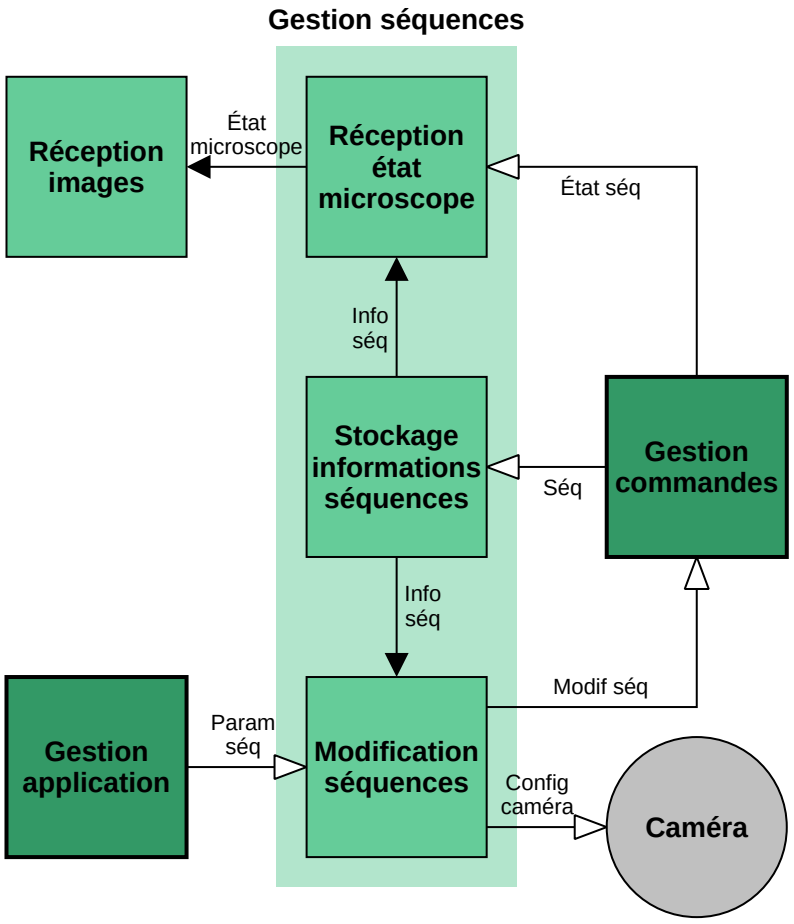


FIGURE 4.6 – Schéma fonctionnel de la macro-fonction *Gestion séquences*. Les abréviations *séq*, *config*, *info* et *param* signifient respectivement séquence, configuration, information et paramètre.

au module de pilotage par cette dernière sous forme de commandes. Si ces modifications concernent des paramètres de la caméra, celles-ci sont envoyées via la sortie *Config caméra*.

Réception de l'état du microscope Le deuxième rôle du *Gestion séquence* est de recevoir des informations sur l'état du microscope lors de la capture d'une image. Nous avons donc créé la fonction élémentaire *Réception état microscope*, recevant et interprétant les réponses retournées par le module de pilotage (voir section 3.4.1) via son entrée *État séq*. Certains périphériques du microscope permettent de lire l'état dans lequel ils se trouvent, comme certaines platines permettent de lire leur position en XY. Ce n'est en revanche pas le cas de tous les périphériques. La simple confirmation qu'une fonction de pilotage a bien été exécutée par le module de pilotage (voir section 3.4.1) n'est donc pas une information suffisante pour connaître l'état précis du microscope. Le système doit donc garder connaître les fonctions devant être exécutées lors de la séquence ainsi que leur paramètres.

Réception des paramètres des séquences d'acquisition Souhaitant laisser la possibilité d'asservir n'importe quel périphérique, nous avons aussi créé une fonction élémentaire, que nous appelons *Stockage informations séquences*, permettant de stocker au sein du système les différentes informations concernant les séquences exécutées par le module de pilotage, comme la liste des paramètres des fonctions de pilotage devant être exécutées (positions de platine, puissances de lasers, etc). Ces informations sont accessibles par les fonctions *Réception état microscope* et *Modification séquences* via leurs entrées *Info séq*. Ainsi, la fonction *Réception état microscope* peut mettre à jour sa sortie *État microscope* en utilisant les informations concernant les fonctions devant être exécutées (via *Info séq*) à chaque réception d'une confirmation d'exécution de fonction via son entrée *État séq*.

4.2.2.5 Description de la fonction *Gestion commandes*

Les modifications de modalités d'acquisition générées par *Gestion séquences* doivent ensuite être transmises au module de pilotage pour être prise en compte. Comme mentionné dans la section 4.2.1, nous utilisons le même protocole de communication entre le module de pilotage et l'interface graphique afin d'interagir avec et d'éviter tout problème de compatibilité. Nous avons donc conçu une macro-fonction servant d'interface que nous appelons *Gestion commandes* et que nous présentons figure 4.7. Cette fonction permet de décoder ou de créer ces commandes. Elle consiste en quatre fonctions élémentaires gérant respectivement l'envoi et la réception des commandes vers et en provenance de l'interface utilisateur, et du module de pilotage. Nous synchronisons ces fonctions entre elles avec différents signaux afin de maîtriser le flux de commandes et d'éviter la perte d'informations.

Cette décomposition de la fonction *Gestion commandes* nous permet de rendre indépendants la gestion des modalités d'acquisition et le protocole de communication entre les modules. Ainsi, si la manière donc sont gérées les commandes vient à changer, les autres fonctions du système n'en seront que peu impactées. De plus, cela nous permet de facilement traiter les cas où des commandes doivent être directement adressées de

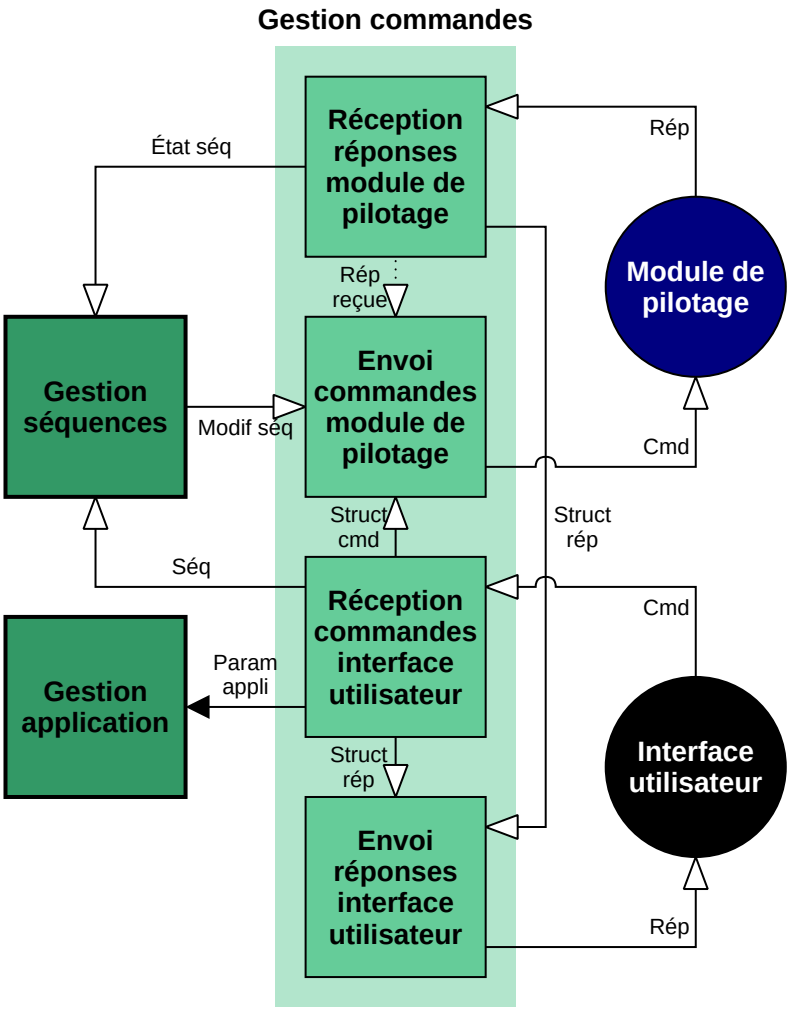


FIGURE 4.7 – Schéma fonctionnel de la macro-fonction *Gestion commandes*. Les signaux *Cmd* et *Rép* signifient respectivement commandes et réponses. Les abréviations *séq*, *struct*, *appli*, *modif* et *param* signifient respectivement séquence, structure, application, modification et paramètre.

l'interface utilisateur au module de pilotage, ce qui fait partie de nos contraintes (voir section 4.1). En effet, avec notre architecture, ces commandes peuvent être directement transférées entre *Réception commandes interface utilisateur* et *Envoi commandes module de pilotage*, donnant ainsi un lien direct entre l'interface utilisateur et le module de pilotage sans avoir à les connecter physiquement.

4.2.2.6 Description de la fonction *Envoi images*

Enfin, les images d'intérêt sélectionnées par la fonction *Gestion application* doivent être retournées vers l'interface utilisateur pour être affichées et stockées sur l'ordinateur. Aussi, il est intéressant de pouvoir retourner les différentes données attachées aux images au cours de leur passage dans le système, comme l'état du microscope ou les résultats de traitement (par exemple la probabilité d'avoir observé une cellule d'intérêt).

Nous avons donc ajouté une dernière fonction élémentaire à notre système servant d'interface entre l'interface utilisateur et le système pour l'envoie des images et de leurs données (via la sortie *Image sortie*).

4.2.2.7 Conclusion sur la conception

Finalement, l'architecture fonctionnelle que nous avons conçu en confrontation à des applications, envisagées théoriquement, nous a permis de mettre en place un modèle répondant aux différentes contraintes que fixaient ces applications. Jusqu'ici, cette confrontation à un grand nombre d'applications plus ou moins complexes n'a pas mis notre modèle théorique en défaut, montrant ainsi sa robustesse. Cela nous a permis de le breveter (Balluet et al. 2020). Ainsi, nous nous sommes appuyé sur ce modèle pour implémenter les différentes fonctions sur un support matériel.

4.2.3 Implémentation sur un système embarqué

Une fois notre modèle théorique mis en place, nous souhaitons à présent implémenter notre système d'asservissement sur un support matériel afin de l'interfacer avec le module de pilotage Inscoper, un ordinateur exécutant l'interface utilisateur et la caméra du microscope. Les ordinateurs sont des appareils multi-usage (utilisés en microscopie pour piloter des microscopes) pouvant exécuter plusieurs tâches différentes, sur lesquels est exécuté le système d'exploitation Windows dans la majorité des cas. Cela peut créer des problèmes de latences et d'instabilités. L'utilisation d'un ordinateur comme support matériel n'est donc pas compatible avec une application devant être exécutée en temps-réel. Nous nous sommes donc tournée vers une implémentation sur système embarqué avec une architecture ARM. En effet, ARM est une architecture de type Reduced Instruction Set Computer (RISC, Processeur à Jeu d'Instrutions Réduit) connue pour être plus prédictible en termes de temps d'exécution que les architectures de type Complex Instruction Set Computer (CISC, Processeur à Jeu d'Instrutions Étendu). Cela la rend plus appropriée pour les applications temps-réel.

Du fait de notre manipulation d'images de microscopie dont la taille peut être importante (au maximum $2\,048 \times 2\,048$ pixels codés sur 16 bits, soit 8 MiB avec la

Hamamatsu ORCA-Flash4.0 que nous présentons section 3.2 par exemple), nous avons donc anticipé l'utilisation de tâches de calcul intensif et nécessitant une quantité de mémoire importante (plusieurs dizaines voir de centaines de MB). De plus, nous avons besoin d'un support facilement programmable et configurable afin d'y exécuter nos fonctions de pilotage et de gestion d'application (fonctions génériques). Ainsi, nous avons choisi une architecture adaptée à ce genre de traitement en nous tournant vers les cartes embarquées Jetson proposées par NVIDIA. Nous pouvons retrouver ces systèmes grand public dans le milieu de la microscopie pour le calcul d'autofocus (Castillo-Secilla et al. 2017) ou pour la détection et la classification de cellules (Waithe et al. 2020). Notre équipe ayant pour ambition de travailler sur des méthodes d'apprentissage profond dans le futur, nous avons choisi de concevoir et de tester notre système sur un modèle ayant les ressources matérielles pour exécuter ce genre d'algorithme, à savoir la NVIDIA Jetson AGX Xavier (voir section 3.1.1).

Ce système embarque par défaut le système d'exploitation Ubuntu (Linux) adapté par NVIDIA pour leurs différents modèles de systèmes embarqués. Il est aussi possible d'y installer facilement des librairies d'analyse d'images telles que OpenCV (*NVIDIA Embedded Systems* 2020 ; Itseez 2015). C'est pourquoi, dans notre contexte de réalisation d'une preuve de concept, nous avons opté pour ce système d'exploitation, qui nous permet ainsi d'exécuter nos fonctions en parallèle (multi-thread) et d'accéder facilement aux différents périphériques de la carte.

Nous programmons les différentes fonctions que nous avons conçues en utilisant le langage C++. Faisant partie des langages de programmation compilés, le C++ est adapté aux applications temps-réel. Il fait aussi partie des langages les plus performants (Gouy 2020) en termes de temps d'exécution. Aussi, nous avons choisi le C++ au C pour sa syntaxe orientée objet nous permettant l'encapsulation de nos différents outils, ainsi que pour la disponibilité de nombreuses librairies standards tels que des conteneurs ou encore des outils de gestion intelligente de la mémoire.

Parmi le choix du matériel, nous avons dû choisir une caméra scientifique à laquelle notre système est connecté. Une difficulté dans le domaine de la microscopie est que la plupart de ces caméras sont utilisables via des logiciels spécialisés (généralement fournis par le fabricant), ou bien grâce à des librairies compilées pour les architectures de processeur x86, souvent pour une exécution sous Windows. Or, nous souhaitons utiliser la caméra avec la carte NVIDIA Jetson, donc dans un environnement Linux sur une architecture ARM. C'est pourquoi nous avons choisi d'utiliser la caméra Hamamatsu ORCA-Flash4.0 C11440-22C (*Hamamatsu* 2020) via son API compilée pour ARM sous Linux (voir section 3.2).

4.2.3.1 Connexion des modules

En tout premier lieu, les différents modules, à savoir la caméra, le module de pilotage Inscoper, l'ordinateur exécutant l'interface utilisateur et la NVIDIA Jetson exécutant notre système, doivent être connectés entre eux.

Connexion de la caméra Nous connectons la caméra à la NVIDIA Jetson via USB3 du fait qu'aucun port Camera Link n'est disponible sur cette dernière (*NVIDIA Embedded Systems* 2020 ; *Hamamatsu* 2020).

Connexion du module de pilotage De par sa conception, le seul port permettant la configuration du module de pilotage est un port USB2. Nous utilisons donc un port USB3 de la NVIDIA Jetson pour communiquer avec ce module.

Connexion avec l'ordinateur Notre système ayant pour but de retourner entre autres des images vers l'ordinateur, il est en théorie nécessaire d'utiliser un port suffisamment rapide pour transférer au moins autant d'images vers celui-ci que le peut la camera. Cela représente un débit d'au moins 240 MiB/s (30 images de $2\,048 \times 2\,048$ pixel de 16 bits par secondes comme décrit dans la section 3.2). Une connexion en USB3 comme avec la caméra semblait donc adéquat. En revanche, la NVIDIA Jetson et l'ordinateur ne comportent que des ports hôtes, rendant ainsi leur connexion difficile. C'est pourquoi nous nous tournons finalement vers une connexion via Ethernet dans un souci de simplicité dans le cadre de cette preuve de concept, malgré un débit théorique inférieur à ce dont nous avons besoin (125 MiB/s).

4.2.3.2 Programmation des tâches

Le modèle fonctionnel que nous avons présenté dans la section 4.2.2 est composé de plusieurs tâches que nous avons la possibilité d'exécuter simultanément. Afin de répondre à nos contraintes de temps en tirant pleinement partie de l'architecture multi-cœurs du CPU de la NVIDIA Jetson, nous avons décidé d'exécuter notre programme en multi-thread pour exécuter les différentes fonctions de notre modèle en parallèle sur les différents cœurs du CPU. Les librairies standard C++ incluent depuis la version C++11 différents outils permettant l'exécution et la synchronisation de plusieurs threads dans un processus.

Lancement des threads La décomposition en fonctions élémentaires de notre système nous permet d'exécuter chaque fonction dans des threads différents. Nous utilisons donc la fonction principale de notre programme comme tâche de fond afin de :

— lancer les différents threads avec leur fonction correspondante ;
— vérifier si un ou des threads se terminent en récupérant un code d'erreur ;
— attendre la fin de tous les threads si le programme doit se terminer.

Nous utilisons la fonction standard *async* afin de pouvoir lancer les threads et de récupérer la valeur de retour de leur fonction associée lorsque ceux-ci se terminent. En effet, cela nous est utile pour retourner des codes d'erreurs à la tâche de fond.

Synchronisation des threads Ces threads peuvent être amenés à partager des données et des informations. D'un point de vue fonctionnel, ces données correspondent aux différents canaux liant les fonctions de notre modèle (voir figure 4.4). En programmation multi-

thread, le partage de données impliquent de devoir soigneusement synchroniser le threads pour éviter les situations de compétition [4]. Nous avons donc implémenté :

- les signaux sous forme de sémaphores ;
- les transferts de données sous forme de files de messages ;
- les partages de données sous formes de pointeur vers une donnée accessible depuis plusieurs threads, et protégée par un mécanisme d'exclusion mutuelle.

Il s'agit d'outils fréquemment utilisés dans la IPC sous Linux adapté à notre modèle. Pour éviter toute situation de compétition, nous utilisons la classe standard d'exclusion mutuelle *mutex* afin de s'assurer que plusieurs thread n'accèdent pas à une donnée simultanément.

4.2.3.3 Acquisition et gestion des images

Les principaux éléments d'entrée de notre système sont les images capturées par la caméra qui sont aussi notre principal élément de mesure. Nous avons donc implémenté dans un premier temps la fonction *Réception images* (voir algorithme 4.1) et développé des outils permettant de les stocker et de les transmettre entre les threads de manière efficace.

Algorithme 4.1 : Réception images

Entrées : imageEntrée, étatMicroscope, listePériphAsservis
Sorties : structImageEntrée
configuration Caméra;
répéter
  recevoir imageEntrée **depuis** Caméra;
  structImageEntrée.pixels ← imageEntrée;
  structImageEntrée.numéro ← nouveauNuméro;
  structImageEntrée.type ← IMAGE_BRUTE;
  pour chaque *périphérique* ***dans*** *étatMicroscope[listePériphAsservis]* **faire**
    structImageEntrée.métadonnées[périphérique.nom] ← périphérique.état;
  fin
  structImageEntrée.métadonnées["numéro séquence"] ←
   étatMicroscope.numéroSéquence;
  envoyer structImageEntrée **vers** Aiguillage images;
jusqu'à *fin du programme*;

Configuration de la caméra Dans un premier temps, nous configurons la caméra depuis la NVIDIA Jetson en utilisant l'API de Hamamatsu. Selon notre modèle fonctionnel, celle-ci doit être configurée par la fonction *Gestion séquences* afin de pouvoir éventuellement l'asservir. Cependant, pour des raisons de simplicité dans l'idée d'une preuve de concept, nous figeons ses paramètres avec des valeurs par défaut. Afin d'acquérir les images lorsque

4. Situation dans laquelle plusieurs threads tentent d'accéder et de modifier une même donnée partagée au même moment, pouvant conduire à des bugs et des valeurs erronées de cette donnée.

le microscope est dans un état le permettant (platine à la bonne position), nous configurons la caméra pour que l'acquisition d'une image soit déclenchée par un signal généré par le module de pilotage Inscoper. Aussi nous figeons la résolution de la caméra au maximum pour nous mettre dans la situation la plus critique en termes de transfert de données dans la mémoire du système.

Acquisition des images Une fois la caméra configurée, nous utilisons l'API Hamamatsu pour lancer l'acquisition des images. Dans un premier temps, nous allouons de la mémoire tampon pour permettre à l'API de stocker les images reçues. Ce tampon fonctionne de la même manière qu'un tableau circulaire : lorsqu'il est plein, une nouvelle image acquise écrase la plus ancienne. Une fois le tampon alloué, l'API peut attendre la capture d'images.

Stockage des images Du fait de la résolution de la caméra, les images font partie des données les plus volumineuses de notre système (8 MiB en résolution maximale). Les opérations de copie de ces images peuvent donc être très coûteuses en temps d'exécution. Pour respecter nos contraintes temps-réel, nous voulons éviter autant que possible ces copies. C'est pourquoi nous utilisons la classe *Mat* de OpenCV pour stocker et copier les images en provenance de la caméra de manière efficace [5] (Itseez 2015). Chaque image reçue par la caméra est donc copiée une fois après sa capture dans une matrice *Mat* afin de ne pas être écrasée lors de leur analyse.

Gestion des métadonnées des images Notre conception fonctionnelle du système implique de pouvoir associer des informations aux images acquises tels que l'état du microscope (voir section 4.2.2.1), les résultats d'analyse (voir section 4.2.2.2) ou encore un numéro et un type [6] permettant d'identifier les images. Nous définissons donc un objet (classe), que nous appelons *Image*, associant la matrice de pixel stockée dans un objet *Mat* à un ensemble de métadonnées (voir figure 4.8). Afin de mettre au point un système générique, nous laissons la possibilité de stocker n'importe quel type de métadonnées au sein des images. Pour cela, celles-ci sont stockées dans une *map* [7] (*C++ reference* 2020) dont les clés sont des chaînes de caractères (classe *string*), et les données de type *void**. Bien que les chaînes de caractères soient moins rapides à traiter que des valeurs entières, leur utilisation comme type de clé nous permet d'obtenir un conteneur de métadonnées répondant à notre contrainte de modularité.

5. Pour chaque image, une seule copie de la matrice de pixel est créée en mémoire. Lorsque l'objet *Mat* est copié, seul le pointeur de la matrice est copié évitant ainsi la copie de tous les pixels. Le compte du nombre d'instances d'une même image est aussi maintenu par la classe *Mat*. La matrice de pixel est donc effacée de la mémoire si le nombre d'instance de l'image atteint 0.

6. Le type de l'image permet d'identifier s'il s'agit d'une image brute ou traitée par une fonction particulière.

7. Une *map* est un conteneur standard C++ permettant d'associer une donnée à une clé unique. Chaque clé permet d'accéder à la donnée à laquelle elle est associée. L'insertion et l'extraction de données ont une complexité logarithmique en fonction de la taille du conteneur (voir `https://en.cppreference.com/w/cpp/container/map`).

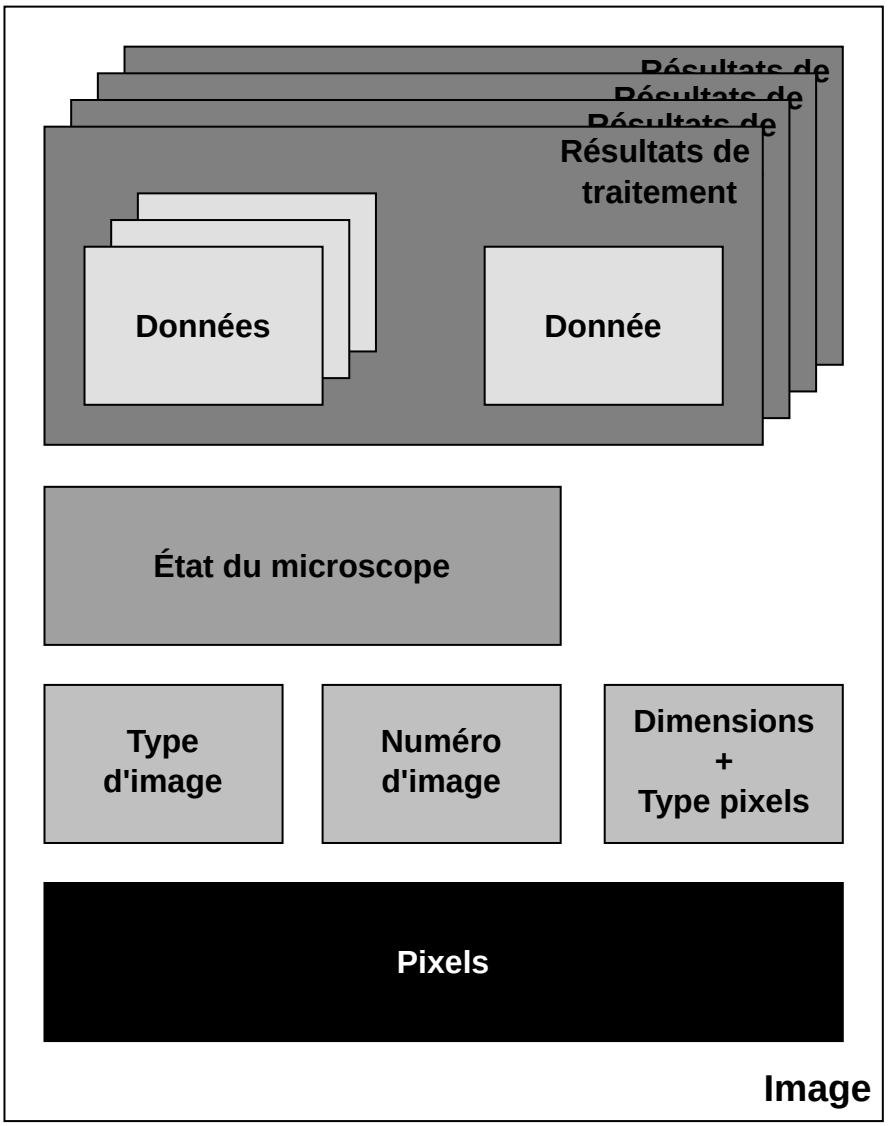


FIGURE 4.8 – Schéma d'un objet *Image* avec ses pixels et ses métadonnées. Les pixels, les dimensions et le type des pixels sont associés dans un objet *Mat*. Le type et le numéro d'image sont stockés directement dans l'objet *Image*. L'état du microscope et les résultats de traitement sont stockés dans différents champs d'une *map*.

Stockage de l'état du microscope Une fois le mécanisme de stockage de métadonnées dans les images mis en place, nous pouvons y stocker l'état du microscope à la réception de chaque image. Comme nous l'indiquons sur la figure 4.3, le microscope est composé de plusieurs périphériques. Nous créons donc un champ de métadonnées pour chaque périphérique, contenant la liste des paramètres indiquant son état (coordonnées de la platine en XY, position du focus en Z, type de laser, puissance de laser, etc). En revanche, en fonction de l'application réalisée, tous ces périphériques ne doivent pas être asservis. Stocker l'état de l'intégralité du microscope représente donc une perte de mémoire et de temps d'exécution. Par conséquent, nous avons décidé de ne stocker que l'état des périphériques devant être asservis dans les métadonnées des images. Pour cela, nous ajoutons un paramètre au sein du système (n'apparaissant pas sur la figure 4.4) listant ces périphériques. Pour une raison de simplicité, dans le cadre d'une preuve de concept, nous figeons ce paramètre pour une application particulière (que nous présentons dans la section 4.3). Il devra être configuré plus tard par l'interface utilisateur.

4.2.3.4 Implémentation des fonctions de traitement des images

Une fois les images acquises et stockées dans la mémoire du système embarqué, nous pouvons les envoyer aux fonctions *Traitement images* pour les analyser et/ou les transformer. Cet ensemble de fonctions étant modulable, plusieurs implémentations sont possibles en fonction de ce que souhaite réaliser l'utilisateur. Nous implémentons donc une fonction d'analyse spécifique à une application (que nous présentons dans la section 4.3), consistant à rechercher les cellules d'intérêt au sein d'une image pouvant contenir plusieurs cellules. Les images que nous utilisons représentant un volume important de données (plusieurs MiB), nous avons anticipé le fait que leur analyse représente la partie critique de notre système en termes de temps d'exécution. C'est pourquoi nous avons consacré une part importante de notre travail à l'élaboration d'une méthode d'analyse de ces images permettant de résoudre notre problème de détection de cellules d'intérêt en temps-réel, tout en respectant notre contrainte de généricité. Nous présentons le développement de cette méthode dans le chapitre 5.

4.2.3.5 Implémentation de la gestion d'une application

Nous lions ensuite les fonctions de traitement des images à celles de réception et d'envoi des images depuis la caméra et vers l'interface utilisateur en implémentant la fonction *Gestion application*. Comme nous le décrivons dans la section 4.2.2.3, cette fonction joue un rôle central dans notre système en définissant l'expérience devant être réalisée. Étant donné la diversité des applications que nous avons envisagées durant le développement du système, et de celles pouvant encore envisagées par de futurs utilisateurs, nous n'avons pas cherché de méthode d'implémentation générique de cette fonction. Au contraire, afin de pouvoir créer un grand nombre d'applications, nous avons décidé que le code des fonctions *Aiguillage images* et *Prise décision* serait généré et compilé par l'interface utilisateur. Toutefois, ces fonctions doivent suivre tout de même une structure définie par les algorithmes 4.2 et 4.3.

Aiguillage des images La structure de la fonction *Aiguillage images* consiste en l'imbrication de plusieurs instructions de sélection (*switch*), permettant d'envoyer une image d'entrée ou une image sélectionnée par *Prise décision* vers une fonction de traitement d'image ou vers l'interface utilisateur (via *Envoi images*), en fonction du type de l'image (image brute, traitée, etc) et de la séquence dont elle provient (voir algorithme 4.2). Cette structure étant facilement modifiable (l'ajout de paramètres à prendre en compte ne nécessite que l'ajout de *switch*), elle répond à notre contrainte de généricité.

Algorithme 4.2 : Aiguillage images

```
Entrées : structImageEntrée, structImageSélectionnée
Sorties : structImageNonTraitée, structImageSortie
répéter
    structImageReçue ← recevoir structImageEntrée OU
     structImageSélectionnée depuis Réception images OU Prise décision;
    suivant structImageReçue.type faire
        suivant structImageReçue.métadonnées["numéro séquence"] faire
            structImageNonTraitée ← structImageReçue;
            envoyer structImageNonTraitée vers Traitement images;
            OU/ET
            structImageSortie ← structImageReçue;
            envoyer structImageSortie vers Envoi images;
        fin
    fin
jusqu'à fin du programme;
```

Prise de décision La structure de la fonction *Prise décision* effectue différents types de tests sur les images qu'elle reçoit en fonction de leur type [8]. Si le ou les résultats remplissent une condition définie par l'application (supérieurs à un seuil par exemple), de nouveaux paramètres d'acquisitions (coordonnées de platine, position de focus, etc) sont calculés en fonction des métadonnées [9] de l'image reçue, puis envoyés à *Modification séquences* (voir algorithme 4.3).

La mise en place de la méthode de compilation des fonctions *Aiguillage images* et *Prise décision* à la configuration de l'expérience sort du cadre de cette thèse. Nous avons implémenté une application spécifique que nous présentons dans la section 4.3.

4.2.3.6 Gestion, interprétation et construction des commandes

Lorsqu'une image a été analysée et qu'une décision a été prise sur les résultats de son analyse, nous devons communiquer les nouveaux paramètres d'acquisition au module de

8. Type d'analyse ayant été appliquée.
9. Résultats d'analyse et état du microscope dans ce cas.

Algorithme 4.3 : Prise décision

```
Entrées : structImageTraitée
Sorties : structImageSélectionnée, paramSéq
répéter
    recevoir structImageTraitée depuis Traitement images;
    suivant structImageTraitée.type faire
        si structImageTraitée.métadonnées[RESULTATS] > seuil alors
            paramSéq ← calcul_param(structImageTraitée);
            envoyer paramSéq vers Modification séquences;
            structImageSélectionnée ← structImageTraitée;
            envoyer structImageSélectionnée vers Aiguillage images;
        fin
    fin
jusqu'à fin du programme;
```

pilotage le cas échéant. Nous présentons donc dans cette section l'implémentation des fonctions de gestion des commandes afin de communiquer avec l'interface utilisateur et le module de pilotage.

Notre système s'intégrant entre l'interface utilisateur et le module de pilotage, les commandes reçues par notre système peuvent soit être adressées à celui-ci, soit à un autre module auquel il est connecté (de l'interface utilisateur au module de pilotage et vice versa). Notre système doit donc être capable de :

— décoder les commandes lui étant adressées ;
— transmettre les commandes ne lui étant pas adressées sans les décoder inutilement ;
— encoder des commandes à partir d'informations qu'il aurait calculées.

De manière similaire aux objets *Image* que nous présentons dans la section 4.2.3.3, nous avons créé une classe que nous appelons *cmd_ interface* dont le but est de pouvoir être envoyée d'un thread à un autre de manière efficace. Cette classe comporte deux champs :

— un tableau d'octets contenant la trame brute de la commande ;
— une liste [10] de données organisées obtenue après décodage de la trame brute, ou stockant les données devant être encodées dans une trame de commande.

L'intérêt de conserver la trame brute au sein de cet objet est de pouvoir la transmettre directement au module de pilotage sans la décoder dans le cas ou elle est à destination de ce dernier. Cela évite le décodage et ré-encodage inutile de trame n'étant pas à destination du système.

Ainsi, la classe *cmd_ interface* comportant deux champs, nous veillons à ce que les copies inutiles de commandes (pouvant atteindre plusieurs dizaines de kB) soient évitées.

10. Nous stockons cette liste dans un objet *deque* du standard C++, offrant une complexité constante lors de l'ajout d'éléments et de l'accès à l'un d'entre eux (*C++ reference* 2020) (voir `https://en.cppreference.com/w/cpp/container/deque`).

Nous transmettons donc les deux champs de cette classe d'un thread à un autre en ne fournissant que les pointeurs vers la trame brute et la liste de données décodées.

4.2.3.7 Gestion des séquences

Une fois le système capable de prendre des décisions sur les résultats d'analyse des images, et capable de communiquer avec le module de pilotage ainsi que l'interface utilisateur, nous implémentons la fonction *Gestion séquences* afin de pourvoir :

— lire l'état du microscope lors de l'acquisition d'une image ;
— renvoyer des informations de pilotage au module de pilotage pour modifier les modalités d'acquisition.

L'implémentation de cette fonction vient ainsi fermer notre boucle d'asservissement.

Stockage des séquences et de leurs paramètres Nous avons dans un premier temps implémenté la fonction *Stockage informations séquences*. Celle-ci permet de récupérer les informations sur la ou les séquences devant être exécutées, nécessaire à la lecture de l'état du microscope. Pour cela, les commandes à destination du module de pilotage, envoyées par l'interface utilisateur, contenant des informations utiles à la lecture de l'état du microscope doivent aussi être envoyées à cette fonction afin de les stocker. Ces informations sont :

— les périphériques connectés au module de pilotage ;
— la liste des fonctions de pilotage devant être exécutées ;
— les séquences d'acquisition indiquant l'ordre d'exécution des fonctions de pilotage.

C'est pourquoi le rôle de cette fonction est de stocker les données de ces commandes dans différentes *maps* (voir algorithme 4.4) afin de permettre à la fonction *Réception état microscope* de facilement accéder aux informations permettant la mise à jour de l'état du microscope.

Récupération de l'état du microscope Le module de pilotage retourne une commande à chaque exécution d'une fonction de pilotage, contenant des informations permettant de l'identifier, et éventuellement des données si cette fonction consiste par exemple à lire l'état d'un périphérique. Par conséquent, l'estimation de l'état d'un périphérique du microscope peut se faire de deux manières :

— soit en interrogeant directement le périphérique sur son état (s'il possède une fonction le permettant) ;
— soit en utilisant les informations stockées par la fonction *Stockage informations séquences*.

Nous implémentons donc la fonction *Réception état microscope* de telle sorte qu'elle puisse déterminer si une réponse du module de pilotage contient ou non des paramètres sur l'état du périphérique en question. Elle peut ainsi extraire cette information dans la bonne source comme décrit dans l'algorithme 4.5.

Modification et lancement d'une nouvelle séquence Enfin, le second rôle de la fonction *Gestion séquences* est de modifier les modalités d'acquisition en interagissant

Algorithme 4.4 : Stockage informations séquences

```
Entrées : séq
Sorties : infoSéq
répéter
    recevoir séq depuis Réception commandes interface utilisateur;
    suivant séq.typeParam faire
        cas où AJOUT_PÉRIPHÉRIQUE faire
            paramSéq.ajouter(séq.paramPériphérique);
        fin
        cas où SUPPR_PÉRIPHÉRIQUE faire
            paramSéq.supprimer(séq.paramPériphérique);
        fin
        cas où AJOUT_FONCTION faire
            paramSéq.ajouter(séq.paramFonction);
        fin
        cas où SUPPR_FONCTION faire
            paramSéq.supprimer(séq.paramFonction);
        fin
        cas où AJOUT_SÉQUENCE faire
            paramSéq.ajouter(séq.paramSéquence);
        fin
        cas où SUPPR_SÉQUENCE faire
            paramSéq.supprimer(séq.paramSéquence);
        fin
    fin
jusqu'à fin du programme;
```

avec les séquences programmées dans le module de pilotage, lorsque la fonction *Prise décision* lui donne des paramètres de modifications de séquences. Nous effectuons ce changement de modalité en trois étapes :

- mise en pause de la séquence en cours ;
- modification des fonctions concernées avec les paramètres envoyées par *Prise décision* ;
- lancement de la séquence indiquée par *Prise décision*.

La mise en pause de la séquence en cours est nécessaire afin de permettre la prise en compte plus rapide des modifications des paramètres par le module de pilotage. Nous décrivons ce processus avec l'algorithme 4.6.

4.2.3.8 Conclusion de l'implémentation

Dans la continuité de notre travail de conception du système, nous avons prêté attention à respecter à la fois nos contraintes de généricité, de temps d'exécution lors de

Algorithme 4.5 : Réception état microscope

```
Entrées : étatSéq, infoSéq
Sorties : étatMicroscope
répéter
    recevoir étatSéq depuis Réception réponse module de pilotage;
    suivant étatSéq.typeInfo faire
        cas où NUMÉRO_SÉQEUNCE faire
            numéroSéq ← étatSéq.données;
            étatMicroscope.numéroSéq ← numéroSéq;
        fin
        cas où FONCTION_EXÉCUTÉE faire
            si étatSéq.données contient étatPériphérique alors
                étatPériphérique ← étatSéq.données.étatPériphérique;
            sinon
                fonctionExécutée ← étatSéq.données.fonctionExécutée;
                étatPériphérique ← infoSéq.paramFonction[fonctionExécutée];
            fin
            étatMicroscope.périphérique[numéroPériph] ← étatPériphérique;
        fin
    fin
jusqu'à fin du programme;
```

son implémentation sur le système embarqué. Cela nous a permis de mettre au point un système modulaire facilement adaptable aux différentes applications biologiques qu'il doit être amené à exécuter. Nous souhaitons après cette approche théorique valider expérimentalement notre système avec une application biologique que nous avons sélectionnée.

4.3 Expérience de validation

Comme nous le mentionnons dans la section 4.1, nous avons choisi une application particulière pour tester expérimentalement les propositions de conception et d'implémentation, ainsi que guider leur révision. Cette expérience composée de deux modalités d'acquisition nous permet de valider et de tester la conception du système et son implémentation sur la carte embarquée.

La première modalité consiste à scanner une lamelle contenant plusieurs cellules à la recherche d'échantillons en début de mitose ou à la transition métaphase-anaphase. Plus précisément, cette modalité consiste à déplacer la platine du microscope à différentes positions, et d'acquérir une image à chacune d'entre elles. Chaque image est analysée par notre système.

Lorsqu'une cellule d'intérêt est détectée, la première modalité d'acquisition doit être interrompue par la deuxième. Celle-ci consiste à déplacer la platine de telle sorte que la cellule d'intérêt soit placée au centre des images. Une fois la platine en place, un

Algorithme 4.6 : Modification séquences

```
Entrées : paramSéq, infoSéq
Sorties : modifSéq
répéter
    recevoir paramSéq depuis Prise décision;
    modifSéq ← pauseSéquenceCourante;
    envoyer modifSéq vers Envoi commandes module de pilotage;
    pour chaque paramFonction dans paramSéq faire
        modifSéq ← paramFonction;
        envoyer modifSéq vers Envoi commandes module de pilotage;
    fin
    modifSéq ← paramSéq.numéroSéquence;
    envoyer modifSéq vers Envoi commandes module de pilotage;
jusqu'à fin du programme;
```

time-lapse est réalisé à haute cadence d'acquisition. La recherche de cellules d'intérêt (première modalité) doit reprendre son cours à la fin de l'exécution du time-lapse.

Cette expérience nous permet de valider notre modèle fonctionnel en vérifiant si chaque fonction permet de réaliser sa tâche dans ce contexte particulier, et en observant l'évolution de leur état et de leurs sorties. De plus, nous validons le fonctionnement général du système en vérifiant que l'expérience est entièrement exécutée, et que les cellules d'intérêt sont bien détectées et retournées vers l'ordinateur.

Nous décrivons donc dans cette section les étapes permettant de paramétrer une telle expérience, ainsi que le comportement que nous attendons du module d'asservissement pour réaliser cette application.

4.3.1 Configuration d'une expérience de détection de cellules en mitose

La première étape consiste à configurer notre système et le module de pilotage pour réaliser l'expérience souhaitée.

Configuration des séquences La première étape de l'expérience consiste à configurer les différentes séquences d'acquisition devant être exécutées à l'aide de l'interface graphique. Notre application implique deux modalités d'acquisition. Nous découpons donc l'expérience en deux séquences. La séquence n°1, par laquelle l'expérience débute, consiste à scanner plusieurs positions d'une lamelle ou d'une plaque multi-puits. Nous devons donc saisir les différentes coordonnées selon la dimension XY que nous souhaitions scanner. Dans cette séquence, une seule position en Z est utilisée, ainsi qu'une seule source de lumière. La séquence n°2 correspond à un time-lapse. Nous sélectionnons donc l'intervalle de temps séparant chaque image. Dans cette séquence, une seule position en Z identique à la séquence n°1 est utilisée, ainsi qu'une seule source de lumière différente de cette dernière. De même une seule position en XY est programmée. En revanche, la valeur de cette

dernière est inconnue, car celle-ci doit être estimée par le module d'asservissement pour centrer la cellule d'intérêt. Cette séquence fixe donc les actions à exécuter mais pas ses paramètres.

Configuration du module d'asservissement La deuxième étape de configuration consiste à paramétrer le module d'asservissement en configurant les fonctions gérant l'application (*Gestion application*). Nous paramétrons plusieurs informations dans le module :

Analyse des images nous configurons la fonction d'analyse d'images pour identifier des cellules en début mitose ou à la transition métaphase-anaphase (nous décrivons notre méthode de détection dans le chapitre 5) ;

Aiguillage des images nous configurons la fonction d'aiguillage des images pour diriger les images brutes provenant de la séquence n°1 vers la fonction de détection de cellules d'intérêt, et les images brutes provenant de la séquence n°2 vers l'interface utilisateur ;

Prise de décision nous configurons la fonction de prise de décision afin de lancer la séquence n°2 lorsque la probabilité d'avoir observé une cellule d'intérêt est supérieure à un seuil de 0,9 ;

Paramètres asservis nous configurons les paramètres de la séquence n°2 devant être asservis, à savoir les coordonnées en XY de la platine.

4.3.2 Exécution de l'expérience

Une fois les séquences d'acquisition et le module d'asservissement configurés, l'utilisateur peut lancer son expérience que nous présentons sur le diagramme de flux de la figure 4.9. Cette application est exécutée par les différents threads du système, entre lesquels les images et différentes données du système sont échangées. Nous illustrons l'exécution attendue des différents threads dans les figures 4.10, 4.11 et 4.12.

Lancement de la séquence n°1 L'expérience est lancée en commençant par la séquence n°1 (séquence de scan).

Acquisition et analyse des images de la séquence n°1 Les images reçues lors de la séquence n°1 sont analysées par le module d'asservissement afin de chercher des cellules d'intérêt.

Détection d'une cellule d'intérêt Lors de l'analyse, si le module d'asservissement estime une forte probabilité de présence d'une cellule d'intérêt sur une image, celui-ci localise le centre de la cellule, et calcule les coordonnées auxquelles la platine doit être placée pour centrer la cellule en question sur les prochaines images. Cette image est retournée vers l'interface graphique pour permettre à l'utilisateur de vérifier les résultats d'analyse selon les paramètres d'aiguillage.

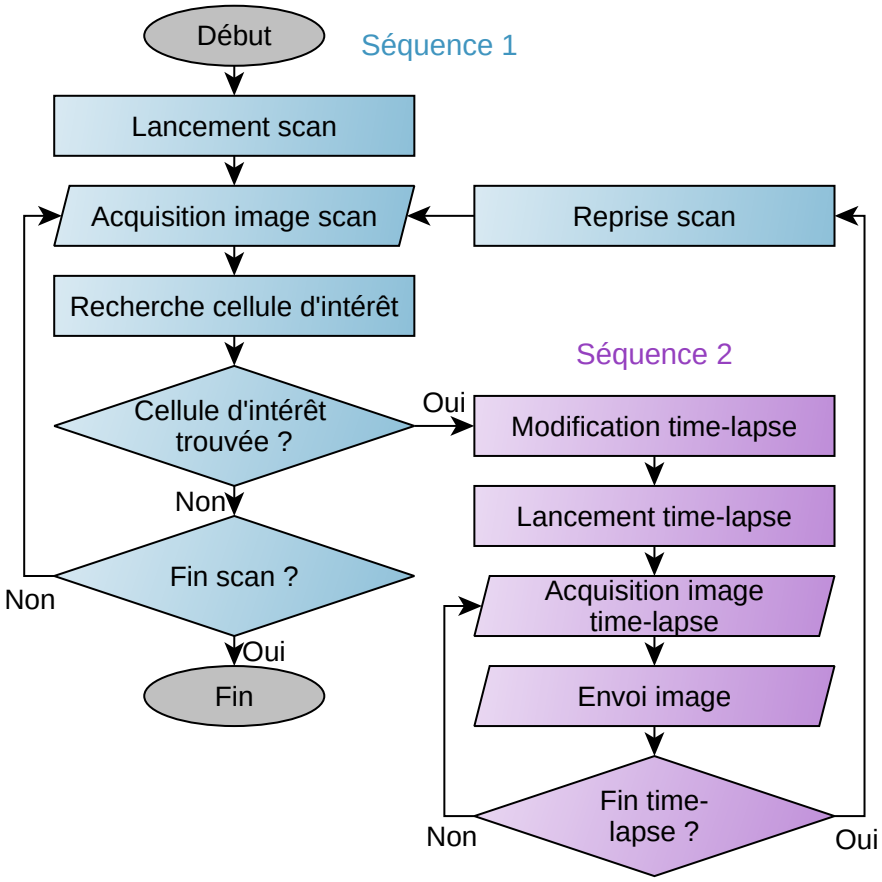


FIGURE 4.9 – Diagramme de flux de l'application exemple comportant une séquence de scan et recherche d'objets d'intérêt (en bleu, séquence 1) et une séquence de time-lapse (en violet, séquence 2).

Envoi des paramètres au module de pilotage Une fois les nouvelles coordonnées calculées (qui étaient jusqu'ici inconnues), celles-ci sont envoyées au module de pilotage pour compléter la séquence n°2. Celle-ci est donc prête à être exécutée.

Exécution de la séquence n°2 La séquence n°2 est lancée (séquence de time-lapse), venant ainsi interrompre la séquence n°1. Les images de la séquence n°2 ne nécessitent aucune analyse. Elles sont donc directement envoyées vers l'interface graphique pour être stockées sur l'ordinateur.

Reprise de la séquence n°1 et fin de l'expérience Une fois la séquence n°2 terminée, la séquence n°1 reprend son exécution là où elle avait été interrompue, à la recherche d'autres cellules d'intérêt. L'expérience se termine à la fin de l'exécution de la séquence n°1.

4.4 Résultats expérimentaux

Nous avons implémenté notre concept sur le système embarqué de manière progressive, les fonctions ayant été testées au fur et à mesure de leur implémentation. L'expérience que nous venons de présenter est utilisée pour tester le comportement du système complet sur un exemple biologique concret. Le système étant encore en développement, nous n'avons pas pu tester toutes ses fonctionnalités, et nous n'avons pu réaliser qu'un seul test de performances de l'acquisition des images avec la caméra.

4.4.1 Test de la capture des images

Un premier test que nous réalisons consiste à vérifier le bon fonctionnement de la caméra Hamamatsu lorsque celle-ci est utilisée avec le système embarqué. Du fait de notre intention d'acquérir des images à haut débit lors de l'exécution du time-lapse de la séquence n°2, nous cherchons à déterminer si nous pouvons atteindre le débit théorique maximal de la caméra (voir table 3.3 dans la section 3.2). Nous testons donc le débit d'images de la caméra Hamamastu lorsqu'elle est utilisée sur la carte NVIDIA Jetson via son API. Pour cela, nous avons besoin uniquement d'exécuter les fonctions de réception (depuis la caméra) et d'envoi (vers l'interface utilisateur) d'images.

Nous n'utilisons pas d'échantillon pour ce test. Les images acquises ne contiennent donc aucune information. Cela nous permet de paramétrer un temps d'exposition de la caméra en nous basant uniquement sur l'idée de tester son débit maximum. Nous configurons donc ce temps à 20 ms, laissant ainsi la possibilité d'atteindre le débit d'images maximal de 30 images/s de la Hamamatsu. Aussi, nous paramétrons le module de pilotage pour envoyer 100 signaux de déclenchement à la caméra.

Une fois la caméra et le module de pilotage configurés, nous pouvons lancer l'acquisition des 100 images. Afin d'estimer le débit d'images, nous mesurons l'instant d'acquisition de chaque image grâce à la librairie standard C++ *chrono*, nous permettant ainsi de mesurer le temps séparant l'acquisition de deux images. À la fin de l'acquisition, nous

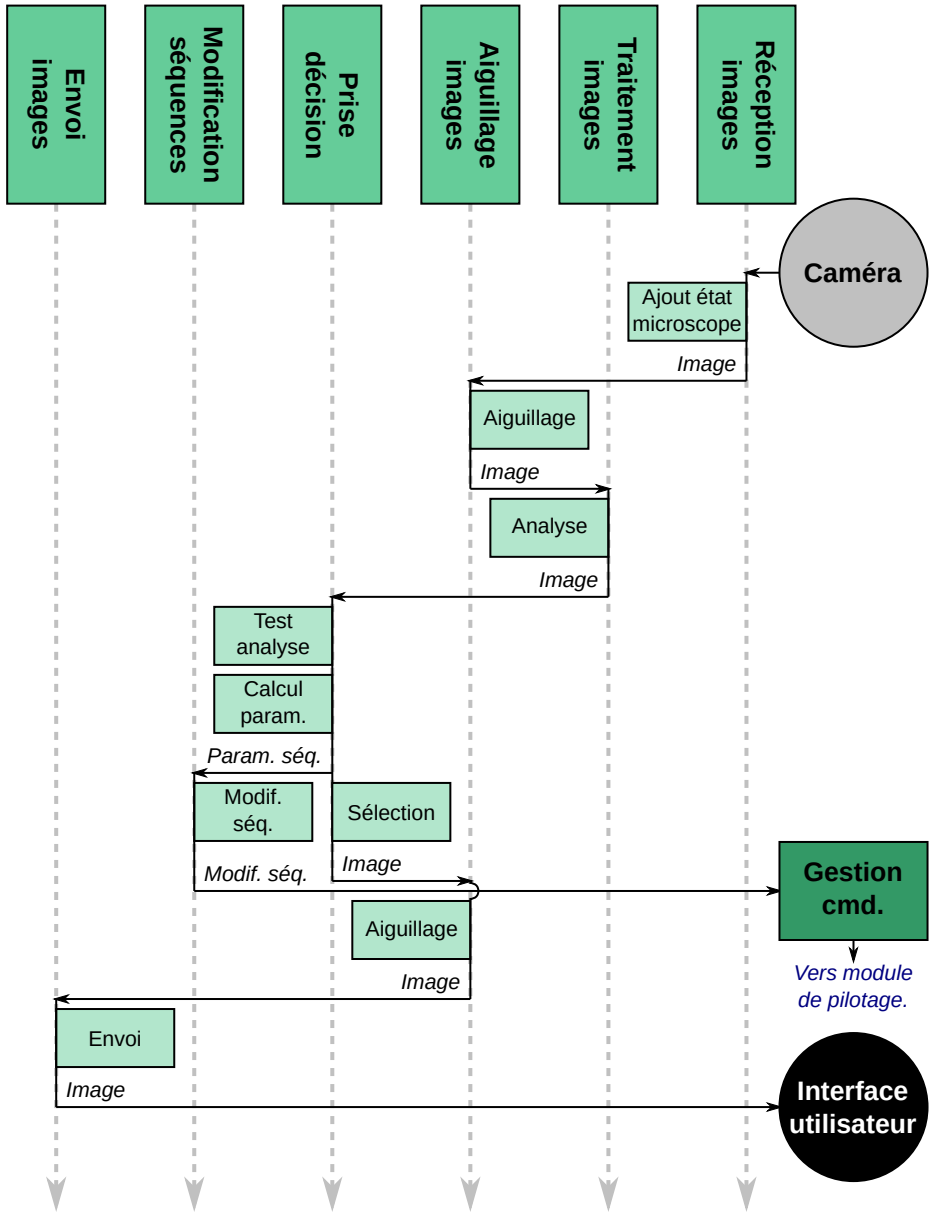


FIGURE 4.10 – Exécution de la séquence de scan (séquence n°1) de l'application exemple sur les différents threads. Les flèches pointillées décrivent l'exécution de chaque thread et les pleines les signaux entres ceux-ci. Les abréviations *séq*, *modif* et *param* signifient respectivement séquence, modification et paramètre.

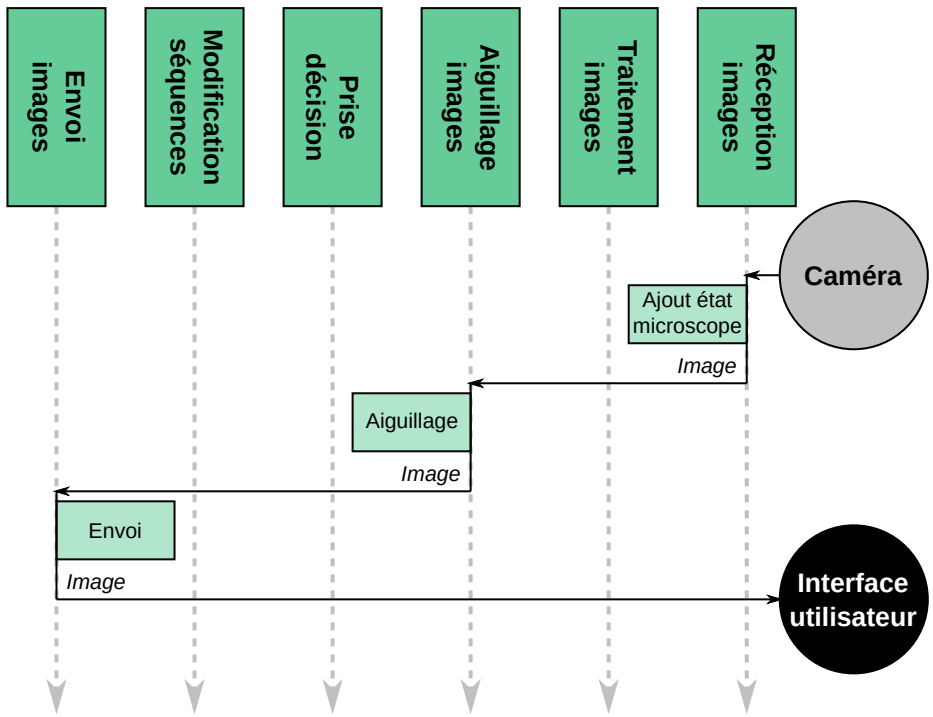


FIGURE 4.11 – Exécution du time-lapse (séquence n°2) de l'application exemple sur les différents threads. Les flèches pointillées décrivent l'exécution de chaque thread et les pleines les signaux entres ceux-ci.

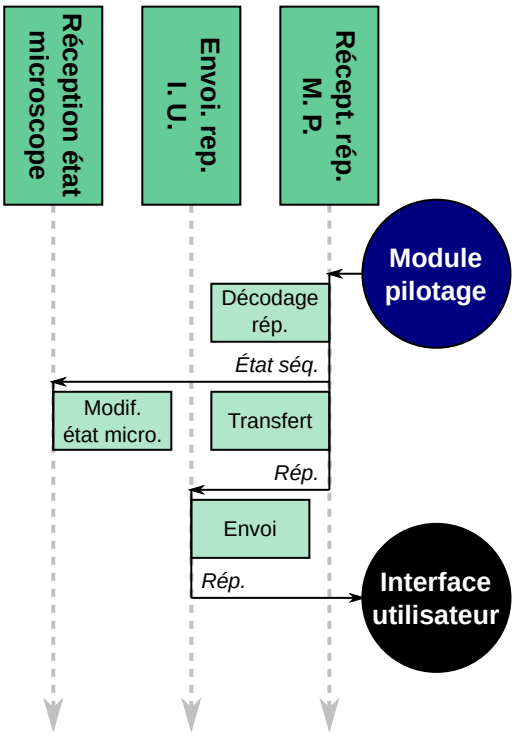


FIGURE 4.12 – Exécution de la réception de l'état du microscope sur les différents threads. Les flèches pointillées décrivent l'exécution de chaque thread et les pleines les signaux entres ceux-ci. Les abréviations *rép*, *séq*, *modif* et *micro* signifient respectivement réponse, séquence, modification et microscope. Les sigles I.U. et M.P. signifient respectivement interface utilisateur et module de pilotage.

constatons en premier lieu que sur les 100 images attendues, nous n'en recevons qu'environ 30. Nous avons aussi mesuré un débit d'environ 7 images/s. Or, nous observons que l'API Hamamatsu compte bien un nombre total de 30 images, nous montrant que les images manquantes ne sont pas envoyées à la NVIDIA Jetson.

Lors du développement des autres fonctions du système, nous constatons étrangement que le débit d'images augmente lorsque nous augmentons la charge du CPU. Pour vérifier cela, nous effectuons un test consistant à exécuter des boucles infinies dans 8 threads différents afin d'occuper les 8 cœurs du CPU au maximum. Nous pouvons dans ce cas atteindre un débit d'images d'environ 13 images/s. Nous avons fait l'hypothèse que l'augmentation de la charge du CPU pouvait augmenter sa fréquence de fonctionnement. Or, nous constatons que lorsque nous forçons cette fréquence de fonctionnement à son maximum, sans charger le CPU, le débit maximum est toujours de 7 images/s.

Jusqu'ici, nous n'avons pas réussi à identifier la source de ce problème. Bien que celui-ci soit handicapant pour réaliser une acquisition haut débit, il ne nous empêche pas de mettre en place et de tester le reste de notre système.

4.4.2 Tests préliminaires du système complet

Nous avons testé les différentes fonctions de notre système au fur et à mesure de leur implémentation sur le système embarqué. Jusqu'ici, nous validons le fonctionnement général de notre système sur l'observation de son comportement lorsque nous y exécutons l'expérience de détection des divisions cellulaire présentée dans la section 4.3.

4.4.2.1 Simulation de l'analyse des images

À ce stade, nous n'avons pas expérimenté notre système sur des échantillons biologiques réels. Nous avons donc choisi de réaliser un test général du système afin de valider son fonctionnement en remplaçant les fonctions d'analyse des images par un simulateur que nous avons réalisé.

Ce simulateur consiste à utiliser les images acquises par la caméra (ne représentant que du bruit), et d'y ajouter des résultats d'analyse aléatoires au même format que s'ils sont obtenus par la fonction d'analyse présentée dans le chapitre 5. Ces résultats consistent en une liste dont chaque élément correspond au résultat d'analyse d'une cellule, contenant :

- les coordonnées du centre de la cellule en question (utiles au calcul des coordonnées de la platine lors du time-lapse de la séquence n°2) ;
- la probabilité que la cellule soit en mitose (afin de simplifier le problème de détection du début de la mitose ou de la transition métaphase-anaphase).

Nous réglons ce simulateur pour générer des résultats pour un nombre de cellules compris entre 1 et 10 (valeurs empiriques choisies pour éviter trop de passages à la séquence n°2). Pour chacune d'entre elles, nous générons :

- des coordonnées en XY comprises entre 0 (inclus) et 2 048 (exclu) correspondants aux pixels de la caméra en résolution maximale (voir section 3.2) ;
- une valeur de probabilité comprise entre 0 et 1, avec une chance d'avoir une forte probabilité de représenter une mitose environ 24 fois plus faible qu'une interface

(valeur choisie sur la base d'une cellule se divisant durant 1 heure, toutes les 24 heures).

Ainsi, nous avons tous les éléments nécessaires à l'exécution complète du système sans échantillon biologique. Cela nous permet d'observer son fonctionnement en s'abstrayant des contraintes de temps de l'analyse réelle d'images.

4.4.2.2 Observations

Afin de valider le fonctionnement de notre système nous observons à ce stade son comportement lors de l'exécution de l'expérience que nous avons présentée. Nous manipulons principalement la platine lors de ce tests. Lors de la séquence de scan, nous cherchons donc à vérifier si les coordonnées de ses déplacements correspondent bien à ce que nous renseignons dans l'interface. Les coordonnées de la platine étant inconnues pour ce qui est de l'exécution de la séquence de time-lapse, nous vérifions que les bonnes coordonnées sont calculées à partir des :

— coordonnées absolues (en µm) auxquelles était la platine au moment de la capture de l'objet d'intérêt ;
— coordonnées (en nombre de pixels) de l'objet d'intérêt par rapport au bord supérieur gauche de l'image.

Nous vérifions donc la bonne conversion des coordonnées en pixels de l'objet d'intérêt vers des coordonnées en µm, qui sont ensuite ajoutées aux coordonnées absolues de la platine.

L'observation du comportement de chaque thread se fait au travers de l'écriture des actions réalisées par ces derniers dans un fichier *log*. Dans le cadre de ce test, nous observons le comportement de la macro-fonction *Gestion séquence* pour mesurer la réception et l'envoie des bonnes coordonnées.

De manière générale, nous observons bien la séquence de scan entièrement exécutée, en constatant le déplacement de la platine à toutes les positions ayant été configurées. Nous constatons aussi l'arrêt des mouvements de la platine à certains instants (correspondant à la détection d'un évènement d'intérêt généré par le simulateur), laissant ainsi place à la séquence de time-lapse pendant quelques secondes. Durant cette période, les images acquises par la caméra sont bien retournées à l'interface utilisateur. Enfin, une fois chaque time-lapse terminé, nous observons la reprise de la séquence de scan à sa dernière position, jusqu'à la fin de celle-ci, mettant ainsi fin à l'expérience. Aussi, nous avons pu analyser lors de cette expérience le comportement de chaque fonction et outil que nous avons développé pour valider leur bon fonctionnement.

Un point intéressant que nous observons est le phénomène d'interruption de la séquence de scan par celle de time-lapse. En effet, un de nos objectifs est de suivre une méthode de fonctionnement évènementielle et non séquentielle (voir section 4.1). Nous avons pour cela utilisé tout au long du projet des tâches (threads) et des modules (module de pilotage, caméra et interface utilisateur) étant exécutés simultanément.

Ainsi, nous cherchons vérifier le mécanisme de modification séquences d'acquisition en temps-réel. Lorsqu'une image est en cours de traitement par notre système, le module de pilotage entame le déplacement de la platine vers sa prochaine position. La détection d'un événement d'intérêt vient ainsi interrompre le scan pour exécuter le time-lapse à la

position déterminée par notre système. Lors d'un tel événement, nous observons donc le déplacement de la platine vers sa nouvelle position après avoir atteint sa prochaine position de scan. Cela montre ainsi la capacité de notre système à modifier des modalités d'acquisition d'images en temps-réel.

Toutefois, lors de nos observations du comportement du système, nous pouvons constater des cas où deux positions consécutives de la platine peuvent contenir un événement d'intérêt. Dans ce cas-ci, si l'image de la deuxième position en question est acquise avant l'interruption de la séquence de scan, celle-ci est analysée en même temps que la réalisation du time-lapse. Dans le cas d'un long time-lapse (plusieurs minutes), l'objet d'intérêt à la deuxième position peut avoir évolué de manière significative (ce qui peut être le cas lors de la division cellulaire), rendant ainsi l'analyse effectuée obsolète à la fin du time-lapse. Nous avons donc prévu d'implémenter une méthode dans notre système lui permettant de créer une courte séquence afin de réanalyser une position ayant été interrompue, avant de reprendre la séquence de scan.

4.4.3 Tests futurs

N'ayant pas testé toutes les fonctionnalités et capacités de notre système, nous envisageons de réaliser différents tests à ce stade.

Un premier test de fonctionnement que nous envisageons consiste à effectuer une vraie analyse d'image en utilisant la méthode que nous présentons dans le chapitre 5, sans utiliser d'échantillon réel dans un premier temps. Lors de ce test, les images acquises par la caméra (ne représentant que du bruit) seraient remplacées par des images représentant des cellules sur lesquelles les outils d'analyse pourraient être configurés et appliqués. Du fait de notre utilisation de OpenCV pour la gestion des images, nous prévoyons aussi d'utiliser cette librairie pour segmenter les images contenant plusieurs cellules en vignettes utilisables avec la méthode que nous présentons dans le chapitre 5 (Itseez 2015). Nous avons aussi identifié la librairie OpenNN (Martin et al. 2019) permettant la classification d'images avec des réseaux de neurones, proposant de nombreuses possibilité de configuration et adaptée à la méthode que nous présentons dans le chapitre 5. Suite à cela, nous pourrions finalement valider le fonctionnement de notre système complet sur des échantillons réels.

Enfin, ayant réalisé ce prototype avec des contraintes temps-réel, il est important de mesurer les performances de celui-ci. La seconde suite de tests que nous souhaitons réaliser consiste à mesurer le temps d'exécution des différentes fonctions sur le système embarqué. Nous souhaitons aussi analyser la répartition des ressources matérielles du système embarqué (charge du CPU, occupation de la mémoire, utilisation des entrées/sorties, etc) pour chaque fonction afin d'identifier les fonctions critiques nécessitant éventuellement d'être optimisées.

4.5 Conclusion et discussion

Dans ce chapitre, nous avons présenté le processus de conception d'un système embarqué permettant d'automatiser la microscopie. Nous nous sommes fixés différentes

contraintes permettant l'acquisition et l'analyse d'images de microscopie, ainsi que la modification des modalités d'acquisition des images du microscope, le tout en temps-réel. Nous avons donc conçu un modèle fonctionnel théorique réalisant une boucle d'asservissement du microscope, retournant des informations de pilotage à ce dernier en fonction des images analysées et du contexte dans lequel elles étaient acquises. En premier lieu, l'implémentation des tâches élémentaires constituant notre système, sur le système embarqué que nous avons ciblé, permet de réaliser les fonctionnalités définies dans notre cahier des charges en :

— gérant la réception et le stockage d'images pouvant provenir d'une ou plusieurs caméras de microscopie ;
— permettant l'intégration de plusieurs fonctions d'analyse d'images ;
— modifiant les modalités d'acquisition en fonction de décisions prisent sur les images analysées ;
— retournant uniquement les images d'intérêt pour l'utilisateur.

De par sa structure, notre modèle est capable d'interrompre des séquences d'acquisition en fonction des images analysées. Cela fait partie de nos contrainte et représente une valeur ajoutée par rapport aux systèmes d'automatisation existants (Conrad et al. 2011 ; Tischer et al. 2014). Cette capacité est basée sur un modèle de fonctionnement évènementiel. L'implémentation de ce modèle théorique sur un système embarqué montre ainsi la faisabilité d'un système de microscopie intelligente, respectant des contraintes temps-réel. Nous avons en effet observé le bon fonctionnement de ces différentes fonctionnalités, à l'exception de l'analyse des images, dont nous avons simulé les résultats pour valider le fonctionnement général de notre système. Particulièrement, nous avons bien observé le fonctionnement du mécanisme d'interruption, montrant son intérêt pour un pilotage du microscope en temps-réel.

Par définition, cet aspect de temps-réel est une notion relative à la vitesse d'évolution des processus biologiques que nous souhaitons analyser. En effet, un système comme Micropilot peut être considéré comme étant temps-réel malgré son fonctionnement séquentiel, lorsque ces processus sont plus longs que le temps de déplacement du microscope et celui de l'analyse des images basse résolution (Conrad et al. 2011). Dans notre cas d'études d'évènements rares et courts, la contrainte de temps est plus stricte, et les délais entres l'acquisition des images, les déplacements du microscope, et le déclenchement d'autres modalités d'acquisition peuvent provoquer des pertes d'information. Notre mécanisme d'interruption permet ainsi de nous rapprocher du temps-réel, en minimisant le temps entre la détection d'un tel évènement, et son analyse en imagerie complexe. En revanche, cela nécessite l'utilisation d'algorithmes d'analyse d'images respectant ces mêmes contraintes de temps.

Au-delà de la réalisation d'une preuve de concept, nous avons prêté beaucoup d'attention à la généricité de notre système, à la fois pour son modèle théorique et son implémentation ; cela lui donne une seconde valeur ajoutée. En effet, beaucoup de systèmes d'automatisation (Conrad et al. 2011 ; Tischer et al. 2014 ; Nitta et al. 2018) ou d'analyse d'image en temps-réel (Castillo-Secilla et al. 2017 ; Waithe et al. 2020) sont conçus pour réaliser des tâches très spécifiques, leur donnant ainsi peu de modularité.

Ayant pour objectif de nous diriger vers une solution industrielle, devant à la fois toucher un maximum d'utilisateurs et être facilement modifiable pour des questions d'évolutivité et de maintenance, cette contrainte de modularité est primordiale. Cette généricité a jusqu'ici été validée avec des scénarios théoriques d'utilisation plus ou moins complexes. En effet, d'un point de vue fonctionnel, notre système est capable de résoudre un grand nombre d'applications biologiques complexes, sans nécessiter d'importantes modifications dans son implémentation. Cependant, une validation théorique de la généricité de notre solution ne suffit pas à démontrer sa capacité à résoudre des problèmes, nécessitant de la rapidité par exemple. C'est pourquoi, cette généricité doit à présent être validée expérimentalement en y intégrant entre autres des contraintes de temps.

Pour implémenter notre système sur un support matériel, nous avons fait le choix d'un kit de développement grand public proposé par NVIDIA. C'est un choix mesuré dans notre contexte de développement d'une preuve de concept impliquant du calcul intensif, comme l'analyse d'images ou l'exécution d'algorithmes d'Intelligence Artificielle (IA). En effet, le choix d'une carte grand public nous permet de nous affranchir des contraintes du développement d'un système embarqué, pouvant nécessiter un important temps de conception. De plus, il s'agit d'un système ayant montré des performances intéressantes dans différentes applications de l'analyse d'images en temps-réel (Castillo-Secilla et al. 2017 ; Waithe et al. 2020 ; *NVIDIA Embedded Systems* 2020).

Toutefois, nous avons utilisé cette solution avec son système d'exploitation par défaut, qui est une distribution Ubuntu. Étant, un système aussi destiné à une utilisation sur ordinateur, ce choix n'est donc probablement pas le mieux adapté à une exécution temps-réel plus stricte sur un système dédié. Le choix d'un système d'exploitation plus optimisé à une exécution en temps-réel, et l'étude plus approfondie dans le choix d'un support matériel plus adapté, doivent donc être considérés. Ce choix doit cependant permettre une exécution en multi-thread de nos différentes tâches, ainsi que permettre l'utilisation de l'API Hamamatsu que nous utilisons pour piloter notre caméra. Cela nous contraint donc à utiliser un système d'exploitation de type Linux, optimisé pour une exécution en temps-réel.

Si le choix du système d'exploitation peut être discuté, il en va de même pour le support matériel. Bien que les modèles NVIDIA Jetson intègrent des éléments d'accélération matérielle pour l'analyse d'image, tels qu'un GPU, d'autres solutions telles que les Field-Programmable Gate Array (FPGA) ont montré des performances intéressantes ces dernières années par rapport aux CPU multi-cœurs et GPU (Asano, Maruyama et Yamaguchi 2009 ; D. Chen et D. Singh 2013 ; Siddiqui et al. 2019 ; Georgis, Lentaris et Reisis 2019). Aussi, une limitation des cartes de développement grand public, telles que les NVIDIA Jetson, est l'absence de ports de communication spécialisés, comme les ports *Camera Link* disponibles sur les caméras Hamamatsu (*Hamamatsu* 2020). En effet, l'utilisation de ces ports permettrait une acquisition des images plus rapide par rapport à l'USB3 disponible sur la plupart des cartes grand public.

Dans notre contexte où le système a pour but d'être commercialisé, sa flexibilité est un paramètre important. Au-delà des aspects de généricité dont nous parlons dans ce chapitre, la commercialisation d'un système de microscopie intelligente nécessite de

pourvoir le faire évoluer pour l'adapter aux applications futures. Il sera aussi intéressant de pour le maintenir facilement, en permettant aux ingénieurs d'Inscoper d'intervenir à distance. Or, ces aspects d'évolutivité et de maintenabilité sont aujourd'hui plus difficiles à atteindre avec des FPGA, leur utilisation nécessitant un temps de développement et une expertise beaucoup plus importants par rapport à la programmation sur CPU et GPU (Bacon, Rabbah et Shukla 2013).

Parmi les supports matériels existants aujourd'hui, nous pourrions aussi nous poser la question d'utiliser un ordinateur classique, pour y exécuter nos fonctions de prise de décision, de pilotage et d'analyse d'images. En effet, contrairement à la majorité des systèmes embarqués, ceux-ci peuvent présenter des ressources matérielles beaucoup plus performantes en termes de CPU multi-coeurs, de GPU (sous forme de cartes graphiques par exemple), et de mémoires. Leur modularité permet par exemple l'ajout de mémoires et de GPU pour de meilleures performances. En revanche, les ordinateurs sont dédiés à un usage général, et ne sont donc pas optimisés pour l'exécution des tâches de microscopie intelligente. Les tâches telles que le gestionnaire de bureau, et toutes les autres applications ouvertes lors de l'utilisation, peuvent ainsi ralentir le fonctionnement du système de microscopie intelligente. Au contraire, les systèmes embarqués sont dédiés à l'exécution des tâches spécifiques auxquelles ils sont attribués, et ne peuvent pas être ralentis par des applications tierces. Avec sa solution de pilotage de microscope, Inscoper a par exemple obtenu des performances trois fois plus élevées et une meilleure reproductibilité des temps de pilotage, en embarquant le pilotage du microscope dans un module dédie, par rapport aux solutions existante sur ordinateur. C'est pourquoi dans un contexte de temps-réel, l'utilisation d'un système embarqué semble mieux appropriée.

Aujourd'hui, la solution de microscopie intelligente que nous proposons est constituée de deux modules : le module de pilotage commercialisé par Inscoper, et module que nous présentons dans ce chapitre. Afin de trouver une solution intermédiaire entre la flexibilité d'une carte de développement et une solution performante comme un FPGA, nous pouvons nous poser la question de séparer les fonctionnalités de notre système de microscopie intelligente sur différents supports matériels. Cela permettrait par exemple d'exécuter la plupart de ces fonctions sur CPU, et de n'exécuter que les fonctions critiques en termes de temps d'exécution, ou nécessitant une connectique spécifique sur un ou des modules plus spécialisés.

Chapitre 5

Classification d'images de microscopie en temps-réel

5.1 Objectifs du travail

Nous avons conçu dans le chapitre 4 un système embarqué permettant l'asservissement d'un système de microscopie, faisant appel à des fonctions d'analyse à la volée d'images et devant retourner des informations de pilotage en fonction de ces images. Au-delà des aspects de précision et de stabilité, les asservissements doivent aussi répondre à des contraintes de temps. Les images de microscopie pouvant être composées de plusieurs centaines de milliers, voir de millions de pixels, nous avons fait l'hypothèse que leur analyse représenterait la partie critique de notre application en termes de temps d'exécution. C'est pourquoi nous avons dédié une importante part de notre travail à la conception de notre fonction de classification d'images en temps-réel.

Notre objectif était de réaliser la classification d'images de microscopie représentant des cellules à différents stades de la division cellulaire, dans l'idée de détecter des évènements rares et/ou courts tels que la transition métaphase-anaphase la mitose. Notre deuxième contrainte, au-delà des aspects temps-réel, était donc de réaliser une classification de qualité en termes de précision.

Enfin, une attention particulière a été accordée à la généricité et la facilité de modification lors de la conception de notre système dans le chapitre 4. Nous avons donc conçu notre algorithme d'analyse d'images dans l'objectif de pouvoir l'adapter simplement à la majorité des applications biologiques.

Ainsi, nous présentons dans ce chapitre une méthode de classification d'images de biologie dans différentes phases de la division cellulaire, respectant à la fois des contraintes de précision, de temps d'exécution et de généricité.

5.2 Présentation des bases d'images

Pour construire notre méthode d'analyse d'images, nous avons utilisé deux bases de données que nous présentons ci-après.

5.2.1 Base de démonstration du logiciel CellCognition

La première base d'images que nous utilisons est construite depuis la base de démonstration du logiciel CellCognition développé par Held et al. (Held et al. 2010). Cette base est composée de "time-lapses" réalisés avec des cellules humaines Hela Kyoto, exprimant différentes protéines fluorescentes. Nous avons utilisé les images de cellules marquées au noyau (H2B) et aux microtubules (α-tubuline). Ce choix a été fait de manière empirique du fait que seules les images représentant le noyau nous intéressent dans cette application. Cette base contient 3 time-lapses réalisés avec un objectif de grossissement 20x à différentes positions. Pour chacune d'entre elles, 206 images de 1392×1040 pixels codés sur 8 bits sont acquises.

L'annotation des images réalisée par Held et al. consiste en l'assignation d'une classe parmi 8 (1 en interphase, 6 phases de la division cellulaire, 1 en apoptose) au centre de certaines cellules (label). Notre objectif étant de classifier des cellules individuelles, nous avons extrait des vignettes de 71×71 pixels dont le centre correspond au label de la cellule comme présenté dans la figure 5.1. Le nombre de vignettes obtenues pour chaque classe est présenté table 5.1.

Classes	**Abréviation**	**Nombre d'images**
Interphase	inter	189
Prophase	pro	85
Pro-métaphase	prometa	64
Métaphase	meta	76
Début d'anaphase	earlyana	28
Fin d'anaphase	lateana	63
Télophase	telo	101
Apoptoses	apo	113

Table 5.1 – Liste des classes annotées sur les images de la base de CellCognition (Held et al. 2010).

Nous avions constaté deux points avec cette base de données pouvant poser problème dans le cas où nous utilisons des algorithmes d'apprentissage :

— la répartition des classes est déséquilibrée (table 5.1) ;

— une même cellule peut apparaître sur plusieurs vignettes à des instants différents.

Pour palier à ces deux problèmes, nous utilisons une méthode de bootstrap (voir section 3.5.2) afin d'obtenir un nombre similaire de vignettes pour chaque classe avec une seule instance de chaque cellule. Finalement, nous obtenons 20 vignettes par classe, à l'exception de la classe apoptose pour laquelle nous avons 19 vignettes, faisant un total de 159 images.

5.2.2 Base Mitocheck

Dans l'idée de valider notre méthode, nous comparons nos résultats avec une seconde base d'images qui nous a été fournie. Comme pour la base précédente, il s'agit de vignettes

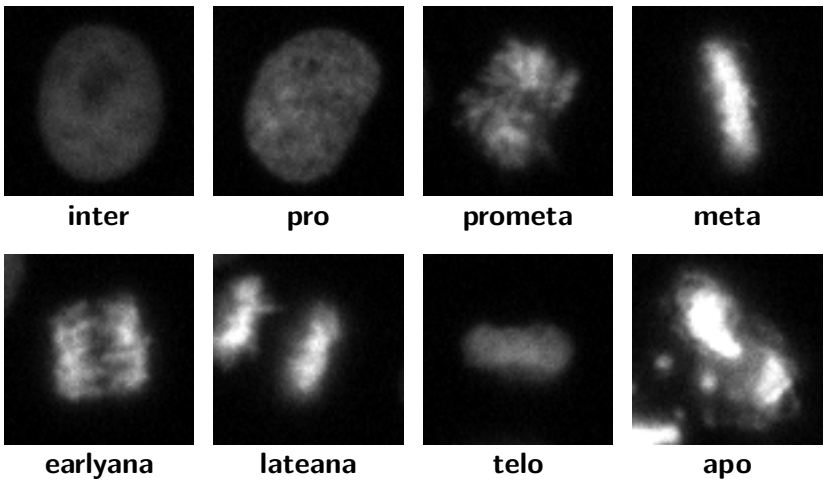


FIGURE 5.1 – Exemple de vignettes pour les différentes classes associées générées à partir de la base d'images de CellCognition (HELD et al. 2010).

représentant des cellules humaines Hela Kyoto mais ne présentant qu'un seul marqueur GFP au noyau, acquises avec un objectif de grossissement 10x. La base nous a directement été fournie sous forme de vignettes de 64×64 pixels codés sur 8 bits.

Cette base de vignettes contient 11 classes présentées dans la table 5.2. Elles représentent certaines phases du cycle cellulaire ainsi que des cas de défauts lors de la division. Un exemple de vignettes dans chaque classe est donné figure 5.2.

Classes	**Abréviation**	**Nombre d'images**
Interphase	inter	771
Gros Noyau	large	212
Noyau Allongé	elong	310
Pro-métaphase	prometa	439
Métaphase	meta	154
Anaphase	ana	177
Apoptose	apo	599
Polylobé	polylob	322
Grappe	grape	150
Problème Alignement Métaphase	metaalign	246
Binucléé	binucle	372

TABLE 5.2 – Liste des classes annotées sur les images de la base Mitocheck.

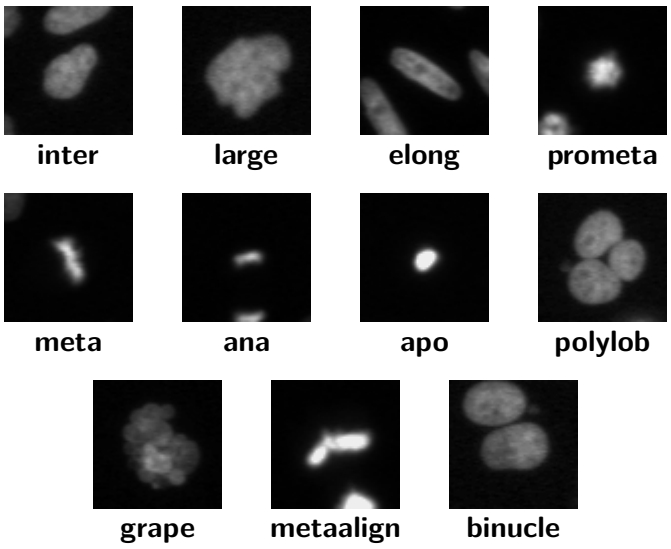


FIGURE 5.2 – Exemple de vignettes de la base Mitocheck contenant une cellule pour différentes classes associées.

Comme pour la base précédente, nous équilibrons et supprimons les cellules redondantes. Nous obtenons finalement une base comptant 1 100 vignettes, soit 100 images par classe.

5.3 Méthodes d'extraction des caractéristiques des images

Nous suivons une méthode d'analyse d'image classique constituée de trois étapes :
— segmentation des images ;
— extraction de caractéristiques des vignettes ;
— classification des vignettes en utilisant leurs caractéristiques.

Les images de nos deux bases de données étant déjà segmentées, nous présentons dans cette section l'outil d'extraction de caractéristiques que nous utilisons.

Une fois les vignettes extraites des images, nous calculons leurs caractéristiques qui seront ensuite utilisées pour les classifier. Un de nos objectifs étant de concevoir une méthode générique pouvant être adaptée à un grand nombre d'applications, nous avons choisi d'utiliser un outil permettant l'extraction d'un grand nombre de caractéristiques. Nous avons sélectionné le programme WND-CHARM. Il s'agit d'un outil développé en C++ par ORLOV et al., extrayant différents types de caractéristiques sur des images (ORLOV et al. 2008). Onze types de caractéristiques sont calculées sur les vignettes brutes, ainsi que sur différentes transformées de ces vignettes comme listé dans la table 5.3. Un type de caractéristiques calculé sur une image, ou une transformée d'image, forme ce que nous appelons par la suite un "groupe de caractéristiques". L'algorithme calcule 37

groupes de caractéristiques, générant à la fin un vecteur de 1 025 caractéristiques par vignette.

Types de caractéristiques	**Nombre de caractéristiques**	**Transformées d'images**
Statistiques sur la transformée de Chebyshev-Fourier	32	Fourier
Statistiques sur la transformée de Chebyshev	32	
Polynôme de Zernike	72	
Statistiques sur la transformée de Radon	12	Fourier, Chebyshev, Fourier & Chebyshev
Statistiques sur les bordures	28	Aucune
Statistiques sur les objets	34	
Textures de Gabor	7	
Quatre premiers moments	48	Fourier, Chebyshev, Ondelettes, Fourier & Chebyshev, Fourier & Ondelettes
Histogrammes multi-échelles	24	
Textures de Haralick	28	
Textures de Tamura	6	

TABLE 5.3 – Liste des caractéristiques calculées sur les vignettes et leurs transformées (ORLOV et al. 2008).

5.4 Détermination du temps d'exécution de l'analyse d'images

Notre objectif est d'analyser les images provenant de la caméra du microscope à la volée afin d'envoyer des informations de pilotage au module de contrôle en fonction des résultats d'analyse (voir chapitre 4). C'est pourquoi, nous devons respecter des contraintes temps-réel. Nous utilisons ici le même système embarqué (NVIDIA Jetson AGX Xavier, voir section 3.1.1) que pour l'exécution des fonctions d'asservissement.

Pour développer notre algorithme de classification et pour évaluer ses performances, nous utilisons la base d'images de démonstration de CellCognition présentée dans la section 5.2.1, comptant des vignettes de cellules en interphase ou en mitose annotées avec 8 classes.

Une fois la base d'images choisie, nous avons jugé important de soigneusement choisir l'algorithme d'analyse de ces images. En effet, nous avons pris en compte trois paramètres :

— la qualité de classification (précision) ;

— le temps d'exécution ;
— la généricité de la solution.

C'est pourquoi, pour répondre au problème de généricité, nous avons sélectionné l'extracteur de caractéristiques WND-CHARM pour sa capacité à calculer un grand nombre de caractéristiques différentes (voir section 5.3). Elles sont extraites sur chaque vignette (ne contenant qu'une seule cellule).

Une manière classique de classifier des objets ou images en biologie consiste à seuiller une seule caractéristique pour en déduire l'appartenance de l'objet à une classe. Cela serait la manière la plus rapide de classifier les cellules en termes de temps d'exécution, par rapport à la combinaison de plusieurs caractéristiques ou à l'utilisation d'extracteurs générés par apprentissage profond. Parmi les caractéristiques de WND-CHARM, nous avons sélectionné la plus discriminante, qui s'avère appartenir à la catégorie "Statistiques sur les objets" (voir table 5.3). Plus précisément, elle représente l'aire de l'objet (ici la cellule) segmenté par un seuil de Otsu (Otsu 1979). Nous présentons dans la figure 5.3 la répartition des valeurs de cette caractéristique en fonction des différentes classes, afin de voir s'il est possible de distinguer ces dernières par seuillage. Du fait de la taille importante du noyau en interphase et prophase, il est facile de distinguer ces deux classes des autres phases du cycle cellulaire (présentant un noyau plus petit) en observant cette caractéristique. En revanche, notre étude étant plus fine, le seuillage d'une seule caractéristique n'est pas suffisant.

En parallèle, nous avons mesuré le temps d'exécution de chaque type de caractéristiques présentées dans la table 5.3 en les calculant séquentiellement sur le CPU du système embarqué sans programmation multi-tâche. Nous mesurons le temps d'exécution total de l'extraction des caractéristiques entre 1 s et 2,6 s, avec une médiane à 1,7 s. Nous détaillons ce temps d'exécution dans la section 5.5.1.

Par conséquent, nous supposons à ce stade qu'il est possible de trouver un juste milieu entre l'utilisation d'une seule et tout l'ensemble des caractéristiques, en sélectionnant un sous-ensemble suffisant pour classifier les différentes phases du cycle cellulaire. Nous avons donc opté pour une approche d'apprentissage automatique classique plutôt qu'une méthode d'apprentissage profond. En effet, comme présenté dans la section 2.2, les approches usuelles de classification d'images, du fait de leur décomposition en plusieurs fonctions, sont plus faciles à optimiser via des méthodes de réduction de dimension que les approches d'apprentissage profond. Au-delà de notre exemple de détection de la transition métaphase-anaphase sur des cellules de type Hela Kyoto, nous souhaitons que notre méthode puisse être adaptée à un grand nombre d'applications et de types d'images. C'est pourquoi il semble intéressant de partir *a priori* d'un grand nombre de caractéristiques pour répondre à notre problème de généricité. Ainsi, après avoir sélectionné un sous-ensemble de caractéristiques adaptées à l'application ciblée, nous pouvons utiliser un algorithme d'apprentissage pour réaliser une classification sur ce sous-ensemble.

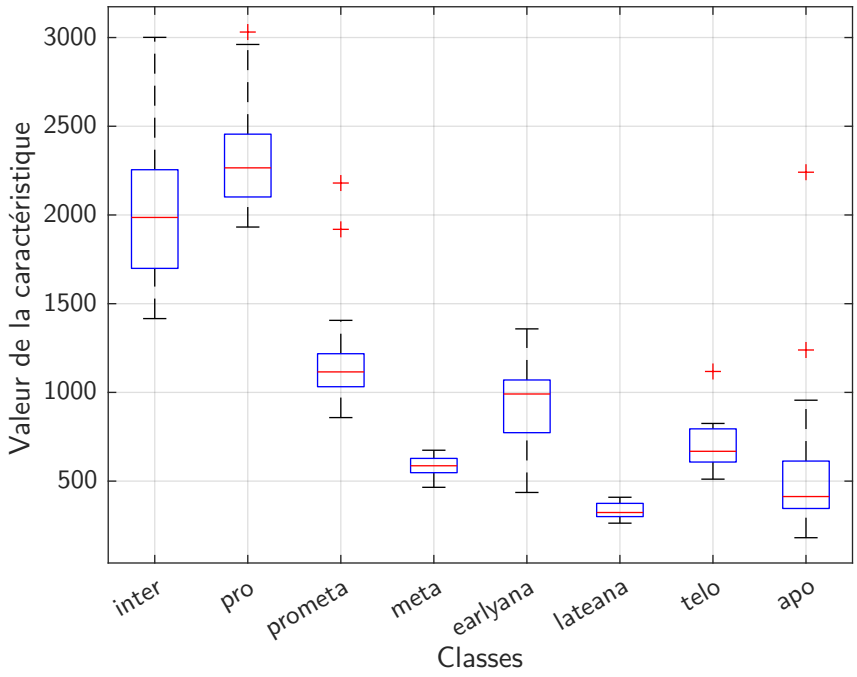


FIGURE 5.3 – Répartition des valeurs de la meilleure caractéristique en fonction de la classe de la vignette, à savoir l'aire de la cellule segmentée par un seuil de Otsu (OTSU 1979).

5.5 Classification des cellules

Dans cette section, nous explorons plusieurs solutions d'apprentissage automatique et de sélection de caractéristiques à l'aide du logiciel Matlab :

— un discriminant linéaire ;
— *Random Forest* ;
— un réseau de neurones.

5.5.1 Classification des images avec un discriminant linéaire

5.5.1.1 Calcul de la discriminance des groupes de caractéristiques

Pour des questions de rapidité, nous ne souhaitons calculer que les caractéristiques pertinentes (ou informatives) pour la classification afin d'optimiser le temps de calcul et ainsi respecter les contraintes temps-réel. En fonction des objets biologiques ou de la méthode d'acquisition des images, le sous-ensemble de caractéristiques informatives peut être différent d'une application à l'autre. Nous avons donc choisi un algorithme de réduction de dimension, en nous penchant dans un premier temps sur une méthode linéaire. Il existe plusieurs approches de réduction de dimension comme l'ACP ou l'ANOVA

par exemple. Dans notre cas, l'utilisation d'une ACP n'est pas appropriée car nécessite de calculer l'ensemble des caractéristiques avant de réduire leur dimension, ce qui ne nous permet pas d'optimiser le temps d'exécution. C'est pourquoi nous avons choisi le discriminant de Fisher (Fisher 1936), qui est aussi utilisé dans le processus de sélection des caractéristiques de WND-CHARM (Orlov et al. 2008). Ce discriminant, s'apparentant à une ANOVA, génère un score pour chaque caractéristique qui est le ratio entre les variances inter-classes et intra-classes, comme présenté dans la section 3.5.1.1 (Fisher 1936 ; Larson Martin G. 2008). Nous calculons donc ce score de la même manière que WND-CHARM à l'aide de l'équation 5.1 (Orlov et al. 2008) où :

- W_f représente le score de Fisher ;
- N représente le nombre de classes dans le jeu de données ;
- $\overline{T_f}$ représente la moyenne de la caractéristique f sur l'ensemble du jeu de données ;
- $\overline{T_{f,c}}$ représente la moyenne de la caractéristique f pour les échantillons appartenant à la classe c ;
- $\sigma^2_{f,c}$ représente la variance de la caractéristique f pour les échantillons appartenant à la classe c.

Dans le cas où des caractéristiques ont une variance égale à 0, l'équation 5.1 ne peut pas être calculée. Nous avons mis les scores de ces caractéristiques à 0. En effet, une variance nulle signifie que la caractéristique n'a aucune sensibilité à la variété de nos images.

$$W_f = \frac{\sum_{c=1}^{N} (\overline{T_f} - \overline{T_{f,c}})^2}{\sum_{c=1}^{N} \sigma^2_{f,c}} \cdot \frac{N}{N-1} \tag{5.1}$$

Notre objectif étant l'optimisation du temps de calcul des caractéristiques, nous souhaitions dans un même temps calculer leur score de Fisher et mesurer le temps d'exécution de chacune d'entre elles. Lors de cette dernière mesure, nous avons constaté que les caractéristiques n'étaient pas calculées indépendamment. Au contraire, elles sont calculées en groupes. En effet, pour la majorité des types de caractéristique de WND-CHARM (listés dans la table 5.3), un tronc commun de calcul est effectué, sur lequel des statistiques sont calculées. Les caractéristiques générées sont donc le résultat de ces statistiques. Un exemple fréquemment utilisé en traitement pour la classification d'images est le calcul des textures de Haralick (Haralick, Shanmugam et Dinstein 1973), consistant en différentes statistiques calculées sur la matrice de cooccurrence de l'image. Nous avons mesuré pour notre application que le temps de calcul de cette matrice représentait entre 90% et 99% du temps total d'exécution du groupe de caractéristique "textures de Haralick". Par conséquent, le temps de calcul des statistiques étant négligeable devant celui du tronc commun de calcul, il semblait pertinent de considérer les caractéristiques par groupes et non pas individuellement.

Sachant cela, nous avons aussi considéré le temps d'exécution pour chaque groupe de caractéristiques. Nous avons conservé l'architecture du logiciel WND-CHARM pour l'extraction des caractéristiques, consistant en l'exécution d'une même fonction pour le calcul de tout un groupe de caractéristiques. Nous avons donc mesuré le temps d'exécution de chaque groupe en mesurant simplement le temps écoulé entre le début et la fin de l'exécution de chaque fonction.

De même, nous avons calculé les scores de Fisher pour chaque groupe de caractéristiques. Ces scores étant mesurés pour chaque caractéristique individuellement, nous avons calculé des scores de groupes en faisant la moyenne des scores individuels pour chaque groupe. Effectivement, cela nous permet d'éviter de donner un score trop important aux groupes ne contenant que peu de caractéristiques discriminantes.

Afin de prendre en compte à la fois le temps d'exécution et les scores de Fisher des groupes caractéristiques, nous comparons ces deux paramètres dans la figure 5.4. Nous faisons apparaître en rouge sur le deuxième panneau de cette figure les groupes de caractéristiques dont le temps de calcul est supérieur à 20 ms. À ce stade, nous considérons ces groupes comme étant "chronophages".

Nous avons constaté que les deux groupes les plus discriminants étaient aussi deux groupes chronophages :

— textures de Gabor calculées sur les vignettes brutes ;
— textures de Haralick calculées sur la transformée en ondelettes des vignettes.

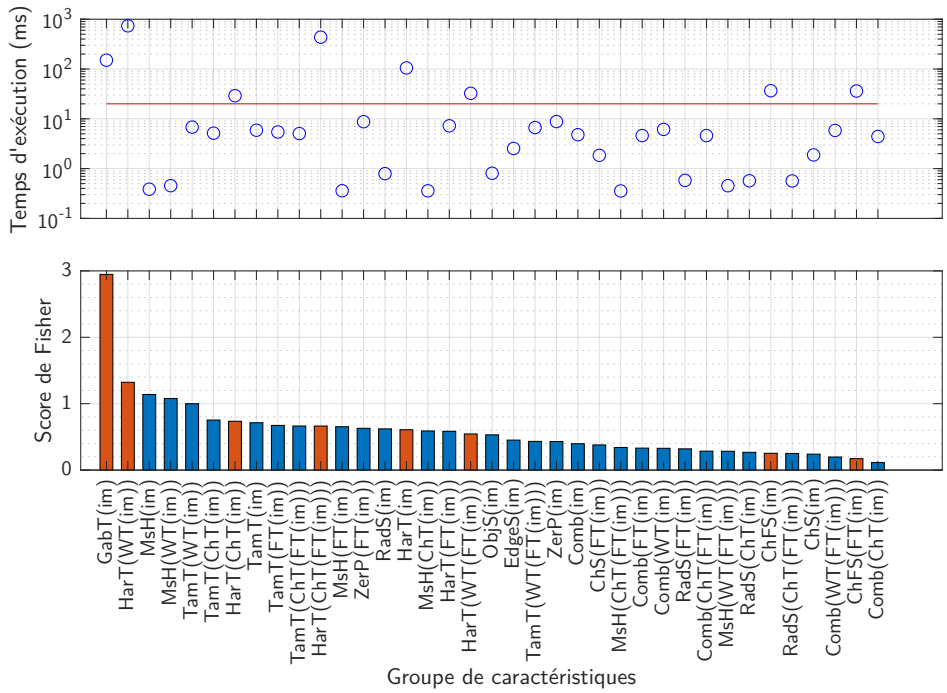


FIGURE 5.4 – Temps d'exécution (en haut) et scores de Fisher des groupes de caractéristiques (en bas) calculés avec la base d'images de CellCognition. Le score de Fisher des groupes de caractéristiques dont le temps d'exécution est supérieur à 20 ms est affiché en orange.

C'est pourquoi nous avons considéré à la fois le temps d'exécution et le score de Fisher des groupes de caractéristiques pour optimiser leur temps de calcul.

5.5.1.2 Classification

Ensuite, nous avons choisi un algorithme pour classifier les vignettes en utilisant leurs caractéristiques. Nous avons choisi d'utiliser un discriminant linéaire afin de rester dans un contexte de méthode linéaire. Cet algorithme, que nous présentons dans la section 2.2.1.3, consiste en la séparation des différentes classes par des hyperplans d'équation $y(\mathbf{x}) = 0$ où $y(\mathbf{x})$ est présentée équation 5.2 (Bishop 2006, pp. 181-184) où :

- $\mathbf{x}$ représente le vecteur de caractéristiques ;
- $\mathbf{w}$ représente un vecteur de poids calculés lors de l'entraînement du discriminant linéaire sur le jeu d'apprentissage ;
- w_0 représente un biais calculé lors de l'entraînement, dont l'opposée est aussi appelée seuil (Bishop 2006, pp. 181-184).

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 \tag{5.2}$$

Afin d'évaluer la qualité de classification des vignettes, nous avons suivi un processus de validation croisée de type k-fold [1] en utilisant deux métriques de qualification :

- la précision globale de classification représentant le ratio entre le nombre d'images correctement classées sur le nombre total d'images ;
- la moyenne des aires sous les courbes Receiver Operating Characteristic (ROC) [2], qui est un outil fréquemment utilisé en apprentissage automatique (Fawcett 2006).

Ainsi, les résultats présentés dans cette section ont été obtenus sur l'ensemble de notre jeu de données.

5.5.1.3 Réduction du nombre de groupes de caractéristiques

Nous ne pouvons pas utiliser le discriminant linéaire pour effectuer la classification sur l'ensemble des caractéristiques. En effet, il ne permet pas de traiter des problèmes sous-échantillonnés, ce qui est notre cas ici (159 échantillon avec 1025 variables). C'est pourquoi nous avons procédé à la réduction du nombre de groupes de caractéristiques afin de n'étudier que des cas sur-échantillonnés.

Notre objectif est de trouver le nombre minimal de groupes de caractéristiques permettant d'optimiser le temps d'exécution de leur calcul, sans dégrader de façon significative la qualité de classification. Pour cela, nous avons suivi une méthode itérative consistant à progressivement supprimer des groupes de caractéristiques en commençant par les moins discriminants. Ce processus est décrit figure 5.5.

Dans un premier temps nous avons suivi ce processus sans considérer le temps de calcul des groupes de caractéristiques en ne considérant que leurs scores de Fisher. Nous listons les résultats obtenus dans la table 5.4a. Nous avons obtenu la meilleure classification lors

1. Les échantillons du jeu de données sont divisés en k sous-ensembles de taille égale. Un des sous ensemble est utilisé pour le test et les k-1 autres pour l'entraînement. Ce processus est répété k fois de telle sorte que chaque sous-ensemble n'apparaisse qu'une fois dans le test.

2. Les courbes ROC sont un outil de qualification de classificateur représentant le taux de vrais positifs en fonction du taux de faux positifs lorsque l'on fait varier le seuil de sélection d'une classe.

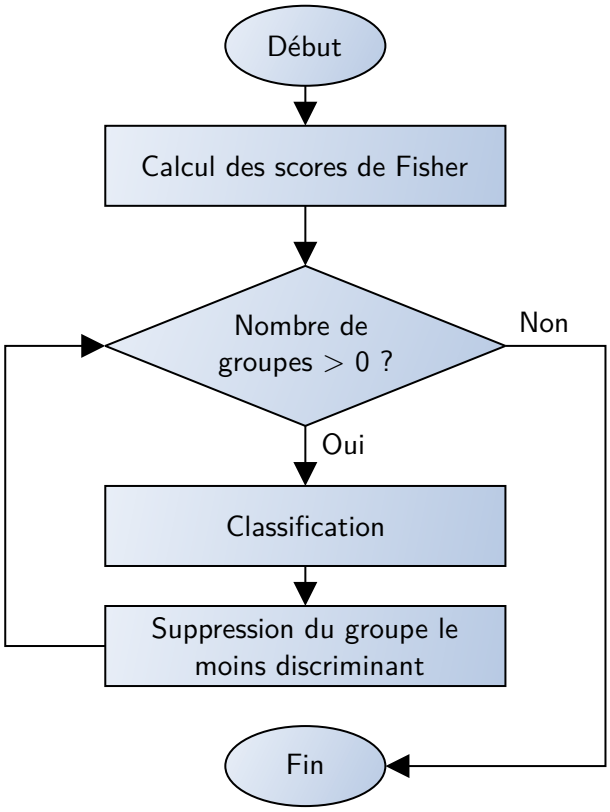


FIGURE 5.5 – Processus de suppression progressive des groupes de caractéristiques en utilisant leur score de Fisher.

de l'utilisation des 2 groupes de caractéristiques les plus discriminants (voir figure 5.4), donnant une précision globale de 78,0 % et une Area Under the ROC Curve (AUC, Aire Sous la Courbe ROC) moyenne de 0,954. La matrice de confusion (figure 5.6a) et les courbes ROC (figure 5.6b) obtenues avec ce sous ensemble montrent une classification à peu près similaire pour chaque classe, malgré quelques erreurs dans la différenciation des classes pro-métaphase et fin d'anaphase. Cela peut être expliqué par la forte ressemblance de certaines images dans ces deux classes. En revanche nous avons constaté un temps d'exécution légèrement inférieur à 1 s, ce qui n'est pas une optimisation suffisante pour une application temps-réel. Ne considérer que les scores de Fisher n'est donc pas suffisant pour réduire le temps d'exécution de l'extraction des caractéristiques.

Nous avons donc dans un second temps pris en compte deux paramètres pour optimiser le temps de calcul des groupes de caractéristiques : leur score de Fisher et leur temps d'exécution. Pour cela nous avons supprimé tous les groupes chronophages (représentés en rouge dans la figure 5.4) avant de progressivement supprimer les groupes les moins discriminants (même processus que celui présenté figure 5.5). Ainsi nous avons obtenu

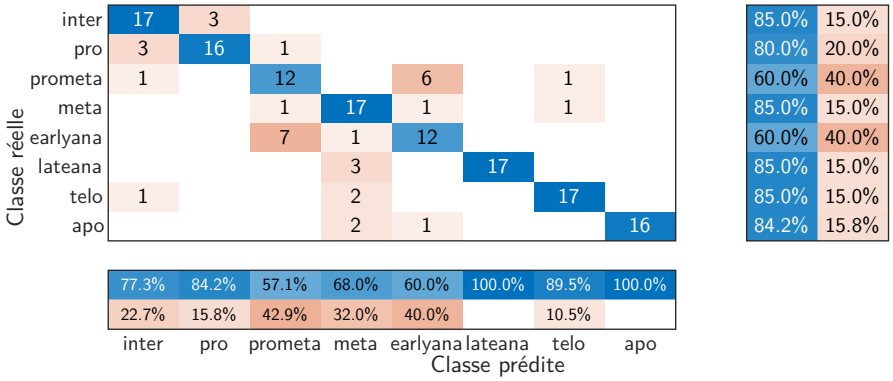
Nb. gp.	**Nb. carac.**	**Préc. globale**	**AUC moy.**	**Temps d'exéc.** (ms)
1	7	56,6 %	0,882	150,32
2	**35**	**78,0 %**	**0,954**	**889,32**
3	58	68,6 %	0,919	889,71
4	81	64,2 %	0,903	890,16
5	86	61,0 %	0,894	896,97
6	114	50,9 %	0,794	935,70
7	142	35,2 %	0,630	1 381,24

(a) Résultats avec groupes chronophages

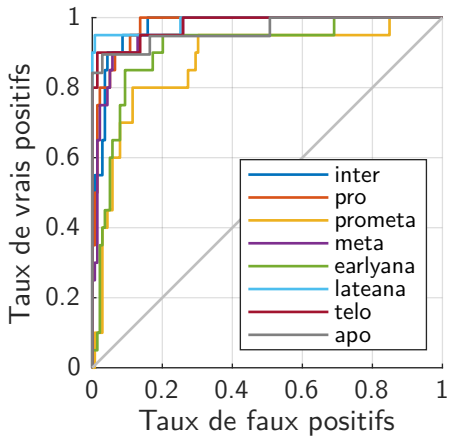
Nb. gp.	**Nb. carac.**	**Préc. globale**	**AUC moy.**	**Temps d'exéc.** (ms)
1	23	42,8 %	0,790	0,39
2	46	45,9 %	0,804	2,36
3	**51**	**52,2 %**	**0,842**	**9,17**
4	123	41,5 %	0,710	19,66
5	128	17,6 %	0,538	34,46
6	140	28,9 %	0,594	35,25
7	145	31,4 %	0,608	41,15

(b) Résultats sans groupe chronophage

TABLE 5.4 – Résultats de classification en utilisant un discriminant linéaire avec (a) et sans (b) groupes de caractéristiques chronophages. Chaque colonne représente le nombre de groupes utilisés, le nombre de caractéristiques incluses dans l'ensemble des groupes, la précision globale, l'AUC moyenne obtenues et le temps d'exécution de l'extraction des caractéristiques.



(a)



(b)

FIGURE 5.6 – Matrice de confusion (a) et courbes ROC (b) obtenues avec un discriminant linéaire sur les 2 caractéristiques les plus discriminantes de la base d'images de CellCognition. Les tableaux sous et à droite la matrice de confusion (a) représentent respectivement la précision et la sensibilité.

les résultats de la table 5.4b, montrant que la meilleure classification est obtenue avec 3 groupes de caractéristiques, pour un temps d'exécution de 9,17 ms :

— histogrammes multi-échelles calculés sur les vignettes brutes ;
— histogrammes multi-échelles calculés sur la transformée en ondelettes des vignettes ;
— textures de Tamura sur la transformée en ondelettes des vignettes.

Cela représente une très bonne optimisation du temps de calcul des caractéristiques, permettant par exemple de traiter une image contenant une centaine de cellules en moins d'une seconde.

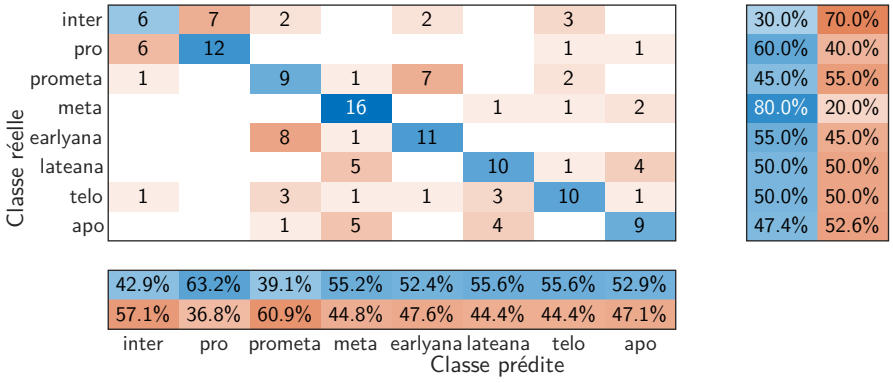
Nous avons en revanche constaté une forte dégradation de la qualité de la classification. En effet, la table 5.4b nous montre que le meilleur résultat obtenu avec ce sous-ensemble tronqué est largement inférieur à la plupart des résultats obtenus en utilisant les groupes de caractéristiques chronophages (table 5.4a). Comme indiqué dans les figures 5.7a et 5.7b, cette dégradation de la qualité de la classification s'applique à l'ensemble des classes.

5.5.1.4 Résultats avec la base Mitocheck

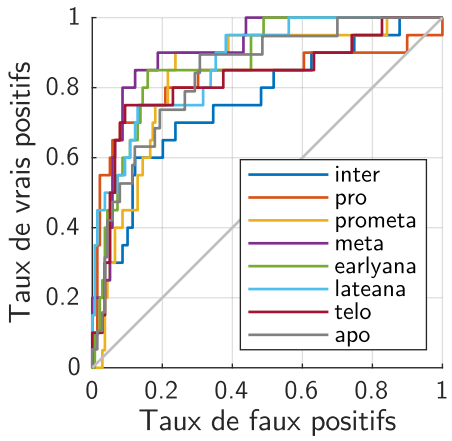
À titre de comparaison, nous avons suivi le même processus avec notre seconde base de données Mitocheck. Du fait que cette base contienne plus d'images, nous ne sommes pas dans un cas de sous-échantillonnage, même si toutes les caractéristiques sont utilisées (1100 échantillons et 1025 caractéristiques). Nous avons donc progressivement supprimé des groupes de caractéristiques en commençant par le jeu de donné complet. Comme avec la base de CellCognition, nous avons suivi ce processus avec et sans les groupes de caractéristiques chronophages. La figure 5.8 nous a montré que dans ce cas-ci, l'AUC moyenne est aussi impactée par la suppression des groupes chronophages et discriminants. Cela confirme que les résultats précédents ne sont pas un cas particulier.

5.5.1.5 Conclusion sur le discriminant linéaire

Finalement, cette méthode linéaire nous a permis de considérablement réduire le temps de calcul des caractéristiques des vignettes, au prix d'une forte dégradation de la qualité de la classification. Cela nous a donc laissé un compromis difficile entre la vitesse et la précision. En revanche, nous avons pu constater une certaine redondance entre les différents groupes de caractéristiques. Effectivement, bien que la suppression de groupes discriminants ait dégradé la qualité de la classification, le discriminant linéaire est capable de classifier correctement la moitié des vignettes, nous laissant penser que plusieurs groupes de caractéristiques pouvaient porter une même information. Nous nous sommes donc tourné vers une méthode non-linéaire dans le but d'améliorer le compromis vitesse/précision.



(a)



(b)

FIGURE 5.7 – Matrice de confusion (a) et courbes ROC (b) obtenues avec un discriminant linéaire sur les 3 caractéristiques les plus discriminantes de la base d'images de CellCognition après suppression des plus chronophages. Les tableaux sous et à droite la matrice de confusion (a) représentent respectivement la précision et la sensibilité.

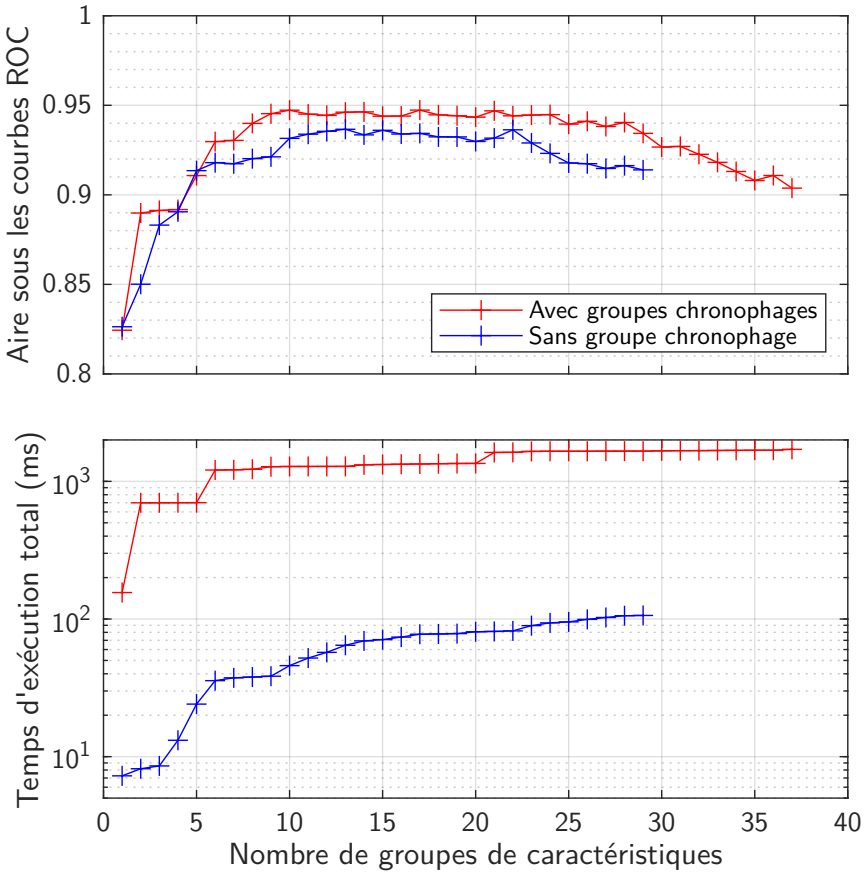


FIGURE 5.8 – Évolution de la moyenne des AUC avec un discriminant linéaire (en haut) et du temps d'exécution de l'extraction des caractéristiques (en bas) en fonction du nombre de groupes de caractéristiques sur la base d'images Mitocheck.

5.5.2 Classification des images avec *Random Forest*

5.5.2.1 Classification et calcul de l'importance des groupes de caractéristiques

La méthode linéaire présentée dans la section précédente (5.5.1) ne nous a pas donné de résultats convaincants. C'est pourquoi nous avons cherché une méthode non-linéaire permettant à la fois de classifier et de sélectionner un sous-ensemble de caractéristiques pertinent afin d'en optimiser le temps de calcul. Nous avons donc choisi d'utiliser l'algorithme *random forest*. Il s'agit d'un outil de classification basé sur un ensemble d'arbre de décision, chacun étant entraîné avec différents sous-ensembles de variables et d'échantillons du jeu d'apprentissage (Breiman 2001). Nous détaillons le fonctionnement de *random forest* dans la section 2.2.1.3. De par sa structure, il présente les avantages suivants (Breiman 2001, p. 10) :

- les précisions obtenues dans l'état de l'art (Azar et El-Metwally 2013) sont aussi bonnes voir meilleures qu'avec des méthodes de boosting comme Adaboost (Freund et Schapire 1996 ; Schapire 2013) ;
- est robuste au bruit et aux valeurs aberrantes ;
- est plus rapide que les algorithmes de boosting ;
- peu d'hyper-paramètres sont nécessaires pour l'entraînement ;
- ne sur-apprend pas ;
- l'entraînement et l'exécution peuvent être utilisés en multi-tâche ;
- contient des outils internes permettant d'estimer l'erreur de classification, ainsi que la corrélation et l'importance des variables d'entrées.

Ce dernier point entre autres nous a fait pencher vers cette solution. En revanche, l'importance des variables d'entrée (ici, il s'agit des caractéristiques) ne peut être calculée qu'après l'entraînement de l'algorithme. Nous présentons le principe du calcul de l'importance des variables dans la classification avec *random forest* dans la section 3.5.1.2.

Parmi les différents paramètres d'apprentissage de *random forest*, nous avons dû déterminer le nombre d'arbres de décision constituant l'ensemble. Un des avantages de *random forest* est que même si un trop grand nombre d'arbres de décision est utilisé, il n'aura pas de problème de sur-apprentissage [3] dont peuvent souffrir d'autres algorithmes tel que les réseaux de neurones (Breiman 2001). Cela augmente en revanche le temps d'apprentissage de l'algorithme ainsi que son temps d'exécution. C'est pourquoi nous avons cherché le nombre optimal d'arbres de décision au-delà duquel la qualité de classification n'est plus améliorée, en mesurant l'erreur de classification des échantillons dits OOB [4] en fonction du nombre d'arbres de décision utilisés. Nous avons mesuré un nombre optimal aux alentours de 300 arbres de décision comme l'indique la figure 5.9.

Le calcul de l'importance des caractéristiques nécessite l'entraînement de *random forest*. Nous avons donc réalisé un premier entraînement en utilisant l'ensemble du jeu de

3. Le sur-apprentissage signifie qu'un modèle statistique apprend les détails du jeu d'apprentissage avec lequel il est entraîné, l'empêchant ainsi de généraliser la solution et d'obtenir une bonne classification sur un jeu de test.

4. Chaque arbre de décision de l'ensemble contient un certain nombre d'échantillons dits Out-Of-Bag, aléatoirement sélectionnés, qui ne sont pas utilisés pour l'entraînement de l'arbre.

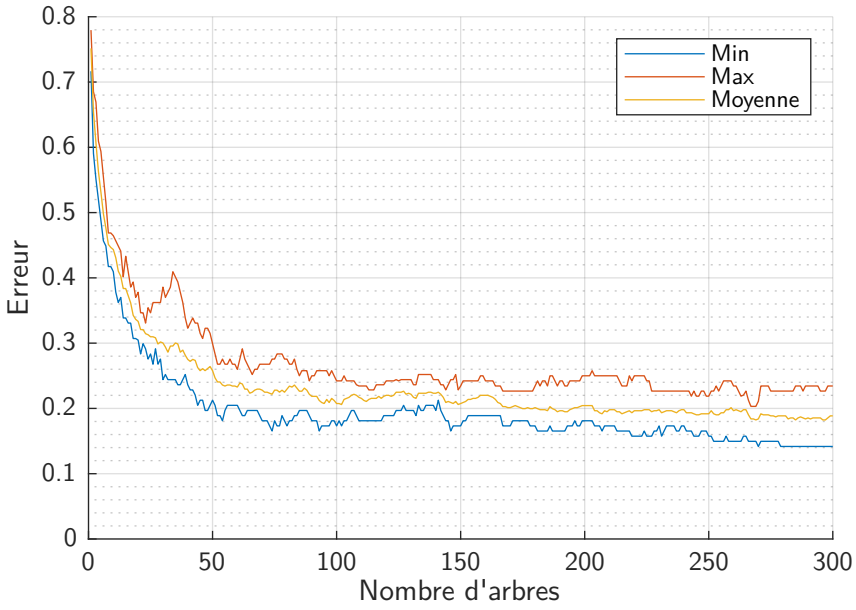
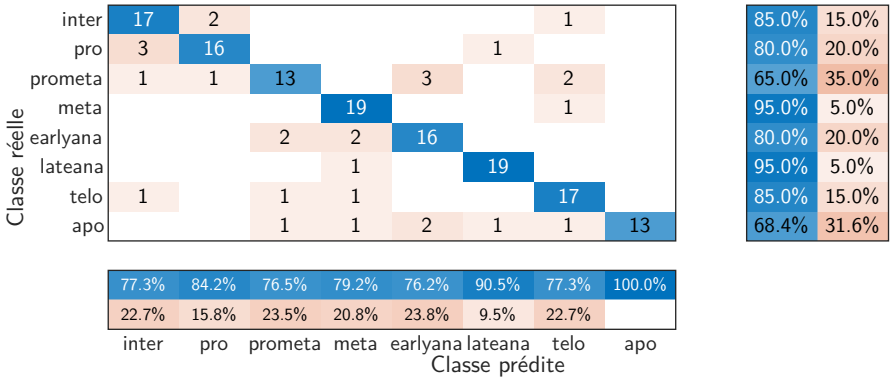


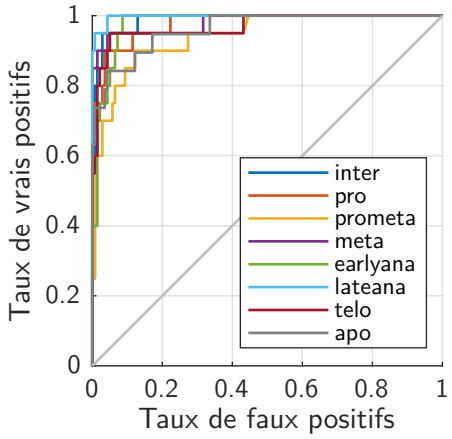
FIGURE 5.9 – Erreur de classification des échantillons OOB avec *random forest* en fonction du nombre d'arbres de décision utilisés. Les minimum, maximum et moyenne sont calculés sur les 5 itérations de validation croisée.

caractéristiques (37 groupes, 1 025 caractéristiques). Comme dans le cas du discriminant linéaire (voir section 5.5.1), nous effectuons une validation croisée de type k-fold avec $k = 5$. La matrice de confusion et les courbes ROC présentées figures 5.10a et 5.10b respectivement, nous ont montré une qualité de classification supérieure à celle obtenue avec le discriminant linéaire dans le meilleur des cas (voir section 5.5.1), avec une précision globale de 81,8 % et une AUC de 0,974. Cela nous a permis de valider que la classification se faisait correctement. Ainsi, *random forest* permettant de quantifier la contribution des caractéristiques dans l'obtention de ces résultats, il est pertinent d'exploiter cette quantification pour réduire le nombre de caractéristiques tout en gardant une qualité de classification similaire.

Nous avons donc pu mesurer la contribution (que l'on appelle aussi importance) de chaque caractéristique dans la classification. Cette importance est déterminée, pour chaque caractéristique, en mesurant la variation de l'erreur de classification des échantillons OOB lorsque que les valeurs de cette caractéristique sont mélangées selon les échantillons OOB. Cela est effectué pour chaque arbre de décision, et est moyenné sur l'ensemble des arbres. La validation croisée nous donnant 5 *random forest*s, ces importances sont moyennées sur ces 5 modèles. Enfin, comme avec la méthode linéaire (voir section 5.5.1), nous résumons ces importances par groupes de caractéristiques en calculant leur moyenne sur chaque groupe, nous donnant ainsi les importances présentées dans le deuxième panneau de la figure 5.11. Nous avons de nouveau observé la présence de groupes de caractéristiques chronophages, représentées en rouge, parmi les plus importantes dans la classification.



(a)



(b)

FIGURE 5.10 – Matrice de confusion (a) et courbes ROC (b) obtenues avec *random forest* sur toutes caractéristiques de la base d'images de CellCognition. Les tableaux sous et à droite la matrice de confusion (a) représentent respectivement la précision et la sensibilité.

Nous nous sommes donc de nouveau intéressé au temps d'exécution et à l'importance des groupes de caractéristiques pour réduire leur nombre.

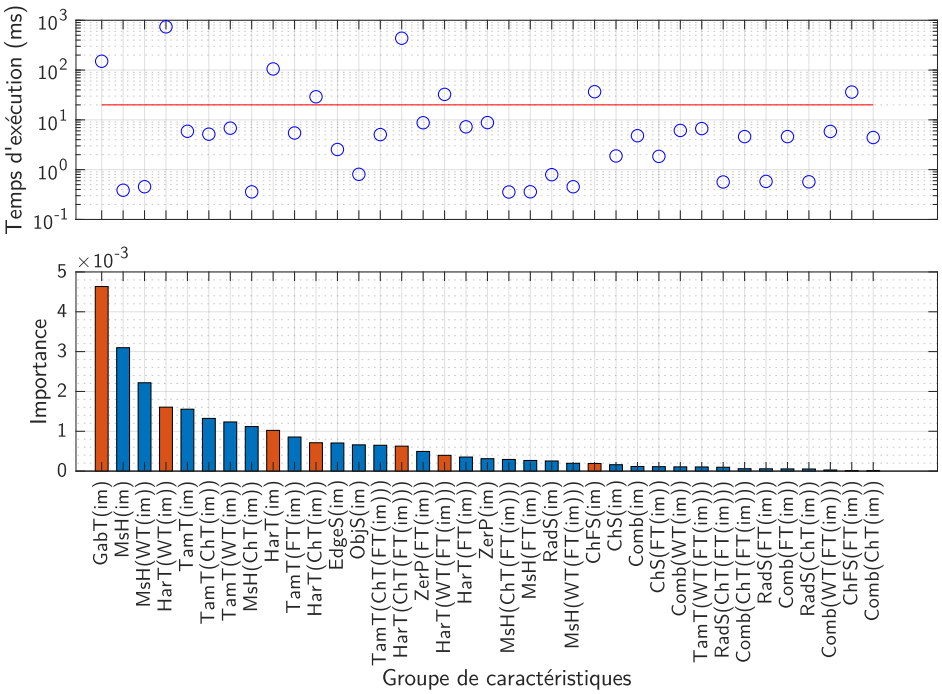


FIGURE 5.11 – Temps d'exécution (en haut) et importance des groupes de caractéristiques (en bas) calculés avec la base d'images de CellCognition. L'importance des groupes de caractéristiques dont le temps d'exécution est supérieur à 20 ms est affichée en orange.

5.5.2.2 Réduction du nombre de groupes de caractéristiques

Comme avec le discriminant linéaire, nous voulions réduire progressivement le nombre de groupes de caractéristiques en supprimant les moins importants pour la classification. Le mécanisme de calcul de l'importance étant différent des scores de Fisher, le processus de suppression des groupes les moins importants est légèrement différent que dans la section 5.5.1 (voir figure 5.12). En effet, contrairement au score de Fisher, il est nécessaire de recalculer l'importance des groupes de caractéristiques à chaque itération.

Nous avons donc dans un premier temps supprimé progressivement les groupes de caractéristiques les moins importants sans considérer leur temps d'exécution. Nous avons obtenu les courbes rouges de la figure 5.13, nous montrant d'une part que la qualité de classification commence à être sérieusement impactée lorsque moins de 8 groupes de caractéristiques sont utilisés. Avec 8 groupes, nous avons obtenu une précision globale de 75,5 % et une AUC moyenne de 0,970. D'autre part, nous avons noté que le temps

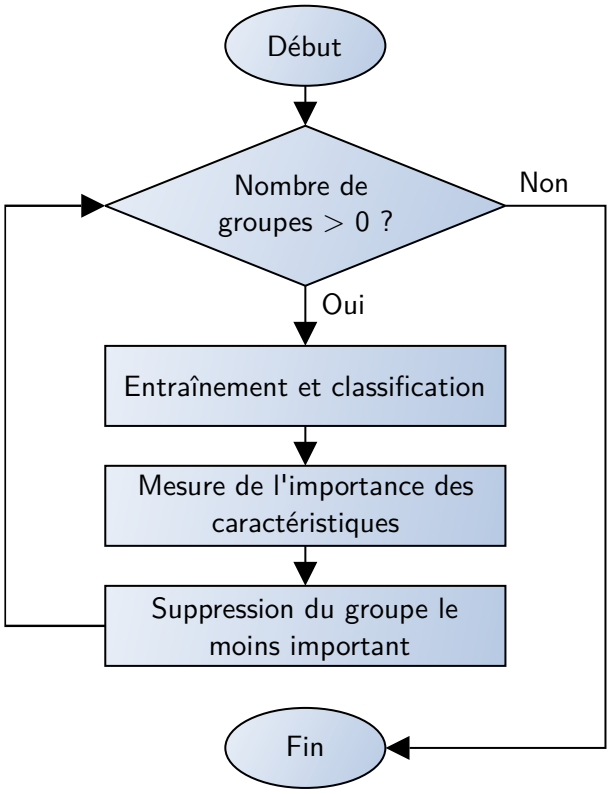


FIGURE 5.12 – Processus de suppression progressive des groupes de caractéristiques en utilisant leur importance dans la classification avec *random forest*.

d'exécution reste très important (plus de 900 ms avec 8 groupes) du fait de la présence de groupes chronophages dans le sous-ensemble.

Nous avons donc de nouveau reproduit ce processus en supprimant dans un premier temps les groupes de caractéristiques dont le temps de calcul est supérieur à 20 ms (groupes chronophages), avant de supprimer les moins importants. Nous avons obtenu les courbes bleues de la figure 5.13. Nous avons observé qu'en utilisant *random forest*, la qualité de classification est beaucoup moins dégradée qu'avec le discriminant linéaire lorsque les groupes de caractéristiques chronophages sont supprimés malgré leur importance dans la classification. Au contraire, les résultats avec et sans ces groupes sont très similaires.

Nous avons regardé plus en détail, et avons estimé que lorsque les groupes chronophages sont supprimés en amont, l'AUC se dégradait lorsque moins de 12 groupes de caractéristiques étaient utilisés. Ainsi nous avons calculé la matrice de confusion (voir figure 5.14a) et les courbes ROC (voir figure 5.14b) lorsque nous utilisions ces 12 groupes, avons constaté que la qualité de classification était même légèrement meilleure que lorsque tous les groupes de caractéristiques étaient utilisés (voir figure 5.10), avec une précision

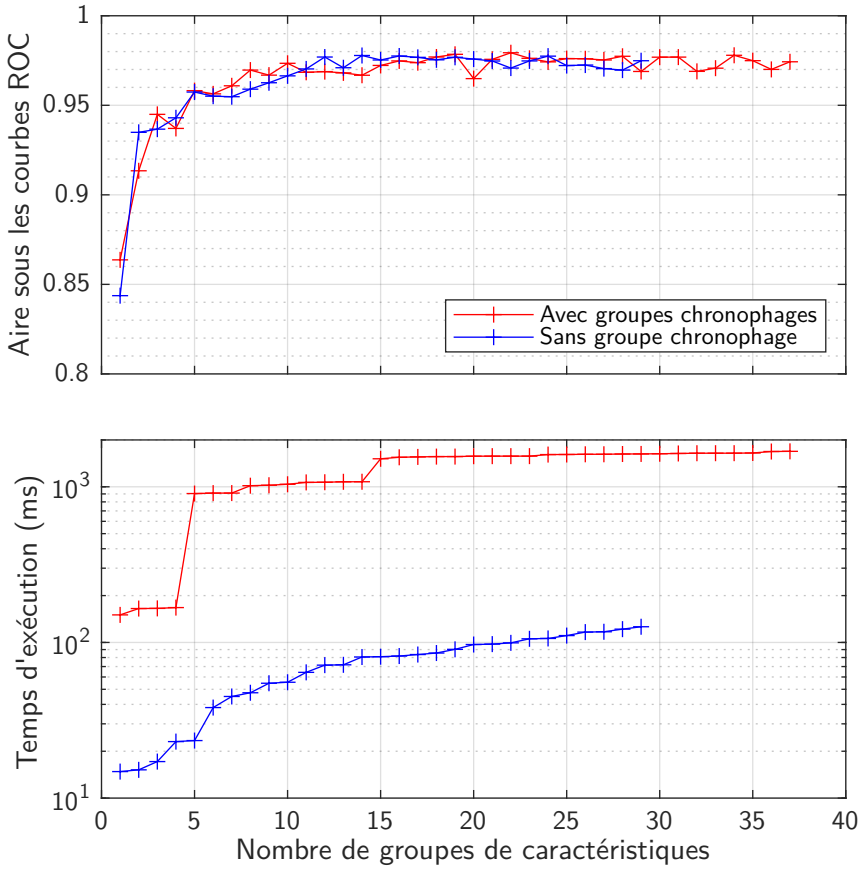
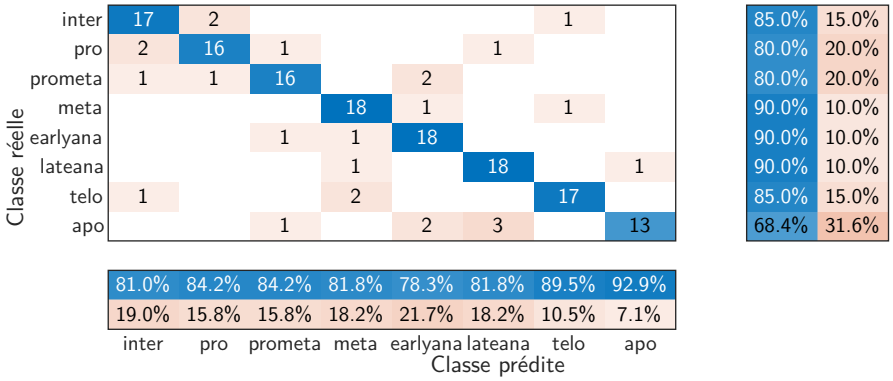
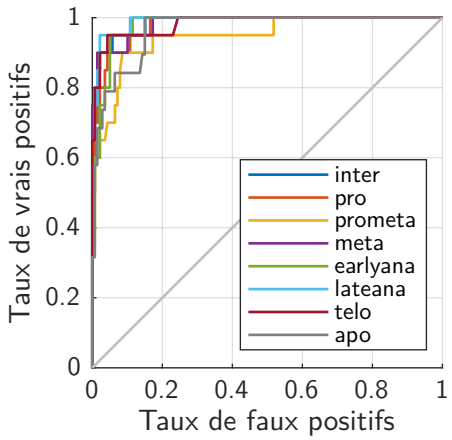


FIGURE 5.13 – Évolution de la moyenne des AUC avec *random forest* (en haut) et du temps d'exécution de l'extraction des caractéristiques (en bas) en fonction du nombre de groupes de caractéristiques sur la base d'images de CellCognition.

globale de 83,6 % et une AUC moyenne de 0,977. Avec un temps de calcul aux alentours de 72 ms, nous avons bien optimisé le temps d'exécution de l'extraction des caractéristiques sans dégrader la précision de la classification. Ainsi, il nous a été possible d'utiliser *random forest* pour classifier plus de 13 cellules par secondes.



(a)



(b)

FIGURE 5.14 – Matrice de confusion (a) et courbes ROC (b) obtenues avec *random forest* sur les 12 caractéristiques les plus discriminantes de la base d'images de CellCognition après suppression des plus chronophages. Les tableaux sous et à droite la matrice de confusion (a) représentent respectivement la précision et la sensibilité.

Afin d'expliquer cette capacité de *random forest* à conserver une bonne qualité de classification malgré la suppression de caractéristiques qui étaient considérées comme importantes pour la classification, nous avons étudié l'évolution de l'importance des groupes de caractéristiques. Nous avons observé dans la figure 5.15 que lorsque que des groupes de caractéristiques importants (comme certains des groupes chronophages) sont supprimés, l'importance des autres groupes augmente jusqu'à presque doubler lorsque nous

utilisons 12 groupes. Cela montre donc une redondance dans les informations que donnent certaines caractéristiques et la capacité de *random forest* à exploiter cette redondance (Tuv et al. 2009 ; Zhao, Anand et M. Wang 2019).

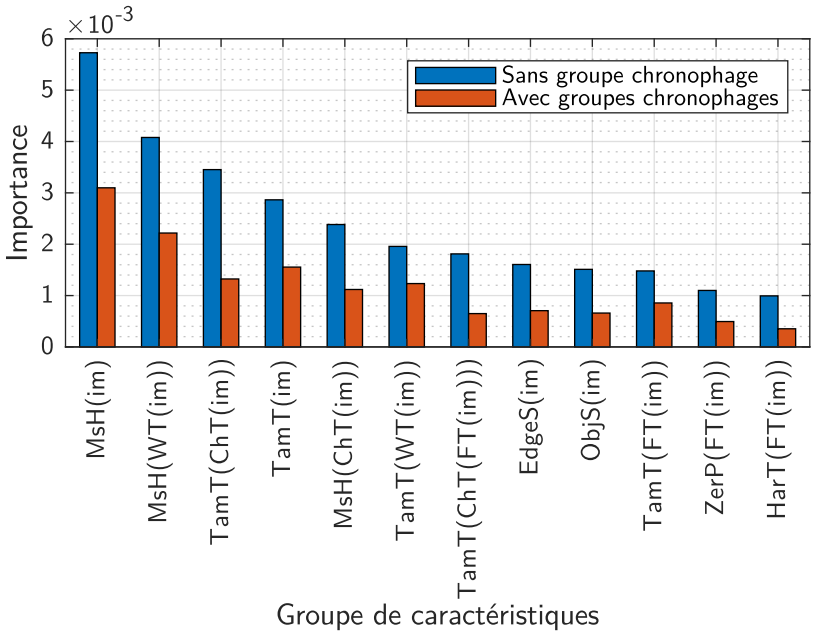


Figure 5.15 – Comparaison de l'importance des groupes de caractéristiques avant (orange) et après (bleu) réduction de leur nombre sur la base d'images de CellCognition. Les groupes affichés sont les 12 meilleurs obtenus après suppression des groupes chronophages.

5.5.2.3 Bootstrap

Du fait de notre utilisation d'une méthode de bootstrap pour sélectionner les cellules constituant notre base d'images, nous avons reproduit cette expérience plusieurs fois en sélectionnant aléatoirement d'autres (voir description de la base d'images dans la section 5.2.1). Ainsi, cela nous permet de valider qu'il ne s'agit pas d'un cas particulier de fonctionnement et que les résultats ne sont pas dépendants du choix des images. Ainsi, nous avons répété 10 itérations du bootstrap, en utilisant toujours une validation croisée de type k-fold avec $k = 5$. Nous avons dans un premier temps mesuré pour chaque itération du bootstrap l'évolution de l'AUC en fonction du nombre de groupes de caractéristiques utilisées pour la classification, en ayant supprimé en amont les groupes chronophages. Nous avons ensuite calculé la moyenne de cette évolution sur les 10 itérations comme présenté dans la figure 5.16. Nous avons constaté une très faible variabilité de la qualité de classification ainsi que du temps d'exécution comme l'indique cette figure. Aussi, nous avons estimé le nombre optimal de groupes à 11,6 en moyenne avec une déviation standard de 2,4 groupes.

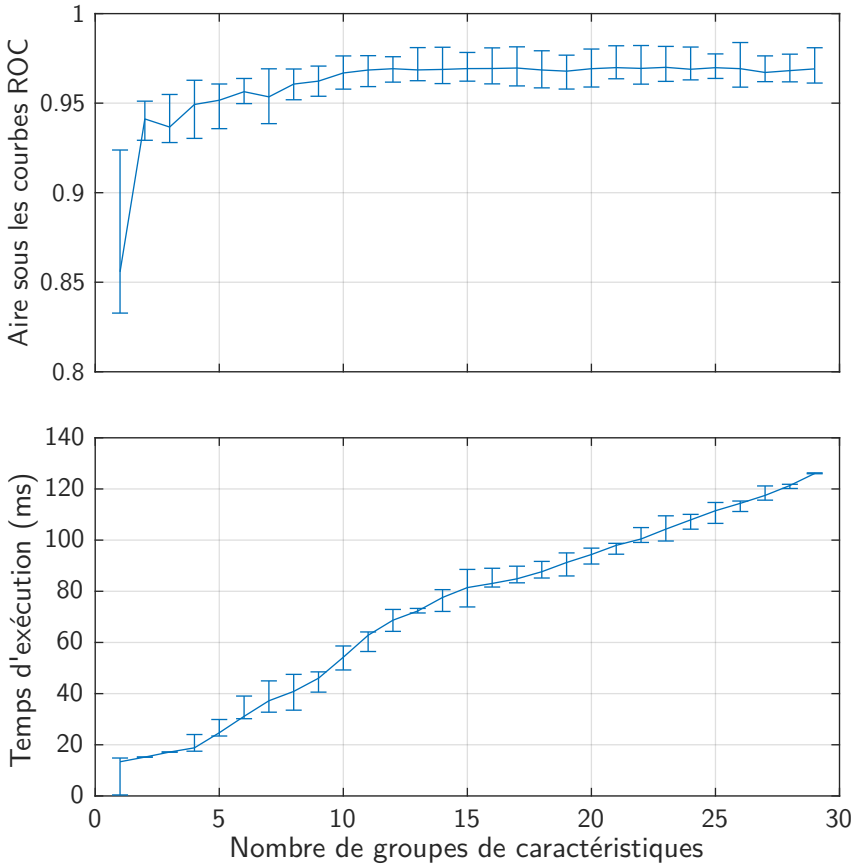


FIGURE 5.16 – Évolution de la moyenne des AUC avec *random forest* (en haut) et du temps d'exécution de l'extraction des caractéristiques (en bas) en fonction du nombre de groupes de caractéristiques sur la base d'images de CellCognition. Ces valeurs sont moyennées sur 10 itérations de bootstrap. Les barres d'erreurs montrent le maximum et le minimum dans chaque cas.

Ayant mesuré une variation du temps d'exécution à chaque itération du bootstrap, nous avons étudié plus en détail les groupes de caractéristiques sélectionnés lors de l'optimisation. En effet, nous avons noté que d'une itération à une autre, les 12 groupes les plus importants n'étaient pas toujours les mêmes. En revanche, comme le montre la table 5.5, cette variation est faible. Nous observons en effet 11 groupes présents dans toutes les itérations, pour un total de 15 groupes présent au moins dans une itération.

Groupes de carac.	**Nombre**
EdgeS(im)	10
MsH(ChT(im))	10
MsH(WT(im))	10
MsH(im)	10
ObjS(im)	10
TamT(ChT(FT(im)))	10
TamT(ChT(im))	10
TamT(FT(im))	10
TamT(WT(im))	10
TamT(im)	10
ZerP(FT(im))	10
HarT(FT(im))	5
MsH(ChT(FT(im)))	3
MsH(FT(im))	1
ZerP(im)	1

TABLE 5.5 – Groupes de caractéristiques présents dans les 10 bootstraps avec la base d'images de CellCognition lorsque 12 groupes sont utilisés. 11 groupes sur 12 sont présents dans les 10 itérations de bootstrap.

Ainsi, la faible variabilité du temps d'exécution nous montre donc la répétabilité de notre méthode avec différents lots l'images. De plus, nous pouvons déduire des faibles variations au niveau de l'importances des groupes caractéristiques que les meilleurs groupes ne sont pas sélectionnés en fonction du détail des images, mais sur des caractéristiques générale de ces dernières.

5.5.2.4 Résultats avec la base Mitocheck

Dans le but de confirmer la robustesse de notre méthode, nous avons testé notre processus d'optimisation avec *random forest* sur la base d'images Mitocheck. Afin de comparer les résultats obtenus avec la base de CellCognition et de Mitocheck, nous avons effectué une validation croisée de type k-fold avec $k = 5$, dont chaque *random forest* est constitué de 300 arbres de décision. Nous avons suivi le même processus avec et sans les groupes de caractéristiques chronophages et obtenu les courbes présentées dans la figures 5.17. Nous avons observé de nouveau que la suppression des groupes de caractéristiques chronophages n'impactait pas la qualité de classification.

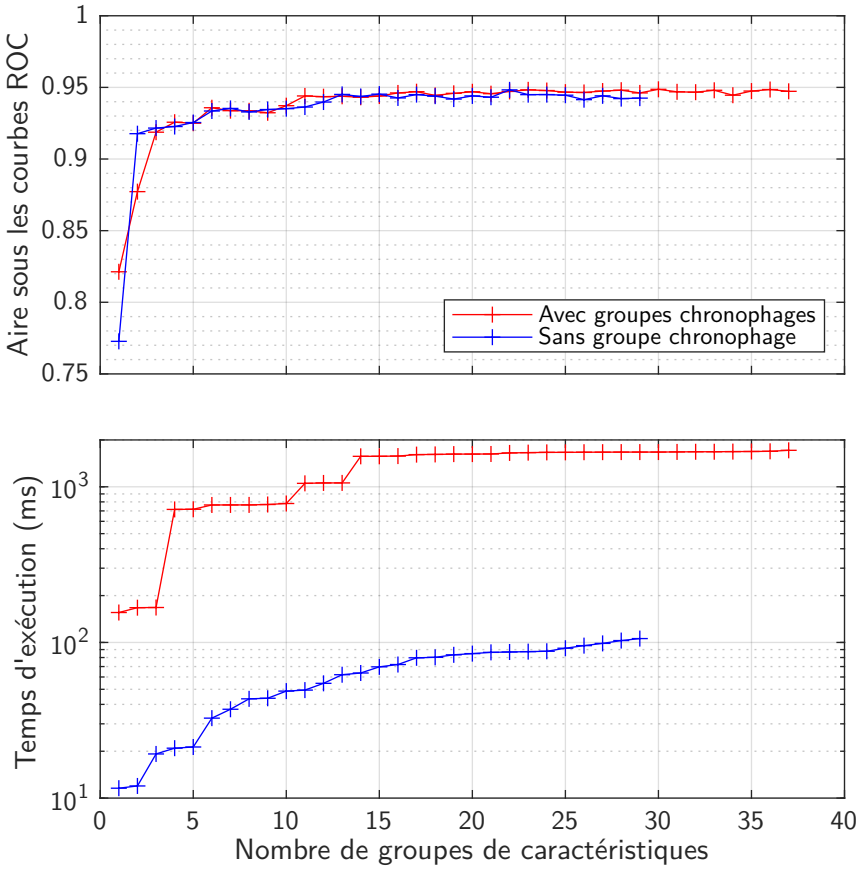


FIGURE 5.17 – Évolution de la moyenne des AUC avec *random forest* (en haut) et du temps d'exécution de l'extraction des caractéristiques (en bas) en fonction du nombre de groupes de caractéristiques sur la base d'images de Mitocheck.

Nous avons ensuite réalisé 10 itérations de bootstrap avec des images aléatoirement sélectionnées comme décrit dans la section 5.2.2. Nous présentons figure 5.18 l'évolution de l'AUC uniquement lorsque les groupes de caractéristiques chronophages sont supprimés en amont. Ainsi, nous avons montré la reproductibilité des résultats à la fois en termes de qualité de classification avec une très faible variabilité de la précision, ainsi qu'en termes de temps d'exécution et de groupes de caractéristiques sélectionnés (voir table 5.6).

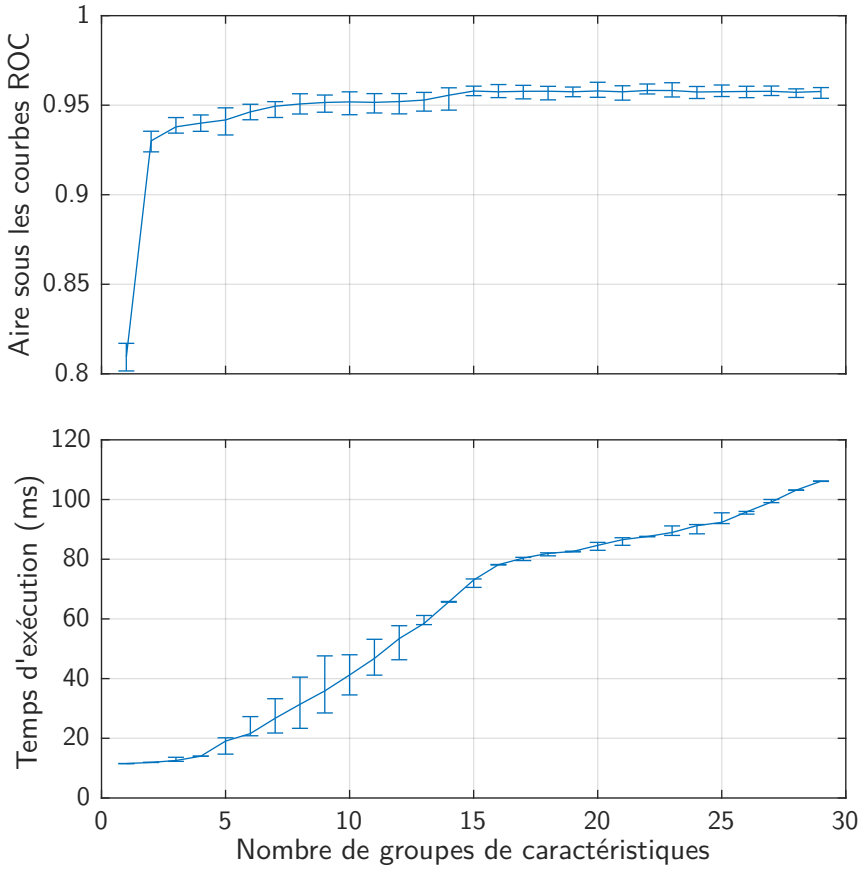


FIGURE 5.18 – Évolution de la moyenne des AUC avec *random forest* (en haut) et du temps d'exécution de l'extraction des caractéristiques (en bas) en fonction du nombre de groupes de caractéristiques sur la base d'images de Mitocheck. Ces valeurs sont moyennées sur 10 itérations de bootstrap. Les barres d'erreurs montrent le maximum et le minimum dans chaque cas.

Finalement, avec la base de Mitocheck, nous avons pu réduire le nombre de groupes de caractéristiques à 8 en moyenne, donnant un temps d'exécution inférieur à 40 ms par vignette, nous permettant ainsi d'analyser plus de 25 cellules par seconde.

Groupes de carac.	**Nombre**
MsH(ChT(im))	10
MsH(WT(im))	10
MsH(im)	10
ObjS(im)	10
TamT(ChT(im))	10
TamT(WT(im))	10
MsH(WT(FT(im)))	6
HarT(FT(im))	4
MsH(ChT(FT(im)))	4
MsH(FT(im))	4
TamT(WT(FT(im)))	2

TABLE 5.6 – Groupes de caractéristiques présents dans les 10 bootstraps avec la base d'images de Mitocheck lorsque 8 groupes sont utilisés. 6 groupes sur 8 sont présents dans les 10 itérations de bootstrap.

5.5.2.5 Temps d'exécution de la classification

Jusqu'ici, dans l'objectif de classifier des images en temps-réel, nous nous sommes focalisés sur l'optimisation de l'extraction des caractéristiques. Or, il nous a aussi fallu prendre le temps de classification en considération dans le processus. En effet, du fait de l'utilisation de plusieurs centaines d'arbres de décision, nous supposions que la classification avec *random forest* devait représenter une part importante du temps d'exécution. C'est pourquoi nous avons aussi cherché à estimer le temps d'exécution de la classification à proprement parler. Pour cela, nous avons utilisé la librairie OpenCV. Il s'agit d'une librairie développée en C++, regroupant un grand nombre de modules destinés au traitement d'images et incluant entre autres des outils d'apprentissage automatique (ITSEEZ 2015). De par sa structure, cette librairie convient à des applications temps-réel et à l'embarquement.

OpenCV contient entre autres un module nommé *RTrees* permettant d'entraîner un *random forest*. Par soucis de simplicité dans un contexte de preuve de concept, et afin d'éviter les problèmes de compatibilité entre les modèles de Matlab et de OpenCV, nous avons effectué l'entraînement d'un *random forest* en utilisant directement OpenCV. Notre objectif étant la mesure du temps d'exécution de la classification, nous n'avons pas effectué de validation croisée. Nous avons entraîné un *random forest* constitué de 300 arbres de décision, en utilisant les 12 groupes de caractéristiques les plus importants (groupes chronophages exclus), et 80 % de notre base d'images de CellCognition pour l'apprentissage (soit 127 échantillons). Nous avons testé la classification avec 20 % de la base (soit 32 échantillons). Ainsi, nous avons mesuré le temps écoulé entre le début et la fin de la classification de chaque échantillon de test à l'aide de la classe standard C++ *chrono*. Nous avons mesuré un temps d'exécution moyen de 89 µs avec une déviation standard de 20 µs. Ce temps est négligeable devant le temps de calcul des caractéristiques.

5.5.2.6 Conclusion sur *Random Forest*

Finalement, la méthode non-linéaire nous a permis d'optimiser le temps d'extraction des caractéristiques en ne calculant que celles étant considérées comme importantes dans la classification, sans dégrader la précision. Nous avons aussi montré la capacité de *random forest* à exploiter la redondance des caractéristiques lorsque des groupes importants et chronophages sont supprimés, augmentant ainsi l'importance des groupes restants.

Nous avons atteint un temps d'extraction des caractéristiques de la base de CellCognition aux alentours de 70 ms. Avec la base Mitocheck, nous parvenions à extraire les caractéristiques en moins de 40 ms. Dans le cas où une image contiendrait par exemple 300 cellules comme nous avons pu voir dans la base de CellCognition, nous sommes en mesure de traiter une image en moins de 22 s. Ce calcul est fait sans prendre en compte que nous pourrions exploiter les 8 cœurs de CPU de la NVIDIA Jetson AGX Xavier. En effet, de la même manière que le système développé dans le chapitre 4 soit exécuté en multi-tâches, nous pourrions diviser le calcul en chaque groupe de caractéristiques en une tâche permettant le calcul simultané de plusieurs groupes.

Aussi, du fait de l'utilisation d'un grand nombre de structures conditionnelles par les arbres de décision, nous pensons que *random forest* n'était pas la solution optimale pour réaliser la classification malgré son temps d'exécution faible. En effet, l'utilisation de structures conditionnelles peuvent conduire à de nombreux vidages du pipeline ralentissant ainsi le CPU, rendant aussi *random forest* difficile à accélérer sur GPU. C'est pourquoi nous nous sommes tourné vers l'utilisation d'un réseau de neurones pour effectuer la classification.

5.5.3 Classification des images avec un réseau de neurones

En recherche fondamentale, certaines applications peuvent nécessiter des temps de traitement courts. Par exemple, l'étude de la transition métaphase-anaphase montre une dynamique de certains composants de l'ordre de la seconde, voir du dixième de seconde (Elting, Suresh et Dumont 2018). Une solution pour augmenter notre vitesse de traitement des images serait d'accélérer l'extraction des caractéristiques en parallélisant les calculs sur le GPU de la NVIDIA Jetson. Or, cela sortait du champ d'étude de la thèse, l'implémentation sur GPU demandant un temps de développement très important. Pour rester dans ce contexte de parallélisation, nous nous sommes tournés vers les réseaux de neurones pour effectuer la classification et ainsi palier au problème de temps d'exécution de *random forest*. En effet, les réseaux de neurones ne font appel qu'à des fonctions arithmétiques telles que des multiplications de matrices et des fonctions non-linéaires (fonctions d'activation des neurones), les rendant ainsi accélérable sur GPU.

En revanche, contrairement à *random forest*, les réseaux de neurones souffrent plus de problèmes de sur-apprentissage, et peu d'informations peuvent être obtenues après apprentissage. Il n'est pas possible par exemple d'estimer facilement l'importance des groupes de caractéristiques dans la classification. C'est pourquoi nous utilisons toujours *random forest* pour sélectionner les groupes de caractéristiques importants dans un premier

temps, puis nous entraînons et exécutons un réseau de neurones dans un second temps avec les caractéristiques sélectionnées.

5.5.3.1 Classification

Nous avons donc construit un réseau de neurones constitué d'une couche cachée de 64 neurones (paramètre choisi de manière expérimentale), avec des fonctions d'activation de type sigmoïde (allant de -1 à 1), et une fonction de sortie de type *softmax*. Nous avons entraîné ce réseau avec une méthode de descente de gradient, consistant en l'ajustement des poids et biais des différents neurones constituant le réseau de manière itérative selon l'équation 5.3, où w_n est le vecteur de poids et de biais à l'instant n. Le détail du calcul du gradient présenté dans l'équation 5.3, où :

- α représente le taux d'apprentissage de la descente de gradient ;
- $L(\mathbf{w}_n)$ représente la fonction de coût représentant l'erreur de classification à laquelle un facteur de régularisation est ajouté (dans notre cas, nous utilisons une Mean Squared Error (MSE, Erreur Quadratique Moyenne)) ;
- λ dans l'équation 5.5 est un facteur de régularisation ajoutant la norme des poids dans la fonction de coût, permettant de réduire les poids n'apportant pas beaucoup de contribution à la classification ;
- μ représente le *momentum* permettant de prendre en considération la variation des poids à l'instant $n-1$ dans le calcul des nouveaux poids.

$$\mathbf{w}_n = \mathbf{w}_{n-1} - d\mathbf{w}_n \tag{5.3}$$

$$d\mathbf{w}_n = \alpha \times \frac{dL(\mathbf{w}_n)}{d\mathbf{w}_n} \times (1-\mu) + \mu \times d\mathbf{w}_{n-1} \tag{5.4}$$

$$L(\mathbf{w}_n) = \epsilon(\mathbf{w}_n) + \lambda \times \|\mathbf{w}_n\|_2 \tag{5.5}$$

Nous avons estimé les différents paramètres d'apprentissage de manière empirique. Nous avons choisi un taux d'apprentissage variable, augmentant lorsque la décente de gradient tend à se stabiliser. Celui-ci diminue lorsqu'il est trop important, pouvant empêcher l'algorithme de converger en oscillant autours d'une solution. La valeur initiale du taux d'apprentissage est $\eta = 0{,}01$. Nous avons aussi choisi un facteur de régularisation $\lambda = 0{,}1$ et un *momentum* $\mu = 0{,}1$.

Comme mentionné précédemment, les réseaux de neurones peuvent avoir des problèmes de sur-apprentissage. C'est pourquoi nous avons eu recours à une méthode d'arrêt prématuré de l'apprentissage en utilisant un sous-ensemble de validation du jeu de données pour pallier ce problème. Nous avons divisé notre base de caractéristique en trois sous-ensembles :

- un sous-ensemble d'apprentissage utilisé pour calculer les poids des neurones du réseau (70 %) ;
- un sous-ensemble de validation utilisé pour stopper l'apprentissage si sa fonction de coût augmente (20 %) ;

— un sous-ensemble de test utilisé uniquement pour tester le réseau entraîné (10 %).

Si la fonction de coût du jeu de validation venait à augmenter plus de 100 itérations d'affilées, nous avons considéré que le modèle était en train de sur-apprendre en apprenant les détails du jeu d'apprentissage. Nous interrompons donc l'apprentissage si cela se produit.

À l'instar des travaux sur les *random forest*, nous souhaitions observer l'évolution de la qualité de classification en fonction du nombre de groupes de caractéristiques utilisés. C'est pourquoi nous avons entraîné des réseaux de neurones avec différents nombres de groupes de caractéristiques sélectionnés avec *random forest*. Comme pour les méthodes précédentes, nous avons effectué une validation croisée de type k-fold pour valider notre méthode, avec $k = 10$. Afin de ne pas biaiser cette validation, nous avons aléatoirement initialisé les poids des neurones à la même valeur pour chaque itération de la validation croisée.

Afin d'étudier la reproductibilité des résultats de classification, nous avons de nouveau effectué plusieurs itérations de bootstrap. L'entraînement des réseaux de neurones étant plus rapide que *random forest*, nous avons réalisé 20 itérations de bootstrap. Nous avons observé une variabilité de la qualité de classification beaucoup plus importante qu'avec *random forest* (voir figure 5.19). Le nombre optimal de groupes de caractéristique est en moyenne de 15,6 groupes avec une déviation standard de 2,5 groupes. En utilisant 15 groupes de caractéristiques, la meilleure classification obtenue du bootstrap donne une matrice de confusion et des courbes ROC (voir figure 5.20) similaires à ce que nous obtenons avec *random forest* (voir figure 5.14), avec une précision globale de 83,0 % et une AUC moyenne de 0,979. Cela montre la possibilité d'utiliser un réseau de neurones pour classifier nos images après réduction du nombre de groupes de caractéristiques de la même manière qu'avec *random forest*.

5.5.3.2 Temps d'exécution de la classification

Un de nos objectifs est de comparer le temps d'exécution d'un réseau de neurones avec celui de *random forest*. Nous avons donc reproduit la même démarche qu'avec *random forest* en utilisant le module *ANN_MLP* de OpenCV (*Artificial Neural Networks - Multi-Layer Perceptrons*) (Itseez 2015). Nous avons entraîné un réseau de neurones constitué d'une couche cachée de 64 neurones, en utilisant les 15 groupes de caractéristiques les plus importants (cas optimal sans les groupes chronophages), et 80 % de notre base d'images de CellCognition pour l'apprentissage. Nous avons testé la classification avec 20 % de la base. Ainsi, nous avons mesuré un temps d'exécution moyenne entre le début et la fin de la classification de 92 µs, avec une déviation standard de 15 µs, ce qui est similaire à ce que nous obtenons avec *random forest*. Notre hypothèse d'une classification plus rapide en utilisant un réseau de neurones qu'avec *random forest* n'est donc pas correcte dans ce cas. De plus, afin d'atteindre des performances similaires à *random forest* avec les réseaux de neurones, nous devons utiliser un plus grand nombre de groupes de caractéristiques. Cela augmente donc finalement le temps d'exécution global (extraction de caractéristiques et classification).

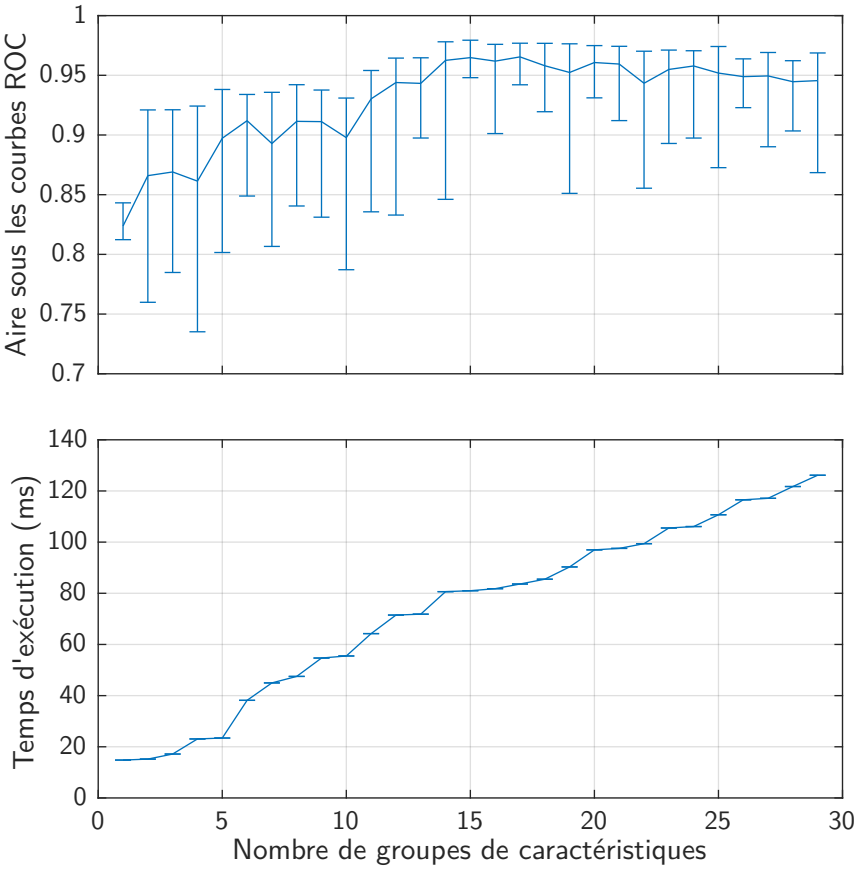
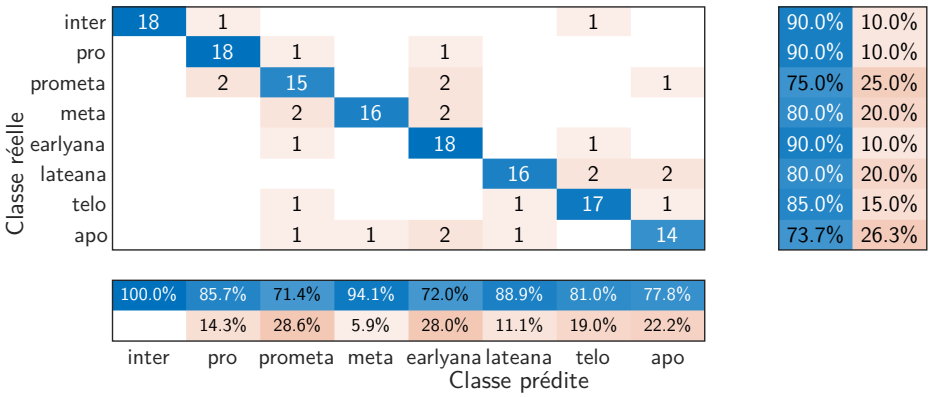
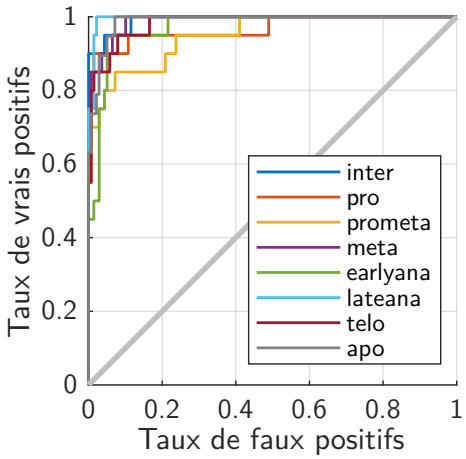


FIGURE 5.19 – Évolution de la moyenne des AUC avec des réseaux de neurones (en haut) et du temps d'exécution de l'extraction des caractéristiques (en bas) en fonction du nombre de groupes de caractéristiques sur la base d'images de CellCognition. Ces valeurs sont moyennées sur 20 itérations de bootstrap. Les barres d'erreurs montrent le maximum et le minimum dans chaque cas.



(a)



(b)

FIGURE 5.20 – Matrice de confusion (a) et courbes ROC (b) obtenues avec réseau de neurones sur les 15 caractéristiques les plus discriminantes de la base d'images de CellCognition après suppression des plus chronophages. Les tableaux sous et à droite la matrice de confusion (a) représentent respectivement la précision et la sensibilité.

5.5.3.3 Conclusion sur les réseaux de neurones

Finalement, nous avons montré la possibilité d'utiliser un réseau de neurones pour réaliser la classification sur les groupes de caractéristiques préalablement sélectionnés avec *random forest*. Nous observons une variabilité de la qualité de classification plus importante qu'avec *random forest*. En revanche, les meilleures qualités de classification obtenues avec les deux méthodes sont similaires. Aussi, nous avons observé que nous pouvions avoir un temps de classification (sans compter celui de l'extraction des caractéristiques) similaire à celui obtenu avec *random forest*.

Cependant, plusieurs éléments viennent tempérer ces résultats. Premièrement, les précisions obtenues avec les réseaux de neurones étaient plus variables qu'avec *random forest*, et seuls les meilleurs étaient similaires à ce dernier. De plus, les paramètres ont été estimés de manière empirique, ce qui a nécessité la répétition de plusieurs apprentissages. Dans un cadre applicatif, cela signifie que plusieurs réseaux de neurones devraient être entraînés afin de pouvoir sélectionner le meilleur. Cela peut représenter un temps d'apprentissage finalement plus long que celui de *random forest* en fonction du nombre de réseaux entraînés et de leur taille.

Enfin, bien que les temps d'exécution de la classification soient similaires à ceux obtenus avec *random forest*, les réseaux de neurones ont utilisé plus de groupes de caractéristiques (ici 15) pour atteindre une qualité de classification similaire à *random forest* (optimal avec 12 groupes). Cela signifie que le gain potentiel de temps de classification que nous pouvons obtenir avec son accélération sur GPU peut finalement être perdu dans le calcul de ces groupes supplémentaires.

5.6 Conclusion et discussion

Dans cette étude, nous avons proposé une méthode permettant d'embarquer et d'exécuter en temps-réel la classification d'images de microscopie représentant des cellules. Cette méthode représentait une fonction essentielle au système de microscopie intelligente présenté dans le chapitre 4. Cette méthode consiste en l'utilisation d'un outil générique d'extraction de caractéristiques d'images existant dans l'état de l'art (Orlov et al. 2008), caractéristiques sur lesquels nous avons appliqué un algorithme d'apprentissage automatique. L'extraction des caractéristiques étant la partie critique de notre processus en termes de temps d'exécution, nous avons analysé la contribution des groupes de caractéristiques dans la classification en tirant parti de la capacité de certains outils d'apprentissage automatique à estimer l'importance de leurs variables d'entrées. Nous avons donc grâce à cela sélectionné un sous-ensemble de caractéristiques optimales permettant de classifier les différentes phases de la division cellulaire avec une bonne précision tout en respectant des contraintes de temps. De même nous avons observé la capacité de la méthode non-linéaire qu'est *random forest* à exploiter la redondance des caractéristiques, lorsque certaines considérées importantes sont supprimées lors de l'optimisation, afin de conserver la même qualité de classification. Cela nous a finalement permis de remplir nos objectifs en termes de précision et de vitesse, avec une classification de 14 cellules par seconde et une précision supérieure à 83 % avec *random forest*.

Lors de l'utilisation des méthodes non-linéaires, nous avons constaté que la suppression de certaines caractéristiques à forte contribution dans la classification n'impactait pas sa précision. Malgré l'appartenance de ces caractéristiques à différents groupes utilisant des stratégies d'analyse d'images différentes, elles possèdent une certaine redondance. La diversité des caractéristiques rend cette redondance non-intuitive et serait très difficile à mesurer autrement que par des méthodes statistiques non-linéaires. La combinaison d'un grand nombre de caractéristiques proposé par WND-CHARM avec une méthode d'apprentissage non-linéaire comme *random forest* semble donc être une bonne solution pour réduire cette redondance (Tuv et al. 2009 ; Bolón-Canedo, Sánchez-Maroño et Alonso-Betanzos 2013). Des processus biologiques similaires ont été observés dans l'état de l'art, montrant une corrélation entre des caractéristiques biologiques mathématiquement indépendantes. Il a été montré par Nagao et al. par exemple que lors de la classification de cellules dans différentes phases de la division en utilisant de l'apprentissage profond, l'ajout de marqueurs fluorescents en plus de celui utilisé pour marquer les chromosomes n'améliorait pas la précision de classification (Nagao et al. 2020). En effet, les phases de la division cellulaire impliquent de nombreuses modifications de la structure cellulaire, entraînant la variation de différentes caractéristiques liées par la régulation du cycle cellulaire (Pollard et Earnshaw 2002). Des études menées au sein de notre équipe sur la longueur du fuseau mitotique, ont démontré de manière similaire qu'après application d'une ACP sur une centaine de variables, 95 % de la variabilité était contenue dans les 3 premières composantes principales (Le Cunff et al. 2021). Cela montre que la diversité des caractéristiques de la structure cellulaire observable est finalement contrôlée par un ou quelques régulateurs, comme nous avons pu observer dans notre stratégie de suppression de caractéristiques.

Nous avons développé ce processus d'optimisation en prenant en compte le fait qu'il doit être utilisable avec des petites bases d'images annotées. Il s'agit d'une contrainte fréquemment rencontrée en biologie (Shaikhina et al. 2015 ; Kourou et al. 2015 ; Foster, Koprowski et Skufca 2014). En effet, la génération des images et en particulier leur annotation sont des processus fastidieux. Il s'agit d'une contrainte que nous pouvons retrouver dans différents milieux de la recherche et de l'ingénierie. Le risque principal dans cette situation est le sur-apprentissage que l'on peut rencontrer par exemple en laissant un degré de liberté trop important à un réseau de neurones (Shaikhina et al. 2015 ; Foster, Koprowski et Skufca 2014 ; Feng, Zhou et Dong 2019 ; Pasupa et Sunhem 2016). C'est aussi pourquoi nous avons choisi *random forest* pour pallier ce problème (Breiman 2001 ; Azar et El-Metwally 2013). La réduction du nombre de caractéristiques en supprimant les moins importantes nous a effectivement permis d'optimiser le temps d'exécution, mais aussi de réduire ce risque de sur-apprentissage. Nous pourrions comparer notre méthode à une méthode de régularisation utilisée en apprentissage profond, nommée *dropout*, consistant à supprimer certains neurones lors de l'apprentissage afin de réduire ce degré de liberté (Srivastava et al. 2014 ; Borisov, Eruhimov et Tuv 2006). Finalement, notre méthode permet la sélection de variables discriminantes, offrant un compromis intéressant entre temps d'exécution et précision. De

plus, elle offre la possibilité d'utiliser un réseau de neurones une fois le sous-ensemble de variables sélectionné.

Pour aller plus loin, nous pourrions envisager d'utiliser un algorithme d'apprentissage profond, comme les réseaux de neurones convolutifs, tout en gardant à l'idée qu'un tel réseau pourrait être optimisé en ne conservant que les caractéristiques (et donc les couches de convolution) utiles à la classification (Molchanov et al. 2016). Un point intéressant avec ce genre de modèle est qu'il est envisageable de les exécuter facilement sur des GPU voir sur des puces dédiées comme on peut trouver sur les systèmes embarqués NVIDIA Jetson. Les résultats que nous avons obtenus lors de la suppression des caractéristiques chronophages de WND-CHARM, nous laissent penser que les pseudo-caractéristiques calculées par les filtres de convolution de CNN peuvent aussi être redondantes. En estimant à la fois l'importance de ces filtres et leur temps de calcul (des filtres plus gros pouvant être plus chronophages), nous pouvons envisager de suivre la même approche en supprimant les filtres chronophages en amont, tout en conservant une bonne qualité de classification en exploitant la redondance des pseudo-caractéristiques.

Troisième partie

Conclusion

Chapitre 6

Conclusion générale et perspectives

6.1 Synthèse générale

Nous avions pour objectif au cours de cette thèse de concevoir un prototype de microscope automatisé et intelligent, dont l'intérêt était d'augmenter la pertinence des images de microscopie, jusqu'ici analysées *a posteriori*. Nous souhaitions donc réaliser un système capable de modifier le cours d'une acquisition d'images sans intervention de son utilisateur, cela suffisamment rapidement afin de capturer des évènements d'intérêt pouvant être rares et de courte durée. Ce système ayant pour but d'être industrialisé, un de nos objectifs était aussi de le concevoir afin de toucher un maximum d'utilisateurs en le rendant adaptable à la majorité des expériences biologiques pouvant être envisagées. En résumé, nous devions créer un prototype asservissant l'acquisition des images et joignant généricité, rapidité et précision.

Nous avons donc mis en place un modèle fonctionnel venant s'interfacer avec la solution de pilotage de microscope commercialisée par Inscoper. Ce prototype analyse à la volée des images acquises par une caméra scientifique et retourne des commandes de pilotages au microscope afin de modifier la ou les modalités d'acquisition en fonction des résultats d'analyse des images. Nous avons confronté ce modèle à différents cas d'usage plus ou moins complexes tout au long de son développement. Grâce à cela, nous avons mis au point une solution fonctionnelle théorique générique, dont la décomposition en fonctions élémentaires lui confère de la modularité. Pour répondre à notre besoin d'une exécution en temps-réel, nous avons implémenté notre solution sur un système embarqué grand public. Nous avons démontré le bon fonctionnement de notre système d'asservissement en y exécutant une application de recherche de cellules en mitoses. Pour cela, nous avons simulé des résultats d'analyse des images. Nous avons ainsi pu observer sa capacité à exécuter des tâches en parallèle et à interrompre des séquences d'acquisition lors de la détection d'événements d'intérêt.

Lors de la conception de ce système, nous avons identifié la fonction d'analyse d'images permettant de détecter des évènements d'intérêt comme la partie critique en termes de temps d'exécution. Nous avons donc mis au point une méthode de classification d'images de microscopie, à l'aide d'un outil générique d'extraction de caractéristiques de ces images

et d'algorithmes d'IA. Afin de respecter nos contraintes de temps, tout en bénéficiant de la généricité de ces outils, nous avons estimé la pertinence des caractéristiques lors de leur classification avec la méthode d'apprentissage automatique *random forest*, afin de ne calculer que celles utiles à cette classification. Nous avons ainsi réduit le temps d'exécution de l'extraction des caractéristiques. En appliquant cette méthode pour classifier des images dans 8 classes représentant des cellules à différentes phases du cycle cellulaire, nous avons pu exploiter la capacité de *random forest* à compenser la suppression de caractéristiques chronophages, malgré leur importance pour la classification. Nous avons pu ainsi classifier jusqu'à 14 cellules par seconde avec une précision allant jusqu'à 83 %. De plus, en utilisant cette même stratégie d'optimisation, nous avons entraîné un réseau de neurones avec un nombre réduit de caractéristiques, et obtenu une précision de classification similaire à *random forest*.

Finalement, l'ensemble des travaux que nous avons réalisés jusqu'ici ont permis de mettre en place un système capable d'interrompre des séquences d'acquisitions en fonction de résultats d'analyses d'images. Cette notion d'interruption est un point clé de notre contribution, différenciant notre solution de celles de l'état de l'art qui fonctionnent de manière séquentielle. Cela nous permet d'optimiser le temps d'exécution des différentes fonctionnalités, rendant notre concept conforme à des applications impliquant la détection d'évènements rares et/ou brefs. Enfin, le fait d'avoir confronté notre système à des applications théoriques plus ou moins complexes, est en adéquation avec la variété d'application pouvant être réalisées par des biologistes. Le résultat de cette confrontation est un système d'asservissement de microscope générique et modulable, par opposition aux solutions de l'état de l'art, spécifiques à quelques applications. Cela représente donc un premier pas vers l'industrialisation de notre preuve de concept.

6.2 Discussion générale

La généricité de notre système est une contrainte clé lors de son développement. Aujourd'hui, nous pouvons configurer notre système pour réaliser différentes expériences de microscopie intelligente, sans avoir à le reprogrammer. Or, les cas d'études théoriques auxquels nous avons confronté notre solution fonctionnelle, représentent des applications actuellement envisagées par des ingénieurs et des chercheurs d'Inscoper et de l'IGDR. Il est aujourd'hui difficile d'imaginer les applications et expériences de microscopie intelligente qui seront réalisées dans les années voir décennies à venir. Intrinsèquement, la décomposition de notre solution en fonctions élémentaires en fait un système modulaire. Cette modularité nous a permis par exemple de concevoir le pilotage du microscope et l'analyse d'images indépendamment. Ainsi, chaque élément de la solution fonctionnelle peut être modifié et évoluer, sans avoir à modifier l'ensemble du système. Nous pouvons donc par exemple facilement envisager de remplacer notre solution de détection d'évènements d'intérêt par des méthodes d'apprentissage profond (CNN), sans influencer les autres fonctions du système. De manière plus générale, cela confère à notre solution une évolutivité qui sera utile à son adaptation aux applications de demain.

Les réseaux de neurones, et particulièrement les CNN, sont aujourd'hui de plus en plus utilisés en analyse d'images. Dans le milieu de la détection d'objets, il existe des modèles permettant d'identifier les régions d'intérêt et de les classifier directement à partir d'une image contenant plusieurs objets. Le temps d'analyse des images pour un traitement à la volée est un paramètre important dans de nombreux domaines. C'est pourquoi des CNN tels que YOLO (Redmon et Farhadi 2017) et RetinaNet (Lin et al. 2017) ont été développés ces dernières années. Contrairement aux approches usuelles de détection d'objets (séparant les étapes de segmentation, de calcul de caractéristiques d'images, et de classification des objets), les calculs réalisés par ces réseaux, comme les convolutions, peuvent facilement être parallélisés sur des GPU dans le but de les accélérer. De plus, leurs temps d'exécution ne sont pas dépendant du nombre d'objets présents sur chaque image. Dans le milieu de la biologie, Waithe et al. ont par exemple confronté ces deux réseaux, et ont obtenu des temps permettant le traitement de 4 à 5 images par seconde (Waithe et al. 2020). Leur utilisation est donc envisageable pour la détection d'évènements biologiques de durée inférieure à la seconde.

Les résultats que nous obtenons lors de l'optimisation d'une approche existante de classification d'images, nous montrent une redondance des caractéristiques d'images pouvant être exploitée afin de supprimer des caractéristiques longues à calculer. Ainsi, nous supposons que cette redondance peut être observée et exploitée de manière similaire sur des CNN tels que YOLO et RetinaNet, afin d'améliorer leurs performances en supprimant des filtres de convolution chronophages, indépendamment de leur poids pour la détection des objets (Molchanov et al. 2016).

Les travaux réalisés au cours de cette thèse à la frontière entre l'ingénierie et la recherche, sont le fruit d'un partenariat entre Inscoper et les équipes CeDRE et MFQ de l'IGDR. Le fait de collaborer à la fois avec des partenaires des milieux industriels et académiques a été une grande source d'idées tout au long de ce projet. Bien que les enjeux et objectifs de ces deux milieux semblent différents, ceux-ci sont loin d'être incompatibles. Au contraire, cela nous a permis de prendre du recul sur le développement de notre solution en y apportant deux regards différents. Sans cette collaboration, les résultats obtenus n'auraient sûrement pas été les mêmes. En effet, nous ne nous serions probablement pas posé les questions de généricité et de modularité dans un cadre purement académique, ces enjeux étant plus importants dans le milieu industriel. De la même manière, le côté recherche de ce projet nous a poussé à explorer des solutions de traitement d'images optimisées pour l'analyse en temps-réel, laissant ainsi la porte ouverte à un grand nombre de pistes que nous pouvons explorer.

Ce constat s'inscrit de manière plus générale dans les bénéfices qu'apporte la pluridisciplinarité dans les milieux de la recherche et de l'ingénierie. Cette thèse a été réalisée dans un environnement liant les domaines de la biologie, de la physique mais aussi de l'informatique, du traitement du signal et des mathématiques. Cette vaste collaboration nous a permis de confronter nos idées et notre solution à des chercheurs et des potentiels utilisateurs dont les attentes peuvent être différentes quant à la microscopie intelligente. Cela appuie ainsi notre observation de l'utilité de la collaboration entres des secteurs de recherche ayant des enjeux différents, outre les aspects académiques et industriels. Des

études sur la pluridisciplinarité dans la recherche ont effectivement montré qu'elle pouvait être bénéfique dans la résolution de problèmes complexes, si celle-ci n'est pas utilisée dans des proportions trop importantes. Auquel cas, elle peut devenir contre-productive (Choi et Pak 2008 ; Martín-Alcázar, Ruiz-Martínez et Sánchez-Gardey 2020).

Au-delà des aspects pluridisciplinaires, réaliser cette thèse CIFRE au sein d'Inscoper nous a placé dans un contexte particulier dans lequel une solution de pilotage de microscopes existait. Nous avons donc entamé ce projet en nous abstrayant des aspects de pilotages des différents périphériques du microscope pour nous concentrer sur son asservissement et sur l'analyse des images. Cela nous a amené à nous poser la question de la stratégie que nous aurions suivi dans le cas où nous avions commencé ce projet de zéro. Nous avons fait le choix d'utiliser un système embarqué pour y implémenter notre solution suite à l'observation des performances que présente le module de pilotage d'Inscoper par rapport aux solutions sur ordinateur. Nous pouvons penser que les nombreuses solutions de pilotages disponibles sur ordinateur auraient pu orienter notre choix vers ce genre de support matériel afin de pouvoir utiliser notre prototype sur un grand nombre de microscopes. Cependant, un objectif principal de notre projet est de respect des contraintes temps-réel. Les systèmes embarqués sont plus appropriés pour ce genre de contrainte et permettent une meilleure reproductibilité des résultats. De plus, les microcontrôleurs embarquent aujourd'hui de plus en plus de puissance de calcul. Nous pensons donc que la solution embarquée reste le meilleur choix pour notre concept, même si nous avions commencé ce projet sans module de pilotage.

Finalement, la microscopie n'est pas le seul domaine s'orientant vers des systèmes intelligents et autonomes. L'intelligence artificielle a récemment soulevé des questions d'éthique dans plusieurs domaines, telles que des questions de responsabilité en cas d'accident ou de dysfonctionnement. Nous nous sommes donc demandé si ces problèmes d'éthique concernaient aussi la microscopie. En recherche fondamentale, la microscopie reste un milieu très souple en termes de "droit à l'erreur", et laissant la possibilité d'une analyse *a posteriori*. En revanche, ces questions seront d'actualité lorsque nous irons vers des applications médicales nécessitant une certaine précision, et pouvant impliquer des vies humaines, au même titre que le milieu de la voiture autonome par exemple.

6.3 Perspectives

6.3.1 Perspectives applicatives

Du côté de la recherche appliquée en biologie, le HCS combine des techniques de microscopie de fluorescence quantitative et de méthodes d'automatisation dans le but d'évaluer l'effet de bibliothèques de drogues sur le phénotype de cellules vivantes. Or, ces méthodes nécessitent actuellement l'acquisition d'une grande quantité d'images analysées *a posteriori*, et ne permettent pas la manipulation des échantillons en cours d'acquisition. Il s'agit aujourd'hui de technique purement observatoires. Ces méthodes de criblage ne permettent pas une sélection fine de ces drogues, pouvant ainsi être rejetées lors d'essais précliniques. Conformément aux problématiques de la recherche fondamentale, l'évolution de l'automatisation de microscopies vers des systèmes capables de modifier des expériences

en temps-réel, en injectant ou activant des drogues à un instant précis du cycle cellulaire, permettra d'affiner le criblage *a priori*.

Pour aller plus loin, le potentiel d'un tel système de microscopie automatisé permettrait de manipuler ou traiter des échantillons vivants à des instants très spécifiques de leur développement. Certains domaines de recherche fondamentale en biologie étudient la dynamique des différents composants et acteurs dans la division cellulaire. Ces études nécessitent souvent l'utilisation de méthodes de photo-manipulation telles que le photo-blanchiment de cellules fluorescentes ou encore la photo-ablation de composants d'une cellule en division comme les centrosomes ou le fuseau mitotique. Or, ces manipulations sont aujourd'hui fortement supervisées par des experts. La microscopie intelligente ouvre donc la porte à une automatisation de ces manipulations, pouvant les rendre non-supervisée, nous laissant envisager à réaliser des expériences de type HCS impliquant de la photo-manipulation.

6.3.2 Perspectives expérimentales

Avant d'entreprendre l'industrialisation de notre solution de microscopie intelligente, plusieurs expériences doivent être réalisées afin de démontrer ses performances. Ces expériences devront être réalisées avec des échantillons biologiques vivants afin de confronter notre système à des conditions réelles d'utilisation.

Dans cette thèse, la conception de notre solution de microscopie intelligente et l'élaboration de notre stratégie d'optimisation d'une analyse d'images ont été développées de manière distincte. Ces deux éléments doivent à présent être exécutés simultanément afin de constituer un même système. Ainsi, nous chercherons à évaluer le temps d'exécution de l'analyse d'images en condition réelle, lorsque celle-ci est exécutée avec les fonctions de pilotage du microscope et de prise de décision. Cela nous permettra par exemple de déterminer le temps écoulé entre le moment où une image a été capturée, et le moment où une décision est prise en fonction du contenu de cette image.

Le mécanisme d'interruption de modalités d'acquisition d'images est une plus-value centrale de notre système, par rapport aux solutions de microscopie intelligente de l'état de l'art. Au cours de cette thèse, nous avons pu observer le bon fonctionnement de cette interruption. Nous chercherons donc ensuite à mesurer l'intervalle de temps écoulé entre la détection d'un évènement d'intérêt et la modification de la modalité d'acquisition de cet évènement. Nous comparerons ce temps à des systèmes similaires tels que Micropilot (Conrad et al. 2011), afin de démontrer l'intérêt du fonctionnement évènementiel de notre solution pour la détection de phénomènes rares et courts en biologie, par rapport au fonctionnement séquentiel des solutions existantes.

De manière plus générale, une application de la microscopie intelligente sera de réaliser des expériences complexes de microscopie, comme de la photo-manipulation, dans un contexte de HCS. Le but de telles expériences sera de réaliser un grand nombre de manipulations complexes sans intervention de l'utilisateur. Il sera donc intéressant d'évaluer la capacité de notre solution à réaliser ces expériences sur de longues durées (plusieurs heures, jours, voir semaines). Afin de démontrer la plus-value de sa méthode de fonctionnement, nous mesurerons le nombre d'expériences réalisées suivant une modalité

d'acquisition complexe telle que de la photo-manipulation, sur un intervalle de temps donné. Nous comparerons ce nombre avec celui obtenu avec d'autres solutions de l'état de l'art telles que Micropilot (Conrad et al. 2011), en réalisant la même expérience.

6.3.3 Perspectives d'évolutions technologiques

Jusqu'ici, nous nous sommes uniquement concentré sur la réalisation d'un module de microscopie intelligente, analysant les images et contrôlant les modalités d'acquisition du microscope. Tout comme la solution de pilotage commercialisée par Inscoper aujourd'hui, ce module nécessite une interface utilisateur (*Inscoper* 2020). Du fait de sa généricité, l'application devant être réalisée par notre solution n'est pas définie à l'avance. C'est donc un rôle de cette interface, qui doit permettre à un utilisateur de définir et configurer l'expérience de microscopie intelligente qu'il souhaite réaliser. Elle devra répondre à des contraintes similaires à celles que nous nous sommes fixés en termes de flexibilités, tout en conservant l'ergonomie et la simplicité d'utilisation de l'interface actuellement développée par Inscoper.

L'aspect temps-réel dans l'automatisation de la microscopie est un point primordial pour l'observation et le traitement d'événements rares et de courte durée. Cette preuve de concept est une base solide respectant des contraintes de temps, que nous avons déjà confronté à de nombreuses applications théoriques. Cela fait d'elle une solution qu'il sera facile de faire évoluer en y ajoutant des fonctionnalités. Un exemple simple nous ayant été proposé par des biologistes est l'ajout d'une fonctionnalité permettant l'exécution différée de séquences d'acquisition sur un objet d'intérêt. Lors de la recherche de la transition métaphase-anaphase, on pourrait imaginer une application permettant d'anticiper l'arriver de cette transition lorsqu'une cellule en prophase ou en métaphase est détectée. L'ajout d'une telle fonction permettrait donc de continuer à scanner d'autres cellules et de revenir plusieurs minutes plus tard sur cette cellule qui se sera rapprochée de la transition métaphase-anaphase. Ce genre de fonctionnalité pourra être intégré grâce à l'ajout de fonctions au sein de notre architecture fonctionnelle.

Dans de nombreux domaines, l'analyse d'images est un élément critique des applications temps-réel du fait de son intensivité en termes de calculs. Au-delà de l'évolution des outils d'accélération matérielle tels que les GPU ou les circuits spécialisés, cela a donné lieu ces dernières années à de nombreuses études ayant pour but d'optimiser ces analyses. En particulier, les réseaux de neurones convolutifs sont devenus un outil incontournable de l'analyse d'images. Certains réseaux telles que YOLO (Redmon et Farhadi 2017) ou Faster R-CNN (Ren et al. 2015) ont montré des performances proches du temps-réel lors de la détection d'objet sur les images de microscopie (Waithe et al. 2020). Leur optimisation pour des applications temps-réel est un domaine de recherche prenant sa place ces dernières années dans le milieu de la microscopie.

À l'instar de notre solution consistant à optimiser un outil générique d'extraction de caractéristiques d'images en supprimant les caractéristiques non-importante pour la classification, la suppression de couches de calculs est aujourd'hui appliquée dans des CNN (Molchanov et al. 2016). Notre utilisation de méthodes génériques d'extraction de caractéristiques d'images et de classification s'apparente aux méthodes d'apprentissage

profond. Notre observation de la redondance de ces caractéristiques, et de l'hétérogénéité de leur contribution dans la classification, nous laisse penser que des méthodes similaires d'analyse de redondance et d'importance des informations au sein de réseaux de neurones convolutifs permettraient de les optimiser (Molchanov et al. 2016). En alliant cette optimisation avec l'accélération matérielle qu'offrent un grand nombre de systèmes embarqués, nous sommes aujourd'hui confiants dans l'utilisation des réseaux de neurones convolutifs dans l'automatisation des systèmes de microscopie.

Bibliographie

Abramowitz, Mortimer et al. (2020). *Fluorescence Microscopy - Anatomy of the Fluorescence Microscope | Olympus Life Science*. URL : https://www.olympus-lifescience.com/en/microscope-resource/primer/techniques/fluorescence/anatomy/fluoromicroanatomy/ (visité le 13/11/2020).

Ali, Ahmad et al. (1er fév. 2016). « Visual object tracking—classical and contemporary approaches ». In : *Frontiers of Computer Science* 10.1, p. 167-188. ISSN : 2095-2236. DOI : 10.1007/s11704-015-4246-3.

Asano, S., T. Maruyama et Y. Yamaguchi (août 2009). « Performance comparison of FPGA, GPU and CPU in image processing ». In : *2009 International Conference on Field Programmable Logic and Applications*. 2009 International Conference on Field Programmable Logic and Applications. ISSN : 1946-1488, p. 126-131. DOI : 10.1109/FPL.2009.5272532.

Azar, Ahmad Taher et Shereen M. El-Metwally (1er déc. 2013). « Decision tree classifiers for automated medical diagnosis ». In : *Neural Computing and Applications* 23.7, p. 2387-2403. ISSN : 1433-3058. DOI : 10.1007/s00521-012-1196-7.

Bacon, David F., Rodric Rabbah et Sunil Shukla (1er avr. 2013). « FPGA programming for the masses ». In : *Communications of the ACM* 56.4, p. 56-63. ISSN : 0001-0782. DOI : 10.1145/2436256.2436271.

Balluet, M. et al. (2020). « Method for managing command blocks for a microscopy imaging system, corresponding computer program, storage means and device ». Brevet en attente.

Banfalvi, Gaspar (2017). « Overview of Cell Synchronization ». In : *Cell Cycle Synchronization : Methods and Protocols*. Sous la dir. de Gaspar Banfalvi. Methods in Molecular Biology. New York, NY : Springer, p. 3-27. ISBN : 978-1-4939-6603-5. DOI : 10.1007/978-1-4939-6603-5_1.

Baroux, Célia et Veit Schubert (2018). « Technical Review : Microscopy and Image Processing Tools to Analyze Plant Chromatin : Practical Considerations ». In : *Plant Chromatin Dynamics : Methods and Protocols*. Sous la dir. de Marian Bemer et Célia Baroux. Methods in Molecular Biology. New York, NY : Springer, p. 537-589. ISBN : 978-1-4939-7318-7. DOI : 10.1007/978-1-4939-7318-7_31.

Bishop, Christopher M. (2006). *Pattern recognition and machine learning*. Information science and statistics. New York : Springer. 738 p. ISBN : 978-0-387-31073-2.

Bolón-Canedo, Verónica, Noelia Sánchez-Maroño et Amparo Alonso-Betanzos (1er mar. 2013). « A review of feature selection methods on synthetic data ». In :

Knowledge and Information Systems 34.3, p. 483-519. ISSN : 0219-3116. DOI : 10.1007/s10115-012-0487-8.

BORISOV, Alexander, Victor ERUHIMOV et Eugene TUV (2006). « Tree-Based Ensembles with Dynamic Soft Feature Selection ». In : *Feature Extraction : Foundations and Applications*. Sous la dir. d'Isabelle GUYON et al. Studies in Fuzziness and Soft Computing. Berlin, Heidelberg : Springer, p. 359-374. ISBN : 978-3-540-35488-8. DOI : 10.1007/978-3-540-35488-8_16.

BREIMAN, Leo (1er oct. 2001). « Random Forests ». In : *Machine Learning* 45.1, p. 5-32. ISSN : 1573-0565. DOI : 10.1023/A:1010933404324.

BREIMAN, Leo et al. (1984). *Classification and regression trees*. CRC press.

C++ reference (2020). URL : https://cppreference.com/ (visité le 29/10/2020).

CARPENTER, Anne E. et al. (31 oct. 2006). « CellProfiler : image analysis software for identifying and quantifying cell phenotypes ». In : *Genome Biology* 7, R100. ISSN : 1474-760X. DOI : 10.1186/gb-2006-7-10-r100.

CASTILLO-SECILLA, J. M. et al. (1er mar. 2017). « Autofocus method for automated microscopy using embedded GPUs ». In : *Biomedical Optics Express* 8.3, p. 1731-1740. ISSN : 2156-7085. DOI : 10.1364/BOE.8.001731.

CHARTRAND, Gabriel et al. (1er nov. 2017). « Deep Learning : A Primer for Radiologists ». In : *RadioGraphics* 37.7. Publisher : Radiological Society of North America, p. 2113-2131. ISSN : 0271-5333. DOI : 10.1148/rg.2017170077.

CHEN, Chih-Yung, Rey-Chue HWANG et Yu-Ju CHEN (1er jan. 2010). « A passive auto-focus camera control system ». In : *Applied Soft Computing* 10.1, p. 296-303. ISSN : 1568-4946. DOI : 10.1016/j.asoc.2009.07.007.

CHEN, D. et D. SINGH (jan. 2013). « Fractal video compression in OpenCL : An evaluation of CPUs, GPUs, and FPGAs as acceleration platforms ». In : *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC 2013). Yokohama : IEEE, p. 297-304. DOI : 10.1109/ASPDAC.2013.6509612.

CHOI, Bernard C. K. et Anita W. P. PAK (1er fév. 2008). « Multidisciplinarity, interdisciplinarity, and transdisciplinarity in health research, services, education and policy : 3. Discipline, inter-discipline distance, and selection of discipline ». In : *Clinical and Investigative Medicine*, E41-E48. ISSN : 1488-2353. DOI : 10.25011/cim.v31i1.3140.

COLE, Richard (3 sept. 2014). « Live-cell imaging ». In : *Cell Adhesion & Migration* 8.5. Publisher : Taylor & Francis, p. 452-459. ISSN : 1933-6918. DOI : 10.4161/cam.28348.

CONRAD, Christian et al. (mar. 2011). « Micropilot : automation of fluorescence microscopy-based imaging for systems biology ». In : *Nature Methods* 8.3, p. 246-249. ISSN : 1548-7091, 1548-7105. DOI : 10.1038/nmeth.1558.

DAI, Jifeng et al. (2016). « R-FCN : Object Detection via Region-based Fully Convolutional Networks ». In : *Advances in Neural Information Processing Systems* 29, p. 379-387.

DURAND, Audrey et al. (7 déc. 2018). « A machine learning approach for online automated optimization of super-resolution optical microscopy ». In : *Nature Communications* 9.1. Number : 1 Publisher : Nature Publishing Group, p. 5247. ISSN : 2041-1723. DOI : 10.1038/s41467-018-07668-y.

Edelstein, Arthur D. et al. (2014). « Advanced methods of microscope control using μManager software ». In : *Journal of biological methods* 1.2. ISSN : 2326-9901. DOI : 10.14440/jbm.2014.36.

Eisenstein, Michael (nov. 2020). « Smart solutions for automated imaging ». In : *Nature Methods* 17.11. Number : 11 Publisher : Nature Publishing Group, p. 1075-1079. ISSN : 1548-7105. DOI : 10.1038/s41592-020-00988-2.

Elting, Mary Williard, Pooja Suresh et Sophie Dumont (2018). « The Spindle : Integrating Architecture and Mechanics across Scales ». In : *Trends in Cell Biology* 28.11, p. 896-910. ISSN : 1879-3088. DOI : 10.1016/j.tcb.2018.07.003.

Esner, Milan, Felix Meyenhofer et Marc Bickle (2018). « Live-Cell High Content Screening in Drug Development ». In : *High Content Screening : A Powerful Approach to Systems Cell Biology and Phenotypic Drug Discovery*. Sous la dir. de Paul A. Johnston et Oscar J. Trask. Methods in Molecular Biology. New York, NY : Springer, p. 149-164. ISBN : 978-1-4939-7357-6. DOI : 10.1007/978-1-4939-7357-6_10.

Fawcett, Tom (juin 2006). « An introduction to ROC analysis ». In : *Pattern Recognition Letters* 27.8, p. 861-874. ISSN : 01678655. DOI : 10.1016/j.patrec.2005.10.010.

Feng, Shuo, Huiyu Zhou et Hongbiao Dong (15 jan. 2019). « Using deep neural network with small dataset to predict material defects ». In : *Materials & Design* 162, p. 300-310. ISSN : 0264-1275. DOI : 10.1016/j.matdes.2018.11.060.

Feng Ning et al. (sept. 2005). « Toward automatic phenotyping of developing embryos from videos ». In : *IEEE Transactions on Image Processing* 14.9. Conference Name : IEEE Transactions on Image Processing, p. 1360-1371. ISSN : 1941-0042. DOI : 10.1109/TIP.2005.852470.

Fisher, R. A. (1936). « The Use of Multiple Measurements in Taxonomic Problems ». In : *Annals of Eugenics* 7.2, p. 179-188. ISSN : 2050-1439. DOI : 10.1111/j.1469-1809.1936.tb02137.x.

Foster, Kenneth R., Robert Koprowski et Joseph D. Skufca (5 juil. 2014). « Machine learning, medical diagnosis, and biomedical engineering research - commentary ». In : *Biomedical Engineering Online* 13, p. 94. ISSN : 1475-925X. DOI : 10.1186/1475-925X-13-94.

Freund, Yoav et Robert E. Schapire (3 juil. 1996). « Experiments with a new boosting algorithm ». In : *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*. ICML'96. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., p. 148-156. ISBN : 978-1-55860-419-3.

Gao, Zhimin et al. (2017). « HEp-2 Cell Image Classification With Deep Convolutional Neural Networks ». In : *IEEE journal of biomedical and health informatics* 21.2, p. 416-428. ISSN : 2168-2208. DOI : 10.1109/JBHI.2016.2526603.

GCC (2020). *GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF)*. URL : https://gcc.gnu.org/ (visité le 04/11/2020).

Georgis, Georgios, George Lentaris et Dionysios Reisis (1er août 2019). « Acceleration techniques and evaluation on multi-core CPU, GPU and FPGA for image processing

and super-resolution ». In : *Journal of Real-Time Image Processing* 16.4, p. 1207-1234. ISSN : 1861-8219. DOI : 10.1007/s11554-016-0619-6.

GOKCEN, Ibrahim et Jing PENG (2002). « Comparing Linear Discriminant Analysis and Support Vector Machines ». In : *Advances in Information Systems*. Sous la dir. de Tatyana YAKHNO. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, p. 104-113. ISBN : 978-3-540-36077-3. DOI : 10.1007/3-540-36077-8_10.

GOUY, Isaac (2020). *Which programming language is fastest ? | Computer Language Benchmarks Game*. URL : https://benchmarksgame-team.pages.debian.net/benchmarksgame/ (visité le 27/10/2020).

Hamamatsu (2020). URL : https://www.hamamatsu.com (visité le 27/10/2020).

HARALICK, Robert M., K. SHANMUGAM et Its'Hak DINSTEIN (nov. 1973). « Textural Features for Image Classification ». In : *IEEE Transactions on Systems, Man, and Cybernetics* SMC-3.6. Conference Name : IEEE Transactions on Systems, Man, and Cybernetics, p. 610-621. ISSN : 2168-2909. DOI : 10.1109/TSMC.1973.4309314.

HARDER, Nathalie et al. (1er nov. 2009). « Automatic analysis of dividing cells in live cell movies to detect mitotic delays and correlate phenotypes in time ». In : *Genome Research* 19.11, p. 2113-2124. ISSN : 1088-9051, 1549-5469. DOI : 10.1101/gr.092494.109.

HE, Jiaye et Jan HUISKEN (9 jan. 2020). « Image quality guided smart rotation improves coverage in microscopy ». In : *Nature Communications* 11.1. Number : 1 Publisher : Nature Publishing Group, p. 150. ISSN : 2041-1723. DOI : 10.1038/s41467-019-13821-y.

HELD, Michael et al. (sept. 2010). « CellCognition : time-resolved phenotype annotation in high-throughput live cell imaging ». In : *Nature Methods* 7.9, p. 747-754. ISSN : 1548-7091, 1548-7105. DOI : 10.1038/nmeth.1486.

HELL, S. W. et J. WICHMANN (1er juin 1994). « Breaking the diffraction resolution limit by stimulated emission : stimulated-emission-depletion fluorescence microscopy ». In : *Optics Letters* 19.11, p. 780-782. ISSN : 0146-9592. DOI : 10.1364/ol.19.000780.

HUISKEN, Jan et al. (13 août 2004). « Optical Sectioning Deep Inside Live Embryos by Selective Plane Illumination Microscopy ». In : *Science* 305.5686. Publisher : American Association for the Advancement of Science Section : Report, p. 1007-1009. ISSN : 0036-8075, 1095-9203. DOI : 10.1126/science.1100035.

Inscoper (2020). *INSCOPER | New Microscope Imaging Software Fluorescence Microscopy*. URL : https://www.inscoper.com/ (visité le 14/10/2020).

IGDR (2020). *Institut Génétique & Développement de Rennes*. URL : https://igdr.univ-rennes1.fr/ (visité le 16/11/2020).

ITSEEZ (2015). *Open Source Computer Vision Library*. https://github.com/itseez/opencv.

JOLLIFFE, Ian T. (2002). *Principal Component Analysis*. 2e éd. Springer Series in Statistics. New York : Springer-Verlag. ISBN : 978-0-387-95442-4. DOI : 10.1007/b98835.

JOLLIFFE, Ian T. et Jorge CADIMA (13 avr. 2016). « Principal component analysis : a review and recent developments ». In : *Philosophical Transactions of the Royal Society*

A : Mathematical, Physical and Engineering Sciences 374.2065. Publisher : Royal Society, p. 20150202. DOI : 10.1098/rsta.2015.0202.

KOTSIANTIS, S. B., I. D. ZAHARAKIS et P. E. PINTELAS (nov. 2006). « Machine learning : a review of classification and combining techniques ». In : *Artificial Intelligence Review* 26.3, p. 159-190. ISSN : 0269-2821, 1573-7462. DOI : 10.1007/s10462-007-9052-3.

KOUROU, Konstantina et al. (2015). « Machine learning applications in cancer prognosis and prediction ». In : *Computational and Structural Biotechnology Journal* 13, p. 8-17. ISSN : 2001-0370. DOI : 10.1016/j.csbj.2014.11.005.

LARSON MARTIN G. (1er jan. 2008). « Analysis of Variance ». In : *Circulation* 117.1, p. 115-121. DOI : 10.1161/CIRCULATIONAHA.107.654335.

LE CUNFF, Y. et al. (2021). « Structured heterogeneity of mitotic spindle elongation suggests a reduced number of independent mechanisms at work during the C. elegans zygote mitosis ». In preparation.

Leica-Microsystems (2021). *Leica-Microsystems : Laica ASAF software*. URL : http://www.leica-microsystems.com/products/microscope-software/life-sciences/las-af-advanced-fluorescence/ (visité le 29/03/2021).

LIN, Tsung-Yi et al. (2017). « Focal Loss for Dense Object Detection ». In : Proceedings of the IEEE International Conference on Computer Vision, p. 2980-2988.

LOGAMBAL, G. et V. SARAVANAN (avr. 2015). « Cancer diagnosis using automatic mitotic cell detection and segmentation in histopathological images ». In : *2015 Global Conference on Communication Technologies (GCCT)*. 2015 Global Conference on Communication Technologies (GCCT), p. 128-132. DOI : 10.1109/GCCT.2015.7342638.

MAHECIC, Dora et Suliana MANLEY (jan. 2019). « Smart microscopy : adaptive temporal sampling ». Poster. Quantitative BioImaging 2019.

MANN, Amandeep Kaur et Navneet KAUR (27 mai 2013). « Review Paper on Clustering Techniques ». In : *Global Journal of Computer Science and Technology*. ISSN : 0975-4172.

MARTIN, P. et al. (2019). *OpenNN : Open Neural Network Library*. URL : https://www.opennn.net/.

MARTÍN-ALCÁZAR, Fernando, Marta RUIZ-MARTÍNEZ et Gonzalo SÁNCHEZ-GARDEY (1er déc. 2020). « Deepening the Consequences of Multidisciplinarity on Research : The Moderating Role of Social Capital ». In : *Minerva* 58.4, p. 559-583. ISSN : 1573-1871. DOI : 10.1007/s11024-020-09404-7.

MEIJERING, E. (sept. 2012). « Cell Segmentation : 50 Years Down the Road ». In : *IEEE Signal Processing Magazine* 29.5. Conference Name : IEEE Signal Processing Magazine, p. 140-145. ISSN : 1558-0792. DOI : 10.1109/MSP.2012.2204190.

MetaMorph (2020). *MetaMorph Microscopy Automation and Image Analysis Software*. URL : https://www.moleculardevices.com (visité le 13/11/2020).

Micro-Manager (2020). URL : https://micro-manager.org/ (visité le 13/11/2020).

MOLCHANOV, Pavlo et al. (1er nov. 2016). « Pruning Convolutional Neural Networks for Resource Efficient Inference ». In : *arXiv e-prints* 1611, arXiv :1611.06440.

Nagao, Yukiko et al. (22 avr. 2020). « Robust Classification of Cell Cycle Phase and Biological Feature Extraction by Image-Based Deep Learning ». In : *Molecular Biology of the Cell*. Publisher : American Society for Cell Biology (mboc), mbc.E20–03-0187. ISSN : 1059-1524. DOI : 10.1091/mbc.E20-03-0187.

Nitta, Nao et al. (20 sept. 2018). « Intelligent Image-Activated Cell Sorting ». In : *Cell* 175.1, 266-276.e13. ISSN : 0092-8674, 1097-4172. DOI : 10.1016/j.cell.2018.08.028.

Nketia, Thomas A. et al. (15 fév. 2017). « Analysis of live cell images : Methods, tools and opportunities ». In : *Methods*. Image Processing for Biologists 115, p. 65-79. ISSN : 1046-2023. DOI : 10.1016/j.ymeth.2017.02.007.

NVIDIA Embedded Systems (2020). *NVIDIA Embedded Systems for Next-Gen Autonomous Machines*. URL : https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/ (visité le 13/10/2020).

ODROID (2020). URL : https://www.hardkernel.com/ (visité le 17/11/2020).

Orlov, Nikita et al. (août 2008). « WND-CHARM : Multi-purpose image classification using compound image transforms ». In : *Pattern Recognition Letters* 29.11, p. 1684-1693. ISSN : 01678655. DOI : 10.1016/j.patrec.2008.04.013.

Otsu, Nobuyuki (jan. 1979). « A Threshold Selection Method from Gray-Level Histograms ». In : *IEEE Transactions on Systems, Man, and Cybernetics* 9.1. Conference Name : IEEE Transactions on Systems, Man, and Cybernetics, p. 62-66. ISSN : 2168-2909. DOI : 10.1109/TSMC.1979.4310076.

Pasupa, Kitsuchart et Wisuwat Sunhem (oct. 2016). « A comparison between shallow and deep architecture classifiers on small dataset ». In : *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*. 2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE), p. 1-6. DOI : 10.1109/ICITEED.2016.7863293.

Perlman, Zachary E. et al. (12 nov. 2004). « Multidimensional Drug Profiling By Automated Microscopy ». In : *Science* 306.5699. Publisher : American Association for the Advancement of Science Section : Report, p. 1194-1198. ISSN : 0036-8075, 1095-9203. DOI : 10.1126/science.1100709.

Pollard, Thomas D. et William C. Earnshaw (2002). *Cell biology*. Philadelphia : Saunders, xiv, 805 p. ISBN : 0721639976.

Redmon, Joseph et Ali Farhadi (juil. 2017). « YOLO9000 : Better, Faster, Stronger ». In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI : IEEE, p. 6517-6525. ISBN : 978-1-5386-0457-1. DOI : 10.1109/CVPR.2017.690.

Ren, Shaoqing et al. (2015). « Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks ». In : *Advances in Neural Information Processing Systems* 28, p. 91-99.

ROCK Pi (2020). *ROCK Pi - a series of Single Board Computers*. URL : http://rockpi.org/ (visité le 17/11/2020).

Roul, Julien, Otmane Bouchareb et al. (mar. 2017). « OPTIMAL CONTROL FOR MULTIDIMENSIONAL MICROSCOPY ». In : p. 2.

Roul, Julien, Marc Tramier et Jacques Pecreaux (25 juin 2019). « Method for controlling a plurality of functional modules including a multi-wavelength imaging device, and corresponding control system ». Brev. amér. 10330911B2. Universite de Rennes 1 Centre National de la Recherche Scientifique CNRS.

Sanderson, Michael J. et al. (1er oct. 2014). « Fluorescence Microscopy ». In : *Cold Spring Harbor protocols* 2014.10, pdb.top071795. ISSN : 1940-3402. DOI : 10.1101/pdb.top071795.

Sargano, Allah Bux, Plamen Angelov et Zulfiqar Habib (jan. 2017). « A Comprehensive Review on Handcrafted and Learning-Based Action Representation Approaches for Human Activity Recognition ». In : *Applied Sciences* 7.1. Number : 1 Publisher : Multidisciplinary Digital Publishing Institute, p. 110. DOI : 10.3390/app7010110.

Schapire, Robert E. (2013). « Explaining AdaBoost ». In : *Empirical Inference*. Sous la dir. de Bernhard Schölkopf, Zhiyuan Luo et Vladimir Vovk. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 37-52. ISBN : 978-3-642-41136-6. DOI : 10.1007/978-3-642-41136-6_5.

Scherf, Nico et Jan Huisken (août 2015). « The smart and gentle microscope ». In : *Nature Biotechnology* 33.8. Number : 8 Publisher : Nature Publishing Group, p. 815-818. ISSN : 1546-1696. DOI : 10.1038/nbt.3310.

Schindelin, Johannes et al. (juil. 2012). « Fiji : an open-source platform for biological-image analysis ». In : *Nature Methods* 9.7. Number : 7 Publisher : Nature Publishing Group, p. 676-682. ISSN : 1548-7105. DOI : 10.1038/nmeth.2019.

Shaikhina, Torgyn et al. (1er jan. 2015). « Machine Learning for Predictive Modelling based on Small Data in Biomedical Engineering ». In : *IFAC-PapersOnLine*. 9th IFAC Symposium on Biological and Medical Systems BMS 2015 48.20, p. 469-474. ISSN : 2405-8963. DOI : 10.1016/j.ifacol.2015.10.185.

Siddiqui, Fahad et al. (jan. 2019). « FPGA-Based Processor Acceleration for Image Processing Applications ». In : *Journal of Imaging* 5.1. Number : 1 Publisher : Multidisciplinary Digital Publishing Institute, p. 16. DOI : 10.3390/jimaging5010016.

Singh, Shantanu, Anne E. Carpenter et Auguste Genovesio (juin 2014). « Increasing the Content of High-Content Screening : An Overview ». In : *Journal of Biomolecular Screening* 19.5, p. 640-650. ISSN : 1552-454X. DOI : 10.1177/1087057114528537.

Sizaire, Florian et al. (fév. 2020). « Automated screening of AURKA activity based on a genetically encoded FRET biosensor using fluorescence lifetime imaging microscopy ». In : *Methods and Applications in Fluorescence* 8.2. Publisher : IOP Publishing, p. 024006. ISSN : 2050-6120. DOI : 10.1088/2050-6120/ab73f5.

Srivastava, Nitish et al. (1er jan. 2014). « Dropout : a simple way to prevent neural networks from overfitting ». In : *The Journal of Machine Learning Research* 15.1, p. 1929-1958. ISSN : 1532-4435.

Stankovic, J. A. (oct. 1988). « Misconceptions about real-time computing : a serious problem for next-generation systems ». In : *Computer* 21.10. Conference Name : Computer, p. 10-19. ISSN : 1558-0814. DOI : 10.1109/2.7053.

Tamura, Hideyuki, Shunji Mori et Takashi Yamawaki (1978). « Textural Features Corresponding to Visual Perception ». In : *IEEE Transactions on Systems, Man, and Cybernetics* 8.6, p. 460-473. issn : 0018-9472. doi : 10.1109/TSMC.1978.4309999.

Raspberry Pi (2020). *Teach, Learn, and Make with Raspberry Pi.* url : https://www.raspberrypi.org/ (visité le 17/11/2020).

Teague, Michael Reed (1er août 1980). « Image analysis via the general theory of moments ». In : *JOSA* 70.8. Publisher : Optical Society of America, p. 920-930. doi : 10.1364/JOSA.70.000920.

Thomas, Nick (1er jan. 2010). « Review Article : High-Content Screening : A Decade of Evolution ». In : *Journal of Biomolecular Screening* 15.1. Publisher : SAGE Publications Inc STM, p. 1-9. issn : 1087-0571. doi : 10.1177/1087057109353790.

Tischer, Christian et al. (2014). « Adaptive fluorescence microscopy by online feedback image analysis ». In : *Methods in Cell Biology* 123, p. 489-503. issn : 0091-679X. doi : 10.1016/B978-0-12-420138-5.00026-4.

Tsien, Roger Y. (juin 1998). « The Green Fluorescent Protein ». In : *Annual Review of Biochemistry* 67.1, p. 509-544. issn : 0066-4154, 1545-4509. doi : 10.1146/annurev.biochem.67.1.509.

Tuv, Eugene et al. (2009). « Feature Selection with Ensembles, Artificial Variables, and Redundancy Elimination ». In : *Journal of Machine Learning Research* 10.45, p. 1341-1366. issn : 1533-7928.

Voie, A. H., D. H. Burns et F. A. Spelman (1993). « Orthogonal-plane fluorescence optical sectioning : Three-dimensional imaging of macroscopic biological specimens ». In : *Journal of Microscopy* 170.3, p. 229-236. issn : 1365-2818. doi : 10.1111/j.1365-2818.1993.tb03346.x.

Waithe, Dominic et al. (5 oct. 2020). « Object detection networks and augmented reality for cellular detection in fluorescence microscopy ». In : *Journal of Cell Biology* 219.10. Publisher : The Rockefeller University Press. issn : 0021-9525. doi : 10.1083/jcb.201903166.

Wang, Haibo et al. (oct. 2014). « Mitosis detection in breast cancer pathology images by combining handcrafted and convolutional neural network features ». In : *Journal of Medical Imaging* 1.3, p. 034003. issn : 2329-4302, 2329-4310. doi : 10.1117/1.JMI.1.3.034003.

Wang, Hongkai et al. (2018). « Biological image analysis using deep learning-based methods : Literature review ». In : *Digital Medicine* 4.4, p. 157. issn : 2226-8561. doi : 10.4103/digm.digm_16_18.

Watanabe, Wataru et al. (1er juin 2020). « Low-cost multi-modal microscope using Raspberry Pi ». In : *Optik* 212, p. 164713. issn : 0030-4026. doi : 10.1016/j.ijleo.2020.164713.

Xing, F. et al. (oct. 2018). « Deep Learning in Microscopy Image Analysis : A Survey ». In : *IEEE Transactions on Neural Networks and Learning Systems* 29.10. Conference Name : IEEE Transactions on Neural Networks and Learning Systems, p. 4550-4568. issn : 2162-2388. doi : 10.1109/TNNLS.2017.2766168.

YASSIN, Nisreen I. R. et al. (1er mar. 2018). « Machine learning techniques for breast cancer computer aided diagnosis using different image modalities : A systematic review ». In : *Computer Methods and Programs in Biomedicine* 156, p. 25-45. ISSN : 0169-2607. DOI : 10.1016/j.cmpb.2017.12.012.

ZANELLA, Fabian, James B. LORENS et Wolfgang LINK (1er mai 2010). « High content screening : seeing is believing ». In : *Trends in Biotechnology* 28.5, p. 237-245. ISSN : 0167-7799. DOI : 10.1016/j.tibtech.2010.02.005.

Zeiss (2021). *Zeiss : Zeiss ZEN software*. URL : https://www.zeiss.com/microscopy/int/products/microscope-software/zen.html (visité le 29/03/2021).

ZHANG, J. et al. (déc. 2016). « Cancer Cells Detection in Phase-Contrast Microscopy Images Based on Faster R-CNN ». In : *2016 9th International Symposium on Computational Intelligence and Design (ISCID)*. 2016 9th International Symposium on Computational Intelligence and Design (ISCID). T. 1. ISSN : 2473-3547, p. 363-367. DOI : 10.1109/ISCID.2016.1090.

ZHANG, Jin et al. (déc. 2002). « Creating new fluorescent probes for cell biology ». In : *Nature Reviews Molecular Cell Biology* 3.12. Number : 12 Publisher : Nature Publishing Group, p. 906-918. ISSN : 1471-0080. DOI : 10.1038/nrm976.

ZHANG, Shugang et al. (2017). « A Review on Human Activity Recognition Using Vision-Based Method ». In : *Journal of Healthcare Engineering* 2017, p. 1-31. ISSN : 2040-2295, 2040-2309. DOI : 10.1155/2017/3090343.

ZHAO, Z., R. ANAND et M. WANG (oct. 2019). « Maximum Relevance and Minimum Redundancy Feature Selection Methods for a Marketing Machine Learning Platform ». In : *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA), p. 442-452. DOI : 10.1109/DSAA.2019.00059.

UNIVERSITÉ DE RENNES 1

Titre : Conception d'un prototype de microscope intelligent et autonome

Mots-clés : Système embarqué, Rétroaction, Temps-réel, Analyse d'images, Microscopie, Biologie cellulaire

Résumé : Les systèmes de microscopie optique sont des outils complexes, indispensables à la compréhension du vivant par la recherche fondamentale et appliquée en biologie, dont l'automatisation est un champ d'étude en plein essor. Nous présentons dans cette thèse la conception d'un prototype de système embarqué analysant des images de microscopie en temps-réel et réalisant une boucle de rétroaction avec un microscope. Sa confrontation avec des applications biologiques théoriques nous a permis d'obtenir un modèle théorique générique et modulaire, dont nous avons entamé le test en conditions réelles. Il permet de modifier des modalités d'acquisition d'images d'un microscope en fonction des images analysées à la volée. De plus, il effectue un tri de ces images en ne retournant que celles représentant des objets d'intérêt pour les biologistes. L'analyse des images en temps-réel, nécessitant des classifications précises et rapides. Nous proposons une méthode d'optimisation d'un outil générique de classification d'images, dont nous identifions les caractéristiques pertinentes avec un faible temps d'extraction. Nous avons mis en évidence une redondance des caractéristiques qui permet d'exclure les plus longues à calculer même si ce sont les plus discriminantes. Ainsi, une dizaine de caractéristiques permet de classer 14 cellules par seconde avec une précision supérieure à 80 % avec une *random forest* ou un réseau de neurones. Ces travaux ouvrent des perspectives d'optimisation des systèmes de classification en apprentissage automatique, y compris profond. En conclusion, nous avons mis en place les bases d'un microscope intelligent et autonome, ouvrant la voie à une nouvelle génération de microscopie, contribuant à l'émergence d'une médecine de précision.

Title: Design of a smart and autonomous microscope prototype

Keywords: Embedded system, Servo-control, Real-time, Image analysis, Microscopy, Cell biology

Abstract: Optical microscopy systems are complex tools, essential to investigate the living through fundamental and applied research. Automating them is a vibrant field. We report in this PhD dissertation the design and development of a prototype of an embedded system performing real-time image analysis and feeding back to microscope control. Confronting it to different kinds of theoretical biological applications, we obtain a generic and modular functional solution. We already started testing in real conditions. In particular, our system can modify the microscope settings in real-time, depending upon simulated acquired images. Furthermore, it sorts out these images, returning only the ones relevant for biology. Key to succeeding in this approach is real-time image processing. We offer a way to optimize a generic image classification, identifying the features relevant for classification and which extracting is not computationally intensive. We highlighted features redundancy, which allowed us to exclude the most time-consuming one despite their high relevance. Thus, about ten so-selected features are enough to classify 14 cells per second with an accuracy greater than 80%, using random forest or a neural network. This work opens perspectives in optimizing machine learning methods including deep learning. In conclusion, we developed the foundation of a smart and autonomous microscope, paving the way to a new microscopy generation. It contributes to rising of high precision medicine based on microscopy.