University of Liège

Faculty of Applied Sciences
Department of Electrical Engineering & Computer Science

PhD dissertation

Addressing data scarcity with deep transfer learning and self-training in digital pathology

by Romain Mormont

Advisors: Pierre Geurts & Raphaël Marée
September 2022

Jury

PIERRE GEURTS (advisor), Professor at the University of Liège (BE)

RAPHAËL MARÉE (advisor), Senior Researcher at the University of Liège (BE)

GILLES LOUPPE (president), Professor at the University of Liége (BE)

LOUIS WEHENKEL, Professor at the University of Liège (BE)

MARC VAN DROOGENBROECK, Professor at the University of Liège (BE)

CHRISTINE DECAESTECKER, Professor at the Université Libre de Bruxelles (BE)

FRANCESCO CIOMPI, Associate Professor at the Radboud University Medical Center (NL)

Acknowledgements

Un doctorat est une aventure parsemée de joies, de peines et d'obstacles. Cette aventure, je n'en aurais jamais vu le bout sans bon nombre de personnes que j'aimerais remercier.

Mes premiers remerciements vont évidemment à mes promoteurs, Pierre et Raphaël. Lorsqu'en 2015, j'ai contacté Pierre à la recherche d'un sujet de TFE en "machine learning sur des images" et qu'il m'a mis en contact avec Raphaël, je n'aurais jamais osé m'imaginer que cela m'amènerait environ 7 ans plus tard à défendre une thèse de doctorat. Je me souviens encore très bien de la première réunion avec Raphaël à l'extérieur de Montefiore et l'introduction à Cytomine puis les réunions de suivi avec le duo au GIGA. Ces premiers échanges furent aussi enrichissants qu'agréables.

Je me souviens également de la discussion avec Pierre au cours de laquelle il m'a proposé d'entamer une thèse. Si le challenge avait attisé ma curiosité, ce sont aussi les qualités humaines de mes futurs promoteurs qui m'ont convaincues d'accepter la proposition. On entend souvent parler de doctorats qui tournent mal à cause d'une relation doctorant-promoteur compliquée, voire toxique. Dans mon cas, ce fut tout le contraire. J'ai eu l'infinie chance de tomber sur deux personnes bienveillantes, encourageantes, curieuses de ce que j'avais à proposer, toujours prêtes à accorder du temps pour une réunion ou une discussion. Au-delà des aspects humains, leur apport scientifique, technique et leur complémentarité dans ces deux domaines abordés dans ma thèse m'ont évidemment grandement aidé et m'ont permis d'apprendre énormément. Pierre, Raphaël, je ne vous remercierai probablement jamais assez d'avoir cru en moi et de m'avoir permis d'arriver où je suis maintenant!

J'aimerais aussi remercier tous les collègues que j'ai eu la chance de côtoyer au long de ces six années. Que ça soit en temps de midi, en LAB Meeting, en conférence ou autres, votre présence et nos discussions étaient toujours bien agréables. Vous avez contribué à me faire aimer mon travail ! Un grand merci à Laurine, Jean-Michel, Ulysse, Vân Anh, Pierre, Raphaël, Michaël, Antho, Navdeep, Marie, Antoine, Pascal L., Matthia, Nicolas, Rémy, Pascal F., et encore plein d'autres que j'oublie fort probablement (désolé!).

Je souhaite également remercier les membres du jury internes, Louis et Marc, et externes, Christine Decaestecker et Francesco Ciompi, qui ont pris le temps de lire mon travail en détail et m'ont fourni un feedback précieux. Merci également à Laurine, Fabrice, Michel, Ulysse et Jean-Michel qui ont relu ce manuscrit avant soumission.

Ce travail n'aurait jamais été possible sans l'accès à des ressources de calcul et j'aimerais également remercier toutes les personnes m'ont permis de près ou de loin d'y avoir accès. Remerciements, donc, à tous les professeurs et équipes de Montefiore qui ont financé le cluster GPU et remerciements particuliers à Joeri qui a mis en place l'infrastructure.

Amis et famille, je ne vous oublie pas ! Durant ces six années remplies de hauts et de bas, au fond de moi, je n'ai jamais perdu cette petite flamme alimentée par votre amour, amitié et soutien. Quoiqu'il se passe, quelle que soit l'épreuve, je sais que je peux compter sur vous et mine de rien, ça fait toute la différence.

À ma famille, Papa, Maman et Odile en tête mais aussi tous les autres, merci pour votre présence et votre soutien inconditionnel. Un remerciement tout particulier à Babone et Papy Linlin de m'avoir accueilli tous les vendredis pendant presque un an. Chez vous, j'ai trouvé un havre de quiétude et de concentration qui m'ont permis d'avancer significativement dans la rédaction de ce manuscrit. Un grand merci à toi aussi Florence, qui m'a rejoint sur la fin du parcours pour ton soutien et ton amour pendant la finalisation du travail.

Les amis, ensuite, que vous soyez présent physiquement ou derrière un écran et un micro, vous m'avez aidé plus que vous ne le pensez dans les périodes difficiles (pandémie, je te vois). Les Componions évidemment (Fabrice, Chloé, Élo, Arnaud, Laurine, Quentin, Hubert), merci pour votre présence, merci d'être ce que vous êtes et de ce que nous devenons. Floriane, merci pour tes visites lors de passages à Montefiore et nos discussions techniques ou non. Michel, merci de m'avoir fait visiter le service d'anatomopathologie du CHU de Liège. Sans toi, mon chapitre 3 ne serait pas ce qu'il est. Les colocs en commençant par Charly, Mirabelle et Nicolas. Que ce fut enrichissant d'emménager et de vivre ensemble. Je n'oublie pas ceux qui ont suivi aussi: Hubert, Michaël, Jérôme, Jiyeon. L'amitié n'est pas toujours là où on l'attend et peut parfois se développer par écran interposé: je tiens à vous remercier les Bloude et en particulier la team "Off" pour tous les bons moments passés ensemble en ligne qui m'ont grandement aidé à passer le cap de la pandémie.

J'aimerais clôturer ces remerciements avec une mention toute particulière pour toi, Nonno, qui nous a quittés voilà maintenant 11 ans. Tu avais 25 ans et ne savais ni lire ni écrire quand tu as quitté ton Italie natale pour trouver un travail afin de demander la main de Nonna. Tu as échangé le soleil et la chaleur des Pouilles pour le ciel gris belge, la pénombre et la poussière des mines. Ce sont les combats que vous avez menés avec Nonna pour permettre à vos enfants de vivre une vie décente qui m'ont permis d'obtenir le titre de Docteur aujourd'hui. C'est pour cela que je te dédie cette réussite.

Contents

Abstract

Pathology, the field of medicine and biology interested in studying and diagnosing diseases, is on the brink of a revolution with technological advances in artificial intelligence and machine learning. Traditionally, in this field, the medium which has been used for research and diagnosis is a glass slide on which tissue and cell samples are applied and later analyzed under an optical microscope. Dedicated scanners are nowadays able to digitize these glass slides into large digital images called whole-slide images which can then be reviewed on a computer. This new medium also offers unprecedented opportunities for computers to assist practitioners by automating the most time-consuming and tedious analysis tasks. The field which is interested in these digitization, automation and related topics is called digital pathology.

Machine and deep learning methods are great candidates for tackling these automation tasks thanks to their ability to automatically learn models and capture complex patterns directly from data. However, digital pathology presents several challenges for learning methods. In particular, the field is suffering from data scarcity as data, especially annotated, is difficult to obtain because of privacy concerns, cost of annotations, *etc*.

In this thesis, we explore different machine learning techniques tailored for tackling data scarcity. We first study different deep transfer learning techniques, a family of methods which consist in re-using a model that has been learned on a different task than the target task. We investigate best practices regarding how deep convolutional neural network models pre-trained on ImageNet, a dataset of photographs, can be transferred to digital pathology image classification tasks. We notably show that, in digital pathology, fine-tuning outperforms feature extraction and draw other practical conclusions regarding transfer from ImageNet. Motivated by the fact that transfer performs better when the source and target tasks are close, we then use multi-task learning to pre-train a model on pathology data directly. We show that this technique is efficient for creating a transferrable model tailored for pathology tasks. Finally, we move to the topic of self-training, a family of methods where a model being learned is used to annotate unlabeled data that is then incorporated into the training process. In particular, we apply this technique to image segmentation for exploiting a dataset which has been only sparsely-labeled. We show that our approach is able to make use of the sparsely-labeled data better than a supervised approach.

Acronyms

ACC	accuracy.
AFDS	attentive feature distillation and selection.
AI	artificial intelligence.
API	application programming interface.
AUC	area under the curve.
BCE	binary cross-entropy.
BN	batch normalization.
CLF	classification.
CNN	convolutional neural network.
CNT	counting.
COCO	Common Objects in COntext.
CPATH	computational pathology.
CPU	central processing unit.
CRF	conditional random field.
CT	computed tomography.
CV	cross-validation.
DET	detection.
DL	deep learning.
DP	digital pathology.
ERM	empirical risk minimization.
ET	extremely randomized trees.
ET-DIC	extremely randomized trees as direct-learner.
ET-FL	extremely randomized trees as feature-learner.
FC	fully connected.
FCN	fully-convolutional network.
FNAB	fine-needle aspiration biopsy.
FPR	false positive rate.
GAN	generative adversarial networks.

GDPR	General Data Protection Regulation.
GLAS	Gland Segmentation challenge.
GPU	graphical processing unit.
H&E	hematoxylin & eosin.
HOG	histogram of oriented gradient.
IHC	immunohistochemistry.
ILSVRC	ImageNet Large Scale Visual Recognition Challenge.
LIS	laboratory information systems.
LOTO	leave-one-task-out.
ML	machine learning.
MLP	multi-layer perceptron.
MONUSEG	multi-organ nuclei segmentation challenge.
MTL	multi-task learning.
NLP	natural language processing.
OVO	one-vs-one.
OVR	one-vs-rest.
PACS	picture archiving and communication system.
RELU	rectified linear unit.
RFE	recursive feature elimination.
ROC	receiver operating characteristic.
ROI	region of interest.
SEG	segmentation.
SEGPC-2021	segmentation of multiple myeloma plasma cells in microscopic images.
SGD	stochastic gradient descent.
SL	supervised learning.
SOTA	state-of-the-art.
SSL	semi-supervised learning.
SVM	support-vector machine.
TCGA	The Cancer Genome Atlas.
TL	transfer learning.
TPR	true positive rate.

ULB	Université Libre de Bruxelles.
USL	unsupervised learning.
ViT	vision transformer.
WHO	world health organization.
WSI	whole-slide image.
WSL	weakly-supervised learning.

1 Chapter Introduction

1.1 Context

Machine learning (ML), the sub-field of *artificial intelligence* (AI) which is concerned with how to make a computer learn from data, has been through quite a journey since its inception in the 1940s and 1950s. From a small research domain, it has grown into a massive rapidly-developing field of research and applications with ramifications in many branches of science and technology. This growth is not surprising in a world where computers become more and more powerful and data, the bread-and-butter of machine learning, increasingly structured and queryable. Applications of machine learning are as varied as they are numerous: spam filtering, fraud detection, face recognition, self-driving cars, robotics and automation, medical data analysis and diagnosis, simulation in particle physics... to name only a few. While it has already revolutionized many domains, machine learning research has still a bright future ahead. In the last decade only, several new families of techniques have been (re-)discovered and have enabled unexpectedly fast progress (*e.g.* deep learning, convolutional neural network, generative adversarial networks, transformers). One can only expect that the next ground-breaking ML method is on the brink of being discovered in a research lab somewhere around the world. Aside from that, research is still ongoing on many fronts such as understanding, applying or improving existing methods.

Medicine is among the numerous fields where machine learning is showing great promises. Although often misrepresented in the mainstream media as a tool that will eventually replace practitioners, the real potential of machine learning in medicine actually lies in its capacity to become a strong, resilient and consistent assistant to the physicians [159], assisting them for tasks ranging from diagnosis and analysis to paperwork. Rather than replacing physicians, an AI-based diagnosis system would be able to complement their opinion and advise them based on experiences of millions of other patients and colleagues. Moreover, such a system would be able to produce these advice based on a very large number of parameters and sources of data that a human could not realistically consider (imaging, written reports, laboratory values, vital signs). However, the road to an AI-assistant is still long and many questions and challenges have yet to be addressed. From a scientific standpoint, current research mostly focuses on improving solutions for tasks of significantly smaller scale and scope (*e.g.* outlining organs in x-rays, detecting disease in CT-scans, classifying skin cancer as malignant or benign). Many recent contributions, some of which the generalization can be questioned [146], have claimed to have matched or surpassed

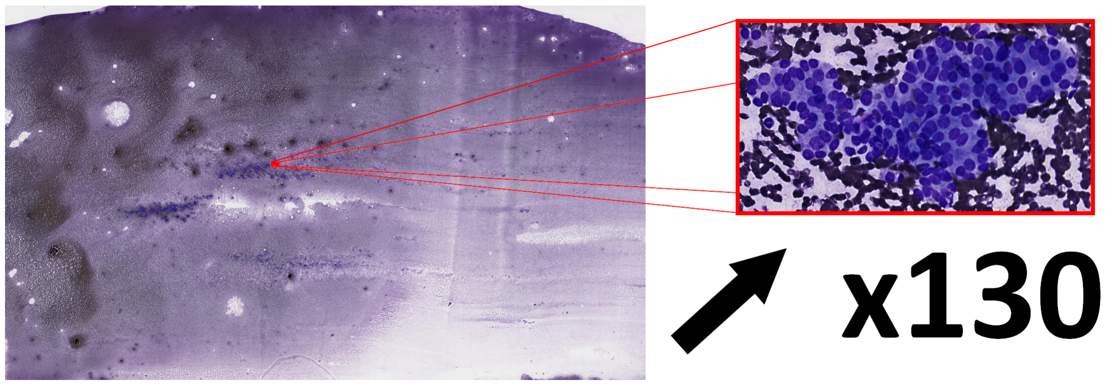


FIGURE 1.1: A typical whole-slide image of size 163840 × 95744 pixels and file size of 2.3 gigabytes. To the left is the original slide and to the right is a structure of interest at zoom level ×130 (size in pixels: 1354 × 736, source: ULB, Erasme).

human experts accuracies by applying machine learning methods on different medical tasks. Whereas progress is certain, many challenges have yet to be solved.

One of these major challenges is *data scarcity*. Not that data itself is lacking as Pramanik, Pal, and Mukhopadhyay [157] reported that the amount of health data stored worldwide could reach 2,258 exabytes in 2020. What is scarce is actually data that is at least partially annotated and of good-enough quality to be used to train machine learning models. Data scarcity has many causes: privacy concerns prevent sharing patient data, the annotation process is time-consuming and expensive, *etc*. This is aggravated by the fact that recent machine learning methods (*i.e.* deep learning) require a significant amount of data to perform optimally.

In this thesis, we are interested in one specific field of medicine: *digital pathology* (DP). Microscope have been used in medicine to analyze samples (*e.g.* cells, tissue or blood) since the 17th century [74] but recent technological progress has enabled the digitization of microscope glass slides into images called whole-slide images (WSI). Digital pathology concerns all the aspects of acquiring, managing, sharing and interpreting these WSIs [50]. With the help of image management systems, modern visualization tools (*e.g.* Cytomine [132], `uliege.cytomine.org`) and image analysis algorithms, WSIs have the potential to revolutionize the daily work of pathologists and biologists but the transition process is complex and expensive. Therefore, only a few hospitals and facilities took the leap towards a full digital environment (*e.g.* [191, 53, 207]). Among the greatest promises of digital pathology is the possibility to automate diagnosis tasks with computer vision techniques including machine learning [42]. Such automation would not only accelerate the diagnosis process but could also improve its accuracy by delegating time-consuming and tedious but simple tasks to algorithms (*e.g.* counting cells, measuring area of tumor) therefore allowing practitionners to focus on the challenging aspects of their work and ultimately improving patient care and treatments. The use of such techniques would also contribute to accelerating drug and pathology research relying on whole-slide image analysis.

The subfield of digital pathology focusing on automating the analysis of pathology data is called *computational pathology* (CPATH) and faces many interesting challenges. A WSI is typically a very large image that can reach few billion pixels at maximum zoom level (see Figure 1.1, a single file can weight several gigabytes) therefore making classical computer vision methods inapplicable without adaptation. The image content is usually complex which excludes the use of simplistic computer vision methods. This explains why machine learning is currently so popular among computational pathology researchers because of its ability to cope with the complexity by automatically learning from data. However, efficient use of machine learning is hampered by data scarcity of which the field is not spared since billion-pixels images are as tedious to analyze as to annotate exhaustively, even for trained pathologists. The data scarcity problem is worsened by the variability introduced by the transformation process of a biological sample into a slide and then into a whole-slide image.

1.2 Contributions and outline

In this thesis, we explore different ways of tackling data scarcity in digital pathology using deep learning, a sub-field of machine learning which focuses on deep neural networks. Our contributions leverage transfer learning (TL), multi-task learning (MTL) and self-training which we apply to the tasks of image classification and segmentation. The manuscript is structured in three parts which are preceded by this introduction.

Part I introduces machine learning and digital pathology. Chapter 2 focuses on machine learning and presents the different concepts and techniques used throughout this thesis. It is not intended to explain the matter in depth but rather to give sufficient background for a reader with a basic knowledge of machine learning to understand the contributions. This chapter also presents works related to ours. More precisely, we discuss how similar techniques are used for general purpose problems (*i.e.* not specific to medical imaging). Chapter 3 focuses on digital pathology and presents the field from the perspective of a computer scientist. It presents how a biological sample is transformed and ultimately becomes a whole-slide image in order to explain how this crucial preparation step can impact the image analysis down the line. We present three example diagnosis tasks that would greatly benefit from automation with computer vision techniques. We also introduce the challenge that is data scarcity more in depth and present works related to ours. Unlike in Chapter 2, the related works are focused on medical and pathology application of methods similar to ours.

Part II presents our first and second contributions which are both related to transfer learning. In Chapter 4, we first review, compare and study different deep transfer learning techniques using 8 classification datasets in order to evaluate the viability and best practices of transfer learning in digital pathology. We confirm the current scientific consensus that using neural networks pre-trained on a dataset unrelated to digital pathology (*i.e.* ImageNet [48]) is indeed interesting in terms of performance. We draw guidelines from our experiments on how the transfer should be performed. In Chapter 5, we then propose a multi-task learning architecture and training scheme for pre-training a network based on an ensemble of potentially-small datasets rather than a single large dataset. We use these in order to pre-train a neural network on 22 classification digital pathology datasets and show that our resulting models yield competitive results compared to networks pre-trained on ImageNet.

Part III and Chapter 6 present our third contribution. We use a machine learning technique called self-training to train an image segmentation model on sparsely-labeled data. With our approach, the machine learning model is trained and then used to predict missing annotations in the dataset. We show that our approach is able to make efficient use of the sparsely-labeled data to improve the segmentation performance. Moreover, we show that it is not always necessary to exhaustively label a dataset to obtain competitive performance.

The manuscript ends with Chapter 7 which presents our conclusions and discussion of future works.

1.3 Publications

This thesis is based on the following publications, considered respectively in Chapters 4, 5 and 6:

- Romain Mormont, Pierre Geurts, and Raphaël Marée. "Comparison of deep transfer learning strategies for digital pathology". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 2262–2271
- Romain Mormont, Pierre Geurts, and Raphaël Marée. "Multi-task pre-training of deep neural networks for digital pathology". In: *IEEE journal of biomedical and health informatics* 25.2 (2020), pp. 412–421
- Romain Mormont, Mehdi Testouri, Raphaël Marée, and Pierre Geurts. "Relieving pixel-wise labeling effort for pathology image segmentation with self-training". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. Accepted for publication. 2022

During the course of my PhD, I have contributed to the development of Biaflows (`biaflows-doc.neubias.org`), a platform built on top of Cytomine designed for standardized benchmarking of bioimage analysis workflows. This work resulted in the following publication:

- Ulysse Rubens*, Romain Mormont*, et al. "BIAFLOWS: A collaborative framework to reproducibly deploy and benchmark bioimage analysis workflows". In: *Patterns* 1.3 (2020), p. 100040 (* these authors contributed equally).

Although I consider this work as a significant contribution of my PhD, it will not be discussed in this manuscript as it does not fit into our narrative around data scarcity in digital pathology. The abstract of this article is however provided in Appendix E, together with a link to the full paper.

Throughout my PhD, I have frequently used and updated the SLDC framework which was originally created in the context of my master thesis:

- Romain Mormont, Jean-Michel Begon, Renaud Hoyoux, and Raphaël Marée. "SLDC: an open-source workflow for object detection in multi-gigapixel images". In: *The 25th Belgian-Dutch Conference on Machine Learning (Benelearn)*. 2016

1.4 Code and models

The code and models for our publications are also publicly available online with permissive licenses:

- Chapter 5: `https://github.com/waliens/multitask-dipath` (code and models)
- Chapter 6: coming soon (code)
- Biaflows: `https://github.com/Neubias-WG5`
- SLDC: `https://github.com/waliens/sldc`

Part I

Background

Chapter

2 Machine learning

Overview

The goal of this chapter is to provide machine learning background and keys to understand our contributions. It is not aimed at being an exhaustive tour of the field of machine learning but rather an overview of topics relevant to this thesis. For the readers who would still like to deepen their knowledge about these methods, we provide pointers to relevant literature. In this chapter, we also introduce the notations that we use throughout this thesis.
Section 2.1 provides a short definition of machine learning and a first example of an ML problem. Section 2.2 explores different ways the field of machine learning can be structured (*e.g.* supervised vs. unsupervised learning, classification vs. regression, classical machine vs. deep learning...) in order to position our work in its context. Section 2.3 discusses what guides machine learning model training from a theoretical point of view. Then, it presents practical concerns regarding model selection and evaluation. It finally provides a description of different evaluation metrics used in this thesis. This chapter then shifts its focus on specific machine learning methods and algorithms: support-vector machine in Section 2.4, tree-based methods in 2.5 and deep learning in Section 2.6 in which we also discuss more thoroughly deep transfer learning (see Section 2.6.4). In the final Section 2.7, we wrap up by positioning our work in the context of machine learning.

2.1 What is machine learning ?

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E [140]. Machine learning concerns the study of such programs, commonly referred to as *models*, and how to build them by learning. A *model* h can be seen as a function taking an input $\mathbf{x} \in \mathcal{X}$ (*a.k.a.* observation, example, instance) and producing an ouput $h(\mathbf{x})$ or $\hat{y} \in \mathcal{Y}$. Entities $\mathbf{x}$ and $h(\mathbf{x})$ are n-dimensional tensors and can encode many kind of data types: data record, image, graph, time series, text... When $\mathbf{x}$ is a vector, its components are commonly called *features*, *variables* or *attributes*. A model can have several inputs (*resp.* outputs) in which case $\mathbf{x}$ (*resp.* y) is a tuple.

In machine learning, a model is built by a *learning algorithm*. Formally, a learning algorithm is defined by a set $\mathcal{H}$ of candidate models called the *hypothesis space*, a performance measure P for a model and an optimization strategy. As input, the algorithm is provided with *training data* (the experience E, *a.k.a.* learning or training set) that it uses to build and optimize the model. The algorithm output is a model $h \in \mathcal{H}$ that maximizes the performance criterion. A model being built by a learning algorithm is said to be in the *training phase*. When this model has been trained and is used on new data, it is said to be in the *inference phase*.

As an example, a common task is *natural image classification* where the model must assign a label to a picture. For instance, one would want to detect whether a picture contains a human, an animal or an inanimate object. In this case, considering that the images are encoded with integers, the input space is the set of all possible color images: $\mathcal{X} \subset \mathbb{N}^{r \times c \times b}$ where r, c and b are respectively the image width, height and number of channels (which equals to 3 in the case of RGB color images). The output space is composed of the 3 labels of interest: $\mathcal{Y} = \{human, animal, object\}$. The model would take an image $\mathbf{x} \in \mathcal{X}$ as input and, based on its content, output $\hat{y}$, one of the predefined labels. This label $\hat{y}$ might be false if the model makes a mistake. Therefore, for the sake of distinction, the correct label is denoted y (*a.k.a.* ground truth). As a performance measure P, one could assess the correctness of the ouput label by assigning 0 to correct predictions and 1 to errors. This performance measure is called the *zero-one loss* and is written as:

$$\ell_{0-1}(y, \hat{y}) = \mathbb{1}_{y \neq \hat{y}} \tag{2.1}$$

There are numerous tasks beyond natural image classification to which machine learning can be applied nowadays. In the next sections, we will discuss some of them and dive a little deeper into algorithms and topics related to learning which are relevant to this thesis.

2.2 Families of learning methods

There are many ways to structure the ecosystem of machine learning methods. This section explores some of them.

2.2.1 Supervised learning

Supervised learning (SL) regroups methods where the learning process is guided by an output signal. We formalize a supervised task as the tuple $(\mathcal{X}; \mathcal{Y}; p(\mathbf{x}, y))$ where $\mathcal{X}$ and $\mathcal{Y}$ are respectively the input and output spaces and $p(\mathbf{x}, y)$ is a probability distribution over those joint spaces. The learning algorithm is provided with a training set

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, ..., n; (\mathbf{x}_i, y_i) \sim p(\mathbf{x}, y)\} \tag{2.2}$$

where $y_i \in \mathcal{Y}$ is the output signal for the observation $\mathbf{x}_i \in \mathcal{X}$. The learning algorithm objective is to find a model $h \in \mathcal{H}$ that approximates at best the output. In general, one wants to train a model that minimizes the generalization error (see Equation 2.9 and Section 2.3.1 for more details) or, alternatively, that performs well on unseen data by predicting the most appropriate output.

When the output space is a finite set of discrete values $\mathcal{Y} = \{v_1, v_2, ..., v_C\}$, the problem is called *classification*. The v_i elements are called the classes (or label) and C denotes the cardinality of the set $\mathcal{Y}$ (*i.e.* the number of classes). When $C = 2$, the problem is said to be *binary*. In classification, in order to minimize the zero one loss for instance, the model should predict the most probable output class given an example:

$$\hat{y}_i = \arg\max_{y_k \in \mathcal{Y}} p(y = y_k | \mathbf{x} = \mathbf{x}_i). \tag{2.3}$$

Examples of classification tasks are for instance assigning a label to an image (*i.e.* the example presented in Section 2.1) or detecting whether an email is a spam or not.

When the output space is continuous and the output is a real scalar value, the problem is called *regression*. In this case, in order to minimize the squared loss

$$\ell_{squared}(y, \hat{y}) = (y - \hat{y})^2 \tag{2.4}$$

for instance, the model should predict the expected value y given any input $\mathbf{x}_i$:

$$h(\mathbf{x}_i) = \mathbb{E}\left[y | \mathbf{x} = \mathbf{x}_i\right]. \tag{2.5}$$

Examples of regression problems are trying to predict the price of a house given its area, the amount of product generated by a factory in a given time period or the review score of a product on an e-commerce platform.

Some models can also predict structured outputs. This is the case with *image segmentation* which focuses on classifying each pixel of an image (*i.e.* answering the question: what kind of object does this pixel belong to in the image?). The output of the model is a segmentation mask where pixel at row i and column j of the image is classified as $\hat{y}_{ij} \in \mathcal{Y}$. For some tasks, a mask is not always necessary, one is rather interested in the coarse location of the objects. This kind of task is called *detection* for which the output of the model, the location, can be encoded as image coordinates (i, j) representing the object's center of gravity or any of its points. Another common representation is the bounding box, a box containing exactly the object of interest, encoded by the position of a corner of the box in the image and its height and width.

In supervised learning, the training signal is often created by humans manually annotating examples. Such guidance allows using well-studied methods and usually helps learning strong models but also comes at the cost of human intervention. This is especially aggravated when the target task is difficult as, the more complex, the more data is required to sample sufficiently the input space. In some domains, annotations are particularly cumbersome to obtain because one lacks raw data, or because the annotation process requires the intevention of experts (*e.g.* medical data). This issue is referred to as *data scarcity*. In general, the lack of data hampers successful application of supervised learning but several approaches exist to work it around which are presented briefly in Sections 2.2.5 and 2.2.6.

2.2.2 Unsupervised learning

In opposition to supervised learning, *unsupervised learning* (USL) regroups methods where no output signal is provided to guide the learning process. The learning algorithm is provided with

$$\mathcal{D} = \{\mathbf{x}_i \mid i = 1, ..., n; \mathbf{x}_i \sim p(\mathbf{x})\} \tag{2.6}$$

and attempts to extract information from this dataset. A common unsupervised task is *density estimation* where the goal is to model the generating distribution $p(\mathbf{x})$ but there also exist other types of methods. With *clustering*, for instance, the algorithm searches for natural groups of observations or features. Another example is *dimensionality reduction* where the algorithm projects high-dimensional data into a lower-dimensional space while attempting to preserve as much information as possible. An interested reader will find more information about these methods in [76].

There exists another family of unsupervised learning methods that was first explored in the 1980s with autoencoders [13, 113, 26] but has gained much traction recently. It is called *self-supervised learning* [114]. The idea behind this family of methods is to exploit supervised learning algorithms but rather than guiding the learning process with human annotations, the training signal is found in the data itself. An example of such methods is *image reconstruction*. Random parts of the input images are truncated (*e.g.* replaced by black squares) and the model must be able to re-generate the truncated parts. In this case, input and output signals are respectively the truncated and the original image. The model input can be generated from the original data without human intervention.

2.2.3 Between supervised and unsupervised learning

The families discussed in Sections 2.2.1 and 2.2.2 do not cover all existing machine learning methods but are rather at the ends of a spectrum. There also exist intermediate families of methods. Halfway between supervised and unsupervised learning is *semi-supervised learning* (SSL) which focuses on methods that use a dataset where only a part of the observations have an associated output signal. The dataset is composed of two subsets:

$$\mathcal{D}_l = \{(\mathbf{x}_i, y_i) \mid i = 1, ..., n_a; y_i \sim p\left(y|x_i\right); \mathbf{x}_i \sim p_a(\mathbf{x})\} \tag{2.7}$$

$$\mathcal{D}_u = \{\mathbf{x}_i \mid i = 1, ..., n_u; \mathbf{x}_i \sim p_u(\mathbf{x})\} \tag{2.8}$$

These methods make some assumptions about the "closeness" of the input distributions $p_a(\mathbf{x})$ and $p_u(\mathbf{x})$ [36] which allow exploiting both sets to solve some particular tasks. One of the earliest forms of semi-supervised learning is *self-training* [178, 226] which consists in an iterative process of which a typical iteration, or round, is composed of two steps. First, the *teacher* model is used to generate pseudo-labels for unlabeled samples from $\mathcal{D}_u$. Then, these samples are used alongside samples from $\mathcal{D}_l$ to train a model, the *student*. We will discuss self-training more thoroughly in Section 2.6.6. Self-training is not to be confused with self-supervised learning (discussed in the previous section) as the latter involves finding the training signal in the input data rather than producing pseudo-labels for unlabeled data with the model being trained.

Closer to supervised learning is *weakly-supervised learning* (WSL) which focuses on methods where the annotations are noisy (*e.g.* y can be incorrect or imprecise), incomplete (similar to unsupervised learning) or coarse. With coarse annotations, the model must produce more information than contained in the training signal y. Coming back to our example of Section 2.1, a weakly-supervised problem would consist in locating the human, animal or object in the image using only the class as the coarse training signal.

2.2.4 Shallow versus deep learning

Many things in our world can be viewed as hierarchies of concepts. For instance, a human body is composed of body parts like the head. The head itself includes the face which is itself composed of several elements: cheeks, eyes, nose, *etc*. This kind of decomposition could also be applied to other concepts. Grasping hierarchies is key for understanding and learning efficiently. As humans, our visual cortex process information in a hierarchical manner [209]. Similarly, this can be applied to computer vision. An image can be seen as a hierarchy going from the actual objects it contains down to the pixels. Direct interpretation of individual pixels is rarely enough for learning anything meaningful. However, pixels combined together make edges, which themselves make textures, which are eventually combined in several rounds to reach meaningful semantic elements (see Figure 2.1). Therefore, being able to somewhat exploit these inherent hierarchies is relevant to achieve image understanding by learning.

Most of the machine learning methods developed until recently can arguably be considered "*shallow*" which means that they are not complex enough or their learning process does not include a mechanism to learn or exploit such hierarchies. In order to successfully apply these methods to complex data with hierarchical structures, one usually needs to help the learning algorithm by pre-processing the data and extracting meaningful information using field knowledge. This process is called *manual feature extraction* or *engineering* and has been an important part of the application of machine learning algorithms. In computer vision, a great body of work has been focused on creating complicated pipelines of feature extraction that produce hundreds of different features that can be used for image understanding (*e.g.* SURF [17], ORB [173]). However, more recently "deep learning" methods based on neural networks have shown that manual feature extraction was usually not the best performing approach for a wide variety of tasks.

Although neural network research is as old as machine learning itself, the real breakthrough of deep learning happened in 2012 at the occasion of the third iteration

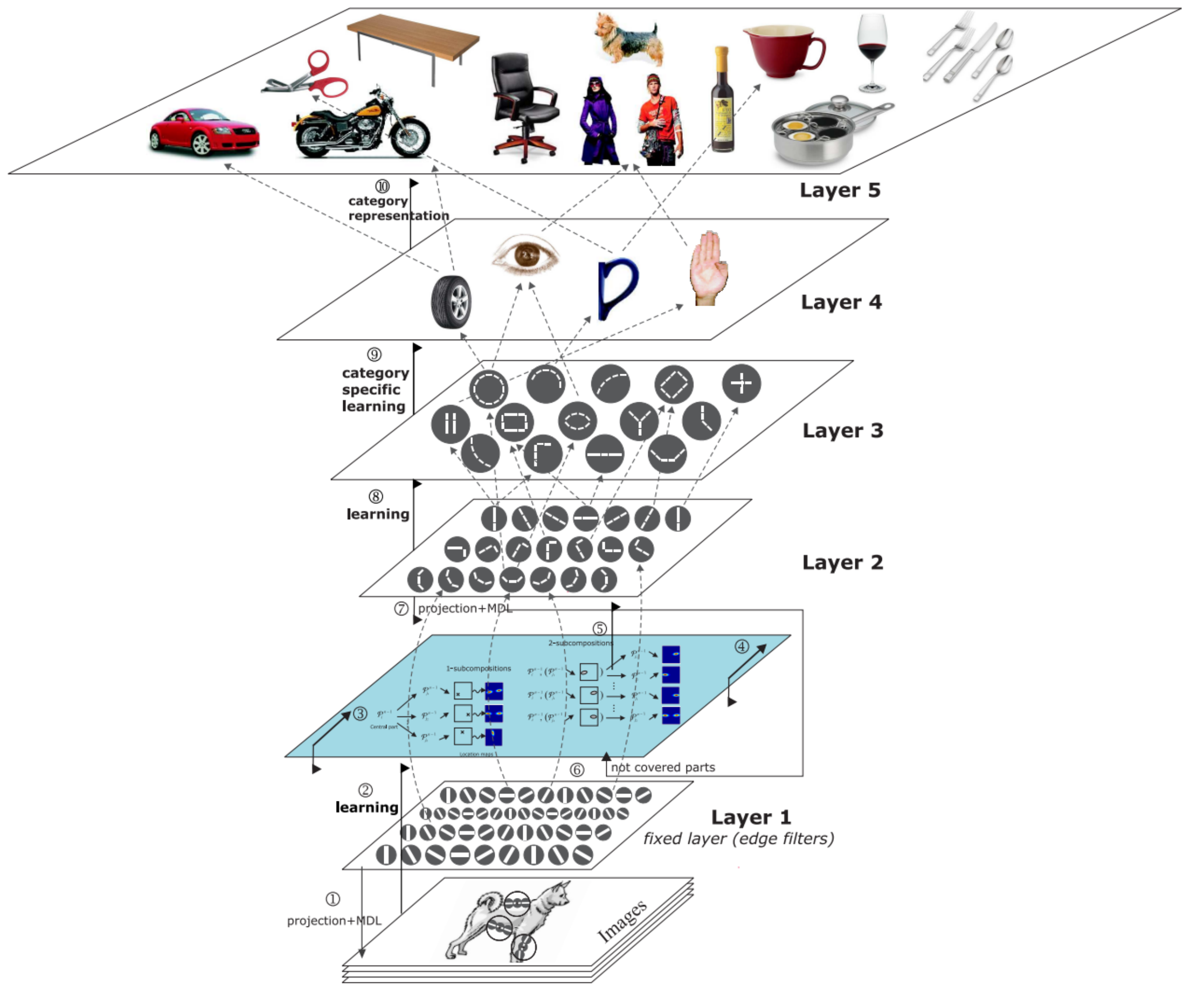


FIGURE 2.1: A custom hierarchical model illustrating the compositionnality and hierarchical nature of vision tasks (source: [117]).

of a machine learning challenge called *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) [176]. One of the tasks was image classification and all but the best method used combinations of shallow learning algorithms and feature extraction. However, the winning team followed a different approach. Building upon neural network research form the previous decades, they used a deep convolutional neural network, AlexNet, trained on the raw images directly [107]. They beat the second best method by a margin of 11% error rate. This extraordinary improvement revived the interest of the machine learning community for neural networks and launched the "deep learning revolution". The success of this method can be partly attributed to some particularly beneficial *inductive bias*, a set of assumptions that narrow the search of a good model in the hypothesis space. This inductive bias includes the use of a trainable multi-layered (hence "deep") structure that can automatically learn hierarchical concepts from the data directly. In other words, instead of manual feature engineering, features are learned automatically. Nowadays, deep learning is a thriving research field that has grown far beyond image classification. Section 2.6 dives a little further into deep learning concepts relevant to this thesis.

2.2.5 Transfer learning

As humans, we have extraordinary learning capabilites. Throughout our lives, we learn to move, communicate, interact with our environments and more. One specific learning ability that we possess is to use knowledge we have acquired in a given context to learn faster in a different context. For example, someone who already plays the violin will probably feel it easier to learn to play the piano than someone who has no music education at all. In a way, we "transfer" knowledge and skills from a task to another.

This idea has been applied in machine learning and from this application emerged *transfer learning* (TL) [225]. This field studies the ways knowledge learned from one or more tasks, called the *source tasks*, can be exploited to learn more effectively on another task, the *target task*. This subject has been researched for few decades now, as the first contributions about transfer learning date back to the end of the 1970s [28]. A surge of interest happened in the 1990s notably with a NIPS-95 workshop called "*Learning to Learn: Knowledge Consolidation and Transfer in Inductive Systems*" which discussed the importance of retaining previously-learned information for efficient learning. Since then, the interest has only been growing and the emergence of deep learning has created new opportunities for transfer learning.

Transfer learning methods are organized based on the properties of the source and target tasks. The different types of supervision (or lack thereof) discussed in Sections 2.2.1 and 2.2.2 also apply to transfer learning in which case the supervision qualifier relates to the target task only. In other words, in supervised transfer learning, the target task is a supervised dataset as described in Equation 2.2. For the remainder of the section, we will assume that the source tasks are also supervised.

Definition 2.2.1. *Transfer learning between a source task* $(\mathcal{X}_s, \mathcal{Y}_s, p_s(\mathbf{x}, y))$ *and a target task* $(\mathcal{X}_t, \mathcal{Y}_t, p_t(\mathbf{x}, y))$ *is said to be* ***homogeneous*** *when* $\mathcal{X}_s = \mathcal{X}_t$ *and* $\mathcal{Y}_s = \mathcal{Y}_t$ *but* $p_s(\mathbf{x}) \neq p_t(\mathbf{x})$ *or* $p_s(y|\mathbf{x}) \neq p_t(y|\mathbf{x})$.

Transfer learning can be *homogeneous* (see Definition 2.2.1) when the source and target tasks only differ by the distributions of their data. As an example, let us suppose we would want to classify pictures taken with a camera equipped with a certain sensor (target dataset B) and that we also have at hands another dataset of pictures taken with a camera equipped with another type of sensor (source dataset A). Each captor has a certain noise pattern which results in dataset A and B to have a slighlty different distributions in the pixel intensities (*i.e.* $p_A(x) \neq p_B(x)$). This specific setup where only the input distributions differ is called *domain adaptation*.

Definition 2.2.2. *Transfer learning between a source task* $(\mathcal{X}_s, \mathcal{Y}_s, p_s(\mathbf{x}, y))$ *and a target task* $(\mathcal{X}_t, \mathcal{Y}_t, p_t(\mathbf{x}, y))$ *is said to be* ***heterogeneous*** *when* $\mathcal{X}_s \neq \mathcal{X}_t$ *and/or* $\mathcal{Y}_s \neq \mathcal{Y}_t$.

Transfer learning can be *heterogeneous* (see Definition 2.2.2). An example that will be addressed later in this thesis is the transfer of a model trained for natural image classification to medical image classification. In this case, the tasks are different as the first consists in identifying the presence of a type of object in the image whereas the other consists in assessing the malignancy of a tumor from an image of a tissue, for instance. The input distributions also differ as medical images have completely different content and appearance.

There exist many different transfer approaches. Based on how they operate, [225] have identified four different categories of transfer learning algorithms. The case when knowledge transferred corresponds to the weights attached to the source examples is called *instance-based* transfer learning. Is referred to as *feature-based* transfer learning the case when knowledge is represented by a subspace spanned by the features in the source and target domains. The transferred knowledge can also be embedded as part of the source domain model: this is *model-based* transfer learning. Finally, when the knowledge is transferred as rules specifying the relations between examples in the source domains, transfer is referred to as *relation-based*.

Transfer learning performance is influenced by several factors including how well the method is able to capture and use transferable knowledge. The task-relatedness is also an important factor: usually the more similar the tasks, the better the performance. Sometimes, performance are worsened by the use of transfer learning. This happens for instance when the source and target tasks are not related enough. Moreover, the training process on the target task can cause some of the previously-learned knowledge to be lost as most machine learning methods do not have an explicit memory mechanism to retain information. These two phenomena are respectively called *negative transfer* [236] and *catastrophic interference* [57]. Although some methods have been proposed to tackle these challenges, how to anticipate and correct them are still open research questions.

We explore further transfer learning in the context of deep learning in Section 2.6.4.

2.2.6 Multi-task learning

In transfer learning, the transfer process happens in two steps. First, knowledge is extracted from the source tasks one way or another, then later used for learning the target task. A similar approach is *multi-task learning* (MTL) where, rather than performing the transfer in two steps, everything happens at once: a model is trained on all tasks simultaneously. Compared to learning each task individually, this approach has several advantages: it increases the total amount of data available for training a model, a more robust and universal representation can be learned by sharing knowledge between tasks and, to a certain extent, it prevents the model to overfit[1] a specific task. In the best scenarii, the use of multi-task learning improves the performance of each individual task compared to a setup where the tasks are treated independently. In opposition, it happens that antagonistic tasks worsen the resulting individual tasks performance. This issue is related to negative transfer and catastrophic interference introduced in Section 2.2.5.

Similarly to transfer learning, multi-task learning methods can be either *heterogeneous* when the tasks are of different types (*e.g.* supervised, unsupervised, classification, regression), or *homogeneous* when tasks have only one type. In supervised multi-task learning, Zhang and Yang [238] have identified three families of methods. *Feature-based* multi-task learning is about sharing knowledge through learning features common among all tasks. With *instance-based* multi-task learning, knowledge is shared through examples deemed useful. Finally, *model* or *parameter-based* multi-task learning use learned models as proxies to extract information about tasks relatedness.

We explore further multi-task learning in the context of deep learning in Section 2.6.5.

2.3 Model evaluation and selection

As stated in Section 2.1, evaluation is a core principle of machine learning as a learning algorithm should select a model that maximizes a performance criterion. In this section, we introduce different concepts related to the evaluation and selection of machine learning models. In Sections 2.3.1 and 2.3.2, we discuss the importance of generalization for machine learning models and the related topics of bias-variance trade-off and overfitting. In Section 2.3.3, we discuss further practical consideration related to model selection. In Section 2.3.4, we finally introduce different metrics used in this thesis.

[1]More on overfitting in Section 2.3.2.

2.3.1 Empirical risk minimization

As stated in the introduction, the objective of a learning algorithm is to find a model $h \in \mathcal{H}$ that maximizes a performance measure or, alternatively, minimizes the expected risk $R(h)$ (for a supervised problem) [212]:

$$R(h) = \mathbb{E}_{\mathcal{X},\mathcal{Y}} \left\{ \ell \left(y, h(\mathbf{x}) \right) \right\} \tag{2.9}$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a loss function which measures the closeness between $h(\mathbf{x})$ and y. The expected risk is also called the *generalization error*. The model that minimizes $R(h)$ is therefore the best possible model for the task and is called the Bayes model h_B. In practice, it is rarely possible to directly minimize the expected risk as one does not have access to the true distributions. It is therefore more convenient to work with an unbiased estimator of the estimated risk, the empirical risk, which is evaluated using the available supervised training set $\mathcal{D}$:

$$R_e(h) = \frac{1}{n} \sum_{i=1}^{|\mathcal{D}|} \ell \left(y_i, h_{\mathcal{D}}(\mathbf{x}_i) \right), (\mathbf{x}_i, y_i) \in \mathcal{D} \tag{2.10}$$

The *empirical risk minimization* (ERM) principle suggests that a learning algorithm should pick a model that minimizes the emprical risk in order to approximate h_B as well as possible. Most machine learning algorithms apply this principle.

2.3.2 Bias-variance trade-off and overfitting

Whereas it can be shown that the empirical risk, under some assumptions, converges to the expected risk when the training set grows ($n \to \infty$), in practice, one does not have access to an infinitely-large dataset. Therefore, the learned model will often differ from h_B and its expected risk will often be larger. The effect of the "*choice*" of a finite training set on the generalization error can be studied using the *bias-variance decomposition* [59, 60, 76]:

$$\mathbb{E}_{LS} \left\{ \mathbb{E}_{\mathcal{X},\mathcal{Y}} \left\{ \ell \left(y, h_{\mathcal{D}}(\mathbf{x}) \right) \right\} \right\} = \text{noise}(\mathbf{x}) + \text{bias}^2(\mathbf{x}) + \text{variance}(\mathbf{x}) \tag{2.11}$$

where $\mathbb{E}_{LS}$ is the expectation over all training sets of size n that can be extracted from $p(\mathbf{x}, y)$ and $h_{\mathcal{D}} \in \mathcal{H}$ is the model learned from a learning set $\mathcal{D} \in LS$. This additive decomposition can be exactly demonstrated when using the squared loss (see Equation 2.4) as ℓ. For other losses, the decomposition may not apply but the intuition is similar. The *noise* term is the error of h_B and is therefore irreducible. The bias is the error of the average model (over models generated from LS) with respect to h_B. The *variance* measures the variability of the predictions around the average model caused by the training set randomness.

Several elements have a direct impact on these error terms such as: model capacity, training set size, noise in the data, *etc*. The *capacity* or *complexity* of a model is related to its ability to capture complex relationships in the data. Linear models (see Section

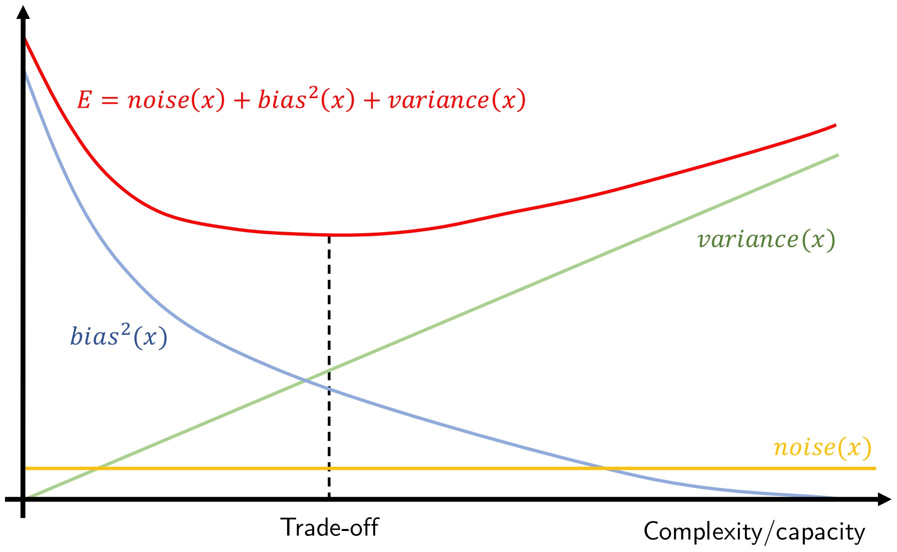


FIGURE 2.2: The bias-variance trade-off illustrated.

2.4) are examples of low-capacity models as they are only able to capture linear relationships. Non-linear models such as random forests (see Section 2.5) or deep neural networks (see Section 2.6) have a higher complexity or capacity compared to linear models.

Low-capacity models usually have low variance as their limited expressiveness does not allow them to change much based on the training data but they also have high bias as failing to capture complex relationships prevents the model from approximating h_B correctly. This behavior is called *underfitting*. In opposition, high-capacity models have low bias as their expressiveness allow them to approximate h_B more accurately but this expressiveness also leads to high variance as the model can learn a function which is too expressive compared to h_B and can fit the noise in the training data. Such models suffer from *overfitting* which hampers generalization, although these models typically present a low empirical risk. The *bias-variance trade-off* results from these observations and states that finding a model that generalizes consists in finding the trade-off between low- and high-capacity or underfitting and overfitting. In Figure 2.2 is plotted the classical representation of this trade-off.

2.3.2.1 Over-parametrized models

With the advent of deep learning, model complexity has exploded as typical deep learning models have millions or even billions of parameters (see Section 2.6.3). The bias-variance trade-off tells us that these models should be extremely prone to overfitting but the reality is different. Typically, it is not uncommon to find tasks for which deep learning models reach an empirical risk close to zero but actually generalize well to unseen data [235]. Although the models are able to basically memorize the training data, the learned function is actually very robust and generalizes well. This phenomenon is currently being investigated and several explanations have been proposed, such as the deep double-descent [20].

2.3.3 Model selection and evaluation in practice

These considerations have direct consequences on practical applications of machine learning. In particular, the presence of overfitting causes the empirical risk to reflect poorly the generalization error of a model. There are two common tasks where using the empirical risk only is not reliable which are *model selection* and *model evaluation*. The former consists in finding the model that would generalize best among a set of candidates and the latter consists in evaluating the expected performance of a model when it will be used on new data. Both tasks involve evaluating the generalization error of a model. There exist several approaches to solve these tasks.

The simplest one is the use of an independent test set: the source dataset is split into two subsets, the train and test sets, which are used to respectively train and evaluate the model. The size of the subsets should be made such that the train set is large enough for the model to be able to learn but the test set should be large enough for the estimate of the error to be reliable. Those two objectives are obviously antagonistic. Therefore, this approach is not viable if the source dataset is too small as reducing even more the size of the sets will make both the training and evaluation unreliable. The test-set approach actually evaluates the generalization error of the model for the given dataset but not the expected generalization error presented in Equation 2.11.

Another approach that actually estimates the expected generalization error is *cross-validation* (CV) which simply repeats the train-test approach with different splits of the source dataset. For each train-test split, a model is trained on the train set and evaluated on the test set. The cross-validation score is computed by averaging the test scores of all splits. There exist several approaches for cross-validation which differ by how they generate their splits. A common approach is *k-fold cross-validation* where the data is randomly split into k subsets and each subset is selected in turn to be the test set while the remaining folds make the train set.

An important consideration is to never use a model selection score as the evaluation score of a model. Indeed, optimizing the choice of a model usually results in the selection score to be an overly optimistic estimator of the generalization performance of the model. In practice, this results in a three-way split of the training dataset: one extracts some train, validation and test sets. The models are trained on the train set, selected on the validation set and the final model is evaluated on the test set. When the dataset is too small for such a three-way split, a common approach consists in extracting a test set from the whole dataset and then performing cross-validation on the train set. As a general rule, every decision influenced one way or another by the output y should be done within a validation loop to avoid overfitting (using either cross-validation or an independent test-set).

These two approaches are based on a strong assumption that the extracted sets are independent. If they are not, this would most probably lead to overfitting and the resulting selection or evaluation scores being overly optimistic. Therefore, it is sometimes not enough to split the sets randomly and domain knowledge must be used during the splitting procedure to enforce sets independence. This issue is sometimes called *data leakage* [95] as information "leaks" between the sets. This topic will be discussed in Section 3.4.1 as digital pathology datasets must be treated carefully to avoid this issue.

2.3.4 Metrics

The metrics presented in this section are mostly supervised classification performance measures. Given a supervised dataset as described in Equation 2.2, the metrics evaluate how the predicted class $\hat{y}_i$ compares to the ground truth class y_i. In the remainder of the thesis, when the higher the better for a metric, we will call it a *score*. In opposition, when the lower the better, we will call it a *loss*. In terms of notations, a loss and a score averaged over a set of data will respectively be denoted $\mathcal{L}$ and $\mathcal{M}$.

2.3.4.1 Accuracy

The accuracy was introduced in our first example of a machine learning problem in Section 2.1 with its dual, the zero-one loss. The accuracy assigns 1 to correct predictions and 0 to misclassified samples. The accuracy score can be computed over a dataset:

$$\mathcal{M}_{\text{acc}} = \frac{1}{n} \sum_{i=1}^{n} (1 - \ell_{0\text{-}1}(y_i, \hat{y}_i)) \tag{2.12}$$

It has the advantage of being simple to interpret, to compute and it applies to both binary and multi-class problems. However, it is affected by *class imbalance*. This phenomenon designates the situation when there is a disparity in the numbers of examples belonging to each of the problem classes in the dataset. For instance, given binary dataset where $n-1$ examples belong to class A and the last example to class B, a classifier predicting only class A would obtain an accuracy close to 1 which seems very good but in reality the classifier did not learn anything from the data.

2.3.4.2 Area under the ROC curve

In binary classification, a specific name is given to each type of successful or unsuccessful predictions and these different types of successes and errors form what is called the *confusion matrix* (see Table 2.1). Each element of this table can be either a number or a proportion of samples falling in the category. The accuracy presented in Section 2.3.4.1 can be re-expressed based on this confusion matrix: $\frac{TN+TP}{N+P}$. In some context, however, it is more informative to look at other types of errors. For instance, when the target task consists in diagnosing cancer, one is more interested in assessing the number of false positives and negatives of the method using, for instance, the *specificity* or *sensitivity* (*a.k.a.* recall) scores.

	Predicted ($\hat{y}$)		
Actual (y)	0	1	Total
0	**T**rue **N**egative	**F**alse **N**egative	**N**egative
1	**F**alse **P**ositive	**T**rue **P**ositive	**P**ositive

TABLE 2.1: A confusion matrix for a binary classification problem.

$$Specificity = \frac{TN}{TN + FP} = 1 - \frac{FP}{N} \tag{2.13}$$

$$Sensitivity = \frac{TP}{P} \tag{2.14}$$

Sensitivity and $(1 - \mathit{Specificity})$ are also called *true positive rate* (TPR) and *false positive rate* (FPR) respectively. These metrics should always be studied together as it is often possible to optimize one at the expense of the other. Therefore a common analysis consists in plotting a model as a point (TPR, FPR) in a two-dimensional graph. Some classification models are able to produce class probabilities instead of just a class. In this case, one can vary a threshold applied to these probabilities in order to generate several points to plot. These points form a visual metric called the *receiver operating characteristic* (ROC) curve (see Figure 2.3) which can be derived into a score: the *area under the ROC curve* (ROC AUC). This score can only be computed for models which produce a meaningfully thresholdable number and it only applies to binary classification in its basic form. However, it provides a measure which is not impacted by class imbalance which is a very interesting property in domains where this issue is common. Another advantage of the ROC AUC is its finer grasp of how a model performs as it evaluates the probabilities, unlike the accuracy where the model is either wrong or right but there is no in-between.

2.3.4.3 Cross-entropy loss

Cross-entropy is rooted in information theory and measures the similarity of two probability distributions p and q that, when defined over a finite and discrete set of events $\mathcal{X}$, is given by:

$$H(p,q) = -\sum_{x_i \in \mathcal{X}} p\left(x_i\right) \log q\left(x_i\right) \tag{2.15}$$

When the model outputs class probabilities, this similarity measure can be used to evaluate the discrepancy between the prediction and the ground-truth. The cross-entropy becomes a loss function called the *categorical cross-entropy*:

$$\mathcal{L}_{ce} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{c=1}^{|\mathcal{Y}|} y_i^{(c)} \log \hat{y}_i^{(c)} \tag{2.16}$$

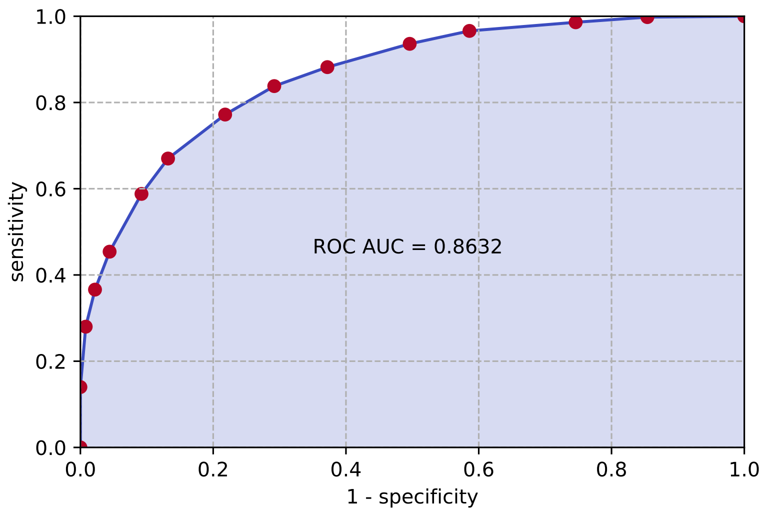


FIGURE 2.3: A ROC curve. The red points correspond to different threshold values applied to the probabilities produced by the model. The area of the light blue zone under the curve is the ROC AUC score.

where $\hat{y}_i^{(c)}$ is the probability predicted by the model for example i and class c and $y_i^{(c)}$ is the one-hot encoding of the ground truth:

$$y_i^{(c)} = \begin{cases} 1, \text{ if } y_i = c \\ 0, \text{ otherwise} \end{cases} \tag{2.17}$$

In the case of binary classification, the metric falls back to the *binary cross-entropy* (BCE):

$$\mathcal{L}_{bce} = -\frac{1}{n}\sum_{i=1}^{n} \left(y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)\right) \tag{2.18}$$

Similarly to ROC AUC, the cross-entropy evaluates probabilities and therefore has a fine grasp on how the model performs. Another important advantage is its differentiability as it can be used as a loss for directly training differentiable models (*e.g.* deep learning models, see Section 2.6.2).

2.3.4.4 Dice score

As opposed to the previous metrics, the dice score is a set similarity measure that is used to evaluate binary image segmentation. Considering two sets $\mathcal{A}$ and $\mathcal{B}$, the dice score is defined as:

$$Dice = \frac{2\,|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A}| + |\mathcal{B}|} \tag{2.19}$$

When working with image segmentation, $\mathcal{A}$ and $\mathcal{B}$ become the sets of true and predicted binary labels for the pixels in the image and the intersection falls back to a dot product. The resulting formula is:

$$\mathcal{M}_{dice} = \frac{2 \sum_i \sum_j y_{ij} \hat{y}_{ij} + \epsilon}{\sum_i \sum_j y_{ij} + \sum_i \sum_j \hat{y}_{ij} + \epsilon} \tag{2.20}$$

where ϵ is a small value added for numerical stability, y_{ij} is the actual class of pixel (i, j) and $\hat{y}_{ij}$ the predicted class for this pixel. The predicted $\hat{y}_{ij}$ can either be binary or a probability. In the latter case, the metric is called the *soft dice score* which is differentiable. When the model outputs class probabilities, instead of using the soft dice score, $\hat{y}_{ij}$ can be binarized using a threshold.

2.3.4.5 Rankings of metrics

In Part II of this thesis, we use a relatively large set of image classification datasets to study transfer learning. In order to compare the different methods, we evaluate how it performs on those datasets. Unfortunately, for reasons that will be explained later, it is not possible to use the same metric for all datasets. Moreover, a metric computed on a dataset is not always comparable to the same metric computed on a different dataset. For this reason, we resort to using *rankings of methods*. First, for each dataset in our pool, we evaluate the methods each with their most appropriate metric and then rank them: the best method gets rank m and the worst gets rank 1 (where m is the number of methods). This convention is used in Chapter 4. Finally, in order to draw general conclusions, we average the rankings over all the datasets. The average ranks therefore provide a single metric for comparing the methods across datasets. In Chapter 5, we use the opposite convention for ranks as the best method is attributed rank 1 whereas the worst is attributed rank m. However, this change of convention does not change the interpretation of rankings except that they can be considered a loss rather than a score.

2.4 Support vector machines

The *support-vector machine* (SVM) binary classification algorithm was invented in the early 1990s [24]. It is originally a linear binary classification method that has been extended for regression and multi-class classification. By using the *kernel trick*, SVM can also learn non-linear models but it is out of the scope of this thesis. An interested reader will learn more about kernel methods and the kernel trick in [76].

In general, assuming a supervised dataset where inputs are such that $\mathbf{x} \in \mathbb{R}^m$ where m is the number of features, a linear model is an hyperplane:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^{m} \theta_j x_j \tag{2.21}$$

where θ_0 and the θ_j are the learnable parameters, respectively called the *bias* and the *weights*. This model can be used for regression in which case the model $h(\mathbf{x}) = f(\mathbf{x})$. A linear model can also be used for binary classification in which case the linear function is called the *decision boundary* and is considered to separate positive from negative samples in the input space: $h(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$ (*i.e.* $\mathcal{Y} = \{-1, 1\}$). Linear methods differ from each other by the way they learn and generate these weights.

Support vector machines optimize the parameters θ_0 and θ_j to generate f in such a way that the minimal distance from the hyperplane to the nearest points from the learning set is maximized. These nearest points are called the *support vectors* and the margin between the support vectors is called the *gutter*. An example of a SVM hyperplane is given in Figure 2.4. It can be shown that the SVM optimization problem falls back to the Lagrange dual:

$$\max_{\boldsymbol{\alpha}} \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{2.22}$$

$$\text{subject to } 0 \leq \alpha_k \leq C, \forall k = 1, ..., n \tag{2.23}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.24}$$

which can be solved efficiently with classical solvers such as LIBLINEAR [55]. When the optimal solution has been found, the weights can be derived by solving the system:

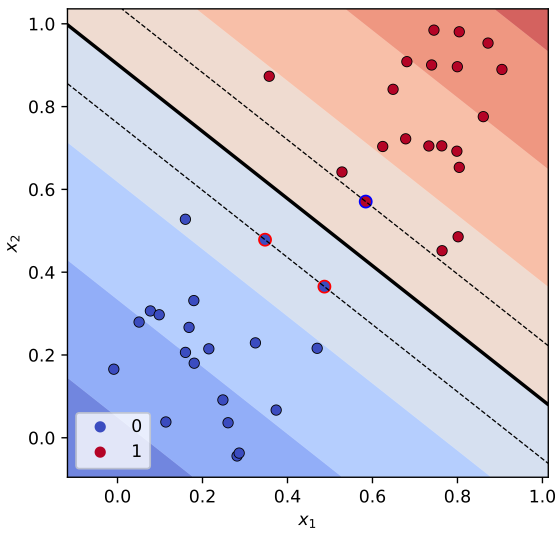
$$\alpha_k \left(y_k \left(\theta^T \mathbf{x}_k + \theta_0 \right) - 1 \right) = 0, \forall k = 1, ..., n \tag{2.25}$$

and the final model written as:

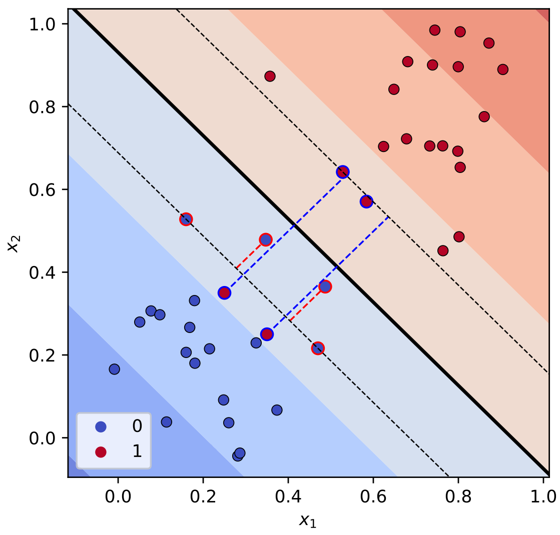
$$h(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + \theta_0 \tag{2.26}$$

In Equation 2.25, $\alpha_k > 0$ for the support vectors x_k and $\alpha_k = 0$ for the other samples. This is an interesting property as it implies that only the support vectors and their weights α_k should be stored for future inference. This makes the final model (see Equation 2.26) very memory efficient for problems where $m \gg n$. This property has also a negative side effect as it means that the model is highly dependent on the support vectors x_k. It can therefore change significantly if the x_k are removed or changed which yields high variance.

The C term in the Lagrange dual formulation is an hyperparameter that relaxes the need for linear separability of the training data and allows for training examples to lie within the gutter or to be misclassified. Low values of C make the optimal solution tolerate more of such examples. Reducing the dependency of the model on very few points also decreases the variance but increases the bias. In opposition, high values of C puts a large penalty on points that lie within the gutter or are misclassified which increases variance but decreases bias.



(A) $C = 10000$



(B) $C = 25$. Notice the new points of class 1 in the class 0 cluster.

FIGURE 2.4: Examples of an hyperplane (the thick black line) learned with SVM applied to a binary classification task ($\mathbf{x} = (x_1, x_2) \in \mathcal{X} \subseteq \mathbb{R}^2$ and $\mathcal{Y} = \{0, 1\}$). The support vectors are delineated with bold colored border. A large value of C penalizes more severely points in the gutter or misclassified.

2.4.1 Multi-class problems

There exist few approaches to use a binary classification algorithm like SVM in a multi-class context. One of them is called *one-vs-one* (OVO) and consists in training $K = \frac{C(C-1)}{2}$ binary classifiers (where C is the number of classes), one for each pair of classes of the dataset. The prediction for a new sample will be the majority class among the predictions of these K models. This requires learning a model number quadratic with regard to the number of classes. When C is large, this can be quite inefficient.

Another approach is called *one-vs-rest* (OVR) and consists in training C models each of which deals with the classification problem of a class versus all the others. The final prediction is given by the model of which the decision function returned largest value. This approach is more computationally efficient than OVO as only C classifiers have to be trained.

2.5 Tree-based methods

Tree-based methods are a popular family of methods invented at least twice in the 1980s by the AI [158] and the statistics [29] communities. Section 2.5.1 presents the basics of decision tree inference and induction, Section 2.5.2 introduces the random forest algorithm. Section 2.5.3 presents a variant of random forest called *extremely randomized trees* (ET) and Section 2.5.4 finally explores how this method can be used for image classification.

2.5.1 Decision trees

Let us consider a supervised dataset where each $\mathbf{x}_i$ is a vector of attributes:

$$\mathbf{x}_i = \left(a_1^{(i)}, a_2^{(i)}, ..., a_m^{(i)}\right),\ a_j^{(i)} \in \mathcal{X}_j \tag{2.27}$$

and each attribute can be either numerical or categorical (binary or multi-valued). A decision tree is a model structured as a tree of which the nodes $\mathcal{Q}$ are divided in two subsets: the internal nodes $\mathcal{I} \subset \mathcal{Q}$ and leaf nodes $\mathcal{F} \subset \mathcal{Q}$. Each internal node $q \in \mathcal{I}$ tests an input attribute $q.k \in \{1, ..., m\}$ and its $q.E$ outwards edges are associated with non-overlapping subsets of this attribute's domain:

$$\mathcal{A}_l^{(q)} \subset \mathcal{X}_{q.k},\ l = 0, ..., q.E - 1 \tag{2.28}$$

which are called *splits*. A leaf node of the tree is labeled with a prediction $\hat{y} \in \mathcal{Y}$. In classification, it can also be a probability distribution over $\mathcal{Y}$. In regression, the output is a real value $\hat{y} \in \mathcal{Y}$. Given x_i, predicting an output value consists in a top-down traversal of the tree. When reaching an internal node q during the traversal, $\mathbf{x}_i$ will follow the edge l such that $a_{q.k}^{(i)} \in \mathcal{A}_l^{(q)}$ which leads to one of the children nodes of q. This process is repeated until the algorithm reaches a leaf of which the associated labeling is the output of the algorithm for $\mathbf{x}_i$.

In the remainder of the thesis, we will only consider numerical attributes ($x_i \in \mathbb{R}$) and binary splits at each internal node ($q.E = 2$). These splits are defined by a threshold $q.v$ such that:

$$\mathcal{A}_0^{(q)} =]-\infty, q.v] \tag{2.29}$$

$$\mathcal{A}_1^{(q)} =]q.v, +\infty[\tag{2.30}$$

This representation allows for an efficient check at each node as the attribute has just to be compared to the threshold. The inference algorithm is formalized in Algorithm 1 for classification with numerical attributes.

2.5.1.1 Decision tree induction

Top-down tree induction is the name of the process of building a decision tree from a supervised dataset. It is a greedy algorithm that, for a node and a set of examples S, will select an attribute and a split which reduce the most the so-called *impurity* of the node. Then, S will be divided in two subsets S_0 and S_1 based on the selected split and these will be used to build recursively the left and right sub-trees respectively. This procedure is repeated until a certain stopping criterion is met. An impurity measure $I(S)$ evaluates the disparity of the output labels y_i in a set S of training examples. For classification, a common impurity measure is the Shannon entropy computed on the

Algorithm 1: Inference with a classification tree and numerical attributes. q.left and q.right denote the left and right children of a node q and the prediction associated with the leaf node $f \in \mathcal{F}$ is accessed through f.pred. The regression algorithm only differs by its type of output.

```
Data: A sample $\mathbf{x}_i = \left(a_1^{(i)}, a_2^{(i)}, ..., a_m^{(i)}\right)$ and the root $r$ of a decision tree.
Result: A probability distribution over $\mathcal{Y}$, the prediction for $\mathbf{x}_i$.
Function TreeInference($r$, $\mathbf{x}_i$):
    $q = r$
    while $q \notin \mathcal{F}$ do
        if $a_{q.k}^{(i)} \leq q.v$ then
            $q = q$.left
        else
            $q = q$.right
        end
    end
    return $q.pred$
```

class frequencies p_i:

$$I(S) = -\sum_{i=1}^{C} p_i \log p_i \tag{2.31}$$

The simplest stopping criterion would be to stop developing the tree when the subset at the node cannot be further divided because either it is pure (*e.g.* only one class represented in S) or because all attributes have a constant value. This algorithm is formalized in Algorithm 2.

A tree built using this simple stopping criterion is said to be *fully-developed*. Such a tree is able to capture the training set perfectly which, in practice, hurts generalization as it usually leads to overfitting. In order to alleviate the problem and reduce the variance, there exist several methods such as pre-pruning which consists in preventing the final tree to grow too deep. Simple pre-pruning techniques are, for example, stopping the induction when a branch reaches a certain depth, or ensuring that all leaf nodes contain at least a given number of samples (see Figure 2.5). An advantage of decision trees are their interpretability as the rules learned by the model can be easily understood by looking at the tree. In practice, decision trees are rarely used alone because of their high variance.

2.5.2 Random forests

Random forests [30] is an ensemble method based on decision trees. Ensemble methods combine the predictions of several models to produce a final prediction. Random forest is part of the family of averaging techniques where models are built independently and their predictions are averaged (*e.g.* a majority vote for classification). It

Algorithm 2: Decision tree induction.

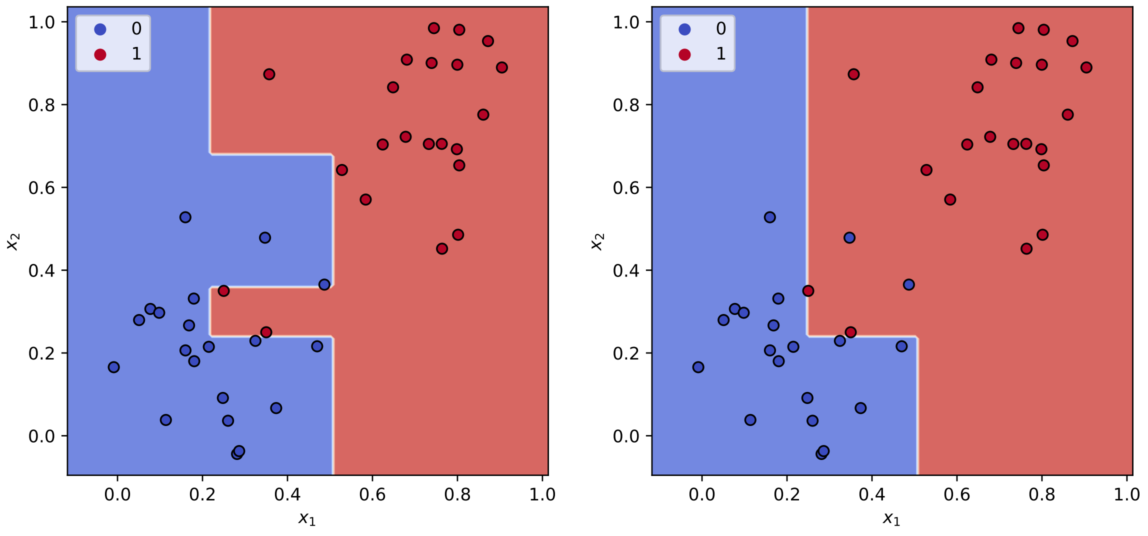
```
Data: A supervised dataset $S$ where each sample $\mathbf{x}_i = \left(a_1^{(i)}, a_2^{(i)}, ..., a_m^{(i)}\right)$.
Result: The root node of a decision tree built for $S$.
Function TreeInduction($S$):
    if StopSplit ($S$) then
        return a node associated with the most appropriate labeling given $S$
    end
    Find the best split $q.v$ for attribute $q.k$ for new node $q$:
    $\arg\min_{v,k} \left[\frac{|S_0|}{|S|} I(S_0) + \frac{|S_1|}{|S|} I(S_1)\right] : S_0 = \left\{(\mathbf{x}_i, y_i) \mid a_k^{(i)} < v\right\}, S_1 = S \setminus S_0$
    $q$.left = TreeInduction($S_0$)
    $q$.right = TreeInduction($S_1$)
    return $q$

Function StopSplit($S$):
    return true if either the $y_i$ or all the attributes are constant in $S$, false otherwise
```

can be shown that this approach reduces variance compared to that of the base model (*e.g.* decision trees) and the more decorrelated the individual models, the larger the variance reduction.

In random forests, decorrelation is achieved through *bagging* (*a.k.a.* bootstrap aggregating) and *random attribute subset selection*. The former consists in training each individual model on a bootstrap sample of the training set: the training set for the t^{th} decision tree in the ensemble is a set S_t built by sampling n examples with replacement from the original training set S. Regarding the attribute subset selection, each tree only considers a subset of $K \in \{1, ..., m\}$ randomly selected features when looking for the best split. In other words, only K random features are considered when searching for $q.k$ at line 6 in Algorithm 2. These two sources of randomness usually increase the bias as the individual models do not have access to all the original training examples and attributes. However, the variance reduction resulting from ensembling is usually of much greater magnitude resulting in a significant performance improvement compared to using a decision tree alone.

2.5.3 Extremely randomized trees

The *extremely randomized trees* (ET, *a.k.a.* extra-trees) [61] are a variant of the random forests algorithm that introduces yet another source of decorrelation by selecting the split $q.v$ at random during the induction (instead of optimizing it, see line 6 in Algorithm 2). Moreover, to attenuate the bias increase, each individual tree is built on the whole training set instead of a boostrap sample. These choices makes the algorithm particularly computationally efficient as it is not necessary to iterate over the training samples to optimize the split anymore.



(A) Fully-developped (B) Maximum depth = 3

FIGURE 2.5: Decision tree decision boundary (white line). Pre-pruning using the maximum depth parameter reduces the complexity of the model.

2.5.4 Image classification with extra-trees

Extra-trees can be used for image classification as presented in [131]. The core idea behind this algorithm is to represent an image as a set of n_w random subwindows. Subwindows are rectangle patches extracted from the image, resized to have all the target height t_r and width t_c and flattened into a vector of dimension $t_r \times t_c \times b$ (b, the number of color channels, or bands, in the image). The original dimensions of the rectangular patches are drawn uniformly at random in a range of proportions s_{min} and s_{max} of the image size ($0 \leq s_{min} < s_{max} \leq 1$). The position of the patches are also drawn uniformly at random. Each subwindow is attributed the class of its parent image. The resulting training dataset containing $n \times n_w$ examples, each a vector of dimension $t_h \times t_w \times c$, can be used to train an extra-trees classifier. At inference, the same extraction process is applied and the final prediction for the image is determined by a majority vote on the predicted classes for its subwindows. Regarding parameters, the more windows and the more trees in the forest, the better. In addition to the tree complexity hyperparameters, the image colorspace, s_{min} and s_{max} should be tuned as well to optimize generalization.

This variant of the algorithm is called *extremely randomized trees as direct-learner* (ET-DIC) but another variant exists where the forest is used as a feature-learner (ET-FL). The idea of ET-FL is to train a limited number of trees with the subwindows dataset similarly to the ET-DIC approach. But rather than using the forest as a direct classifier, it is used to create a new representation for the images. This representation is a vector of the same dimension as the number of leaves in the forest and value v_i for leaf i corresponds to the frequency of subwindows of the image reaching the leaf when propagated into the forest. This representation can then be used to train another classifier like SVM. At inference, the representation for the new samples is extracted from the forest which are then classified with the second classifier. In general, ET-FL is superior to ET-DIC in terms of performance.

2.6 Deep learning

As introduced in Section 2.2.4, deep learning is a broad research field that has grown rapidly since the breakthrough of AlexNet in 2012. It encompasses many different research topics covering all areas of machine learning. In this section, we dive further into deep learning concepts and methods relevant to this thesis. We introduce the basic components of modern neural network architectures in Section 2.6.1. Section 2.6.2 discusses how neural networks are learned and optimized in practice. Section 2.6.3 presents few modern neural network architectures. Finally, in Sections 2.6.4, 2.6.5 and 2.6.6, we respectively discuss deep transfer learning, multi-task learning and self-training and present works and methods related to ours.

2.6.1 Components of modern neural networks

A feedforward neural network can be seen as a parametrized function approximator h which combines several intermediate functions $h^{(l)}(\cdot;\theta^{(l)})$ ($l = 1, ..., L$), also called *layers*, through composition: $h(\mathbf{x};\theta) = (h^{(L)} \circ h^{(L-1)} \circ ... \circ h^{(1)})(\mathbf{x})$. θ is the set containing all the learnable parameters of the neural network h and $\theta^{(l)}$ are the parameters of the function $h^{(l)}$ (or layer l). The feedforward nature of the network specifies that information only flows one-way, from the input $\mathbf{x}$ to the output y. In other words, there is no feedback loop.

Unlike decision trees, for instance, which learn the tree structure automatically, the architecture of a neural network is built manually (most of the time) and consists in choosing and combining the appropriate functions $h^{(l)}$ for the task at hand. Although it introduces some complexity when it comes to finding the best architecture for a problem, this modularity actually offers an unprecedented way of approaching model design as an engineering problem and implementing inductive bias. Moreover, as long as the chosen functions are differentiable, the final network can be trained using the *backpropagation* algorithm (see Section 2.6.2) independently of the architectural choices.

The idea behind one of the first machine learning algorithms ever published, the perceptron [168], is still at the core of most deep learning architectures today. Loosely modeling the working of brain neurons, the perceptron can be represented as:

$$f(\mathbf{x};\boldsymbol{\theta}) = \sigma\left(\theta_0 + \sum_{i=0}^{m} \theta_i x_i\right) \tag{2.32}$$

where $\sigma(\cdot) : \mathcal{X} \to \mathbb{R}$ is a non-linear differentiable function called the *activation function*, $\boldsymbol{\theta}$ is the vector of learnable parameters of which θ_0 and θ_i ($\forall i$) are respectively bias and weights. An eligible function for $\sigma(x)$ should have at least two modes transitioning around $x = 0$. In the past decades, typical σ functions were the hyperbolic tangent, the sigmoid or the step function. Nowadays, the most common is the *rectified linear unit* (ReLU) function:

$$\sigma(x) = \mathrm{ReLU}(x) = \begin{cases} 0, \text{ if } x < 0 \\ x, \text{ else} \end{cases} \tag{2.33}$$

which is particularly efficient to compute and has some properties that interact favorably with the training algorithm making neural networks easier to train.

The perceptron, or *neuron*, is simply a linear model and is therefore not complex enough for many types of tasks. However, perceptrons can be combined together into a perceptron layer: a parallel combination of f perceptrons (f is the width of the layer) each with their own set of weights and biases and all taking the same vector as input. A perceptron layer therefore outputs a vector $\mathbf{z} \in \mathbb{R}^f$.

In the spirit of learning a hierarchy, perceptron layers can be connected one after another to form a *multi-layer perceptron* (MLP) (see Figure 2.6). It can be shown that sufficiently complex MLPs are universal function approximators (*i.e.* they can learn any function) [78] which is an interesting property. However, realistically dimensioned MLPs have the drawback of being very complex models easily reaching millions or even billions of learnable parameters. This makes them difficult to optimize and prone to overfitting. Regularization techniques exist to reduce model complexity such as weight decay (*i.e.* small absolute weights values are preferred over large ones) or dropout [190] (*i.e.* during training, some neurons are disabled at random) but unfortunately are often not enough. Therefore, whereas MLPs still have their use in some applications and or as part of larger and more diverse architectures, they are rarely used by themselves nowadays.

2.6.1.1 Convolutional neural networks

As introduced in Section 2.2.4, *convolutional neural networks* (CNN) are the origin of the "deep learning revolution". Despite the recently-revived interest, CNNs have been researched since the late 1980s [115]. A CNN is a network containing at least one convolutional layer. Nowadays, such layers are a core component of all networks processing structured data (time series, images, video, *etc.*). For instance, in image classification, typical architectures (see Figure 2.7) stack several convolutional and pooling layers followed by a fully-connected network (*i.e.* a MLP). The convolutional layer is introduced in this section and the pooling layer in Section 2.6.1.2.

Let us suppose a naive implementation of an image classifier with a MLP where each neuron in the first layer is connected to each input pixel. This model is extremely complex and prone to overfitting, as it would realistically contain billions of weights for a regular image. Natural images and related machine learning tasks have interesting properties that can be used to improve this naive model.

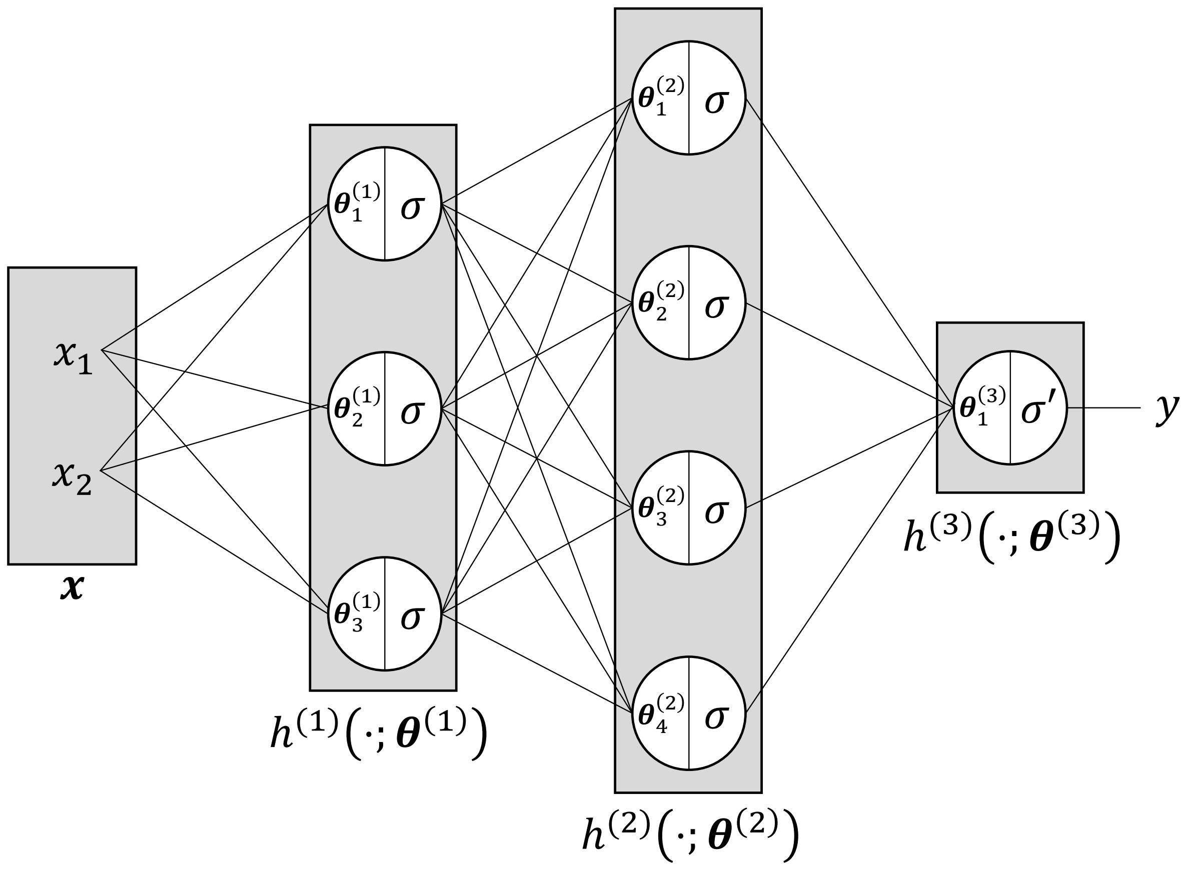

x_1
x_2
$\boldsymbol{x}$
$\boldsymbol{\theta}_1^{(1)}$ σ
$\boldsymbol{\theta}_2^{(1)}$ σ
$\boldsymbol{\theta}_3^{(1)}$ σ
$h^{(1)}(\cdot;\boldsymbol{\theta}^{(1)})$
$\boldsymbol{\theta}_1^{(2)}$ σ
$\boldsymbol{\theta}_2^{(2)}$ σ
$\boldsymbol{\theta}_3^{(2)}$ σ
$\boldsymbol{\theta}_4^{(2)}$ σ
$h^{(2)}(\cdot;\boldsymbol{\theta}^{(2)})$
$\boldsymbol{\theta}_1^{(3)}$ σ'
$h^{(3)}(\cdot;\boldsymbol{\theta}^{(3)})$
y

FIGURE 2.6: A multi-layer perceptron with 2 hidden layers $h^{(1)}$ and $h^{(2)}$ for an input $\mathbf{x}$ with two features.

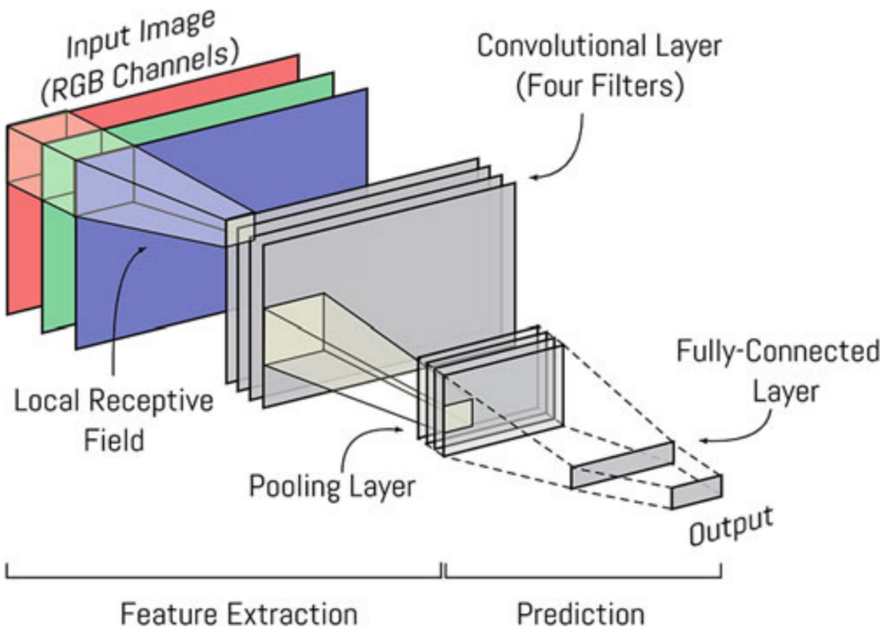


FIGURE 2.7: A convolutional neural network (source: [138]).

A first property is that *pixel correlations are local*, meaning that pixels far from each other in the image are less likely to be correlated than close pixels. It implies that connecting individual neurons to all input pixels is excessive and one would rather want to connect them to patches of close pixels. A second property is that an image can be considered a *stationary signal* (*i.e.* pixel statistics are similar at different locations of the image). Therefore, it is relevant to use the same set of weights for all the locally-connected neurons of a layer. These architectural choices are examples of inductive bias and motivate the *convolutional layer*. Although, we have discussed the case of images, convolutional layers can also be generalized for 1D (*e.g.* time series) or 3D (*e.g.* 3D image volume) when the locality and stationarity assumptions hold.

Formal definition. A two-dimensional convolutional layer h can be seen as a function generating an arbitrary number f_{out} of *feature maps* $\mathbf{z}^{(out)} \in \mathbb{R}^{r_{out} \times c_{out} \times f_{out}}$ from an input $\mathbf{z}^{(in)} \in \mathbb{R}^{r_{in} \times c_{in} \times f_{in}}$. Feature map $k \in \{1, 2, ..., f_{out}\}$ is the result of convoluting a learnable weight tensor $\boldsymbol{\theta}_k \in \mathbb{R}^{r_f \times c_f \times f_{in}}$ called *kernel* or *filter* over the input $\mathbf{z}^{(in)}$:

$$a_{ijk} = \theta_{k;0} + \sum_{r=1}^{r_f} \sum_{c=1}^{c_f} \sum_{f=1}^{f_{in}} \theta_{k;r,c,f} \times z^{(in)}_{i+r,j+c,f} \tag{2.34}$$

where $\mathbf{a} \in \mathbb{R}^{r_{out} \times c_{out} \times f_{out}}$ are the intermediate activation maps, (i, j) are the coordinates of a vector of $\mathbf{z}^{(in)}$, $\theta_{k;0}$ is the bias and $\theta_{k;r,c,f}$ is the weight at coordinates (r, c, f) of tensor $\boldsymbol{\theta}_k$. Similarly as for the perceptron, this linear operation is then followed by a

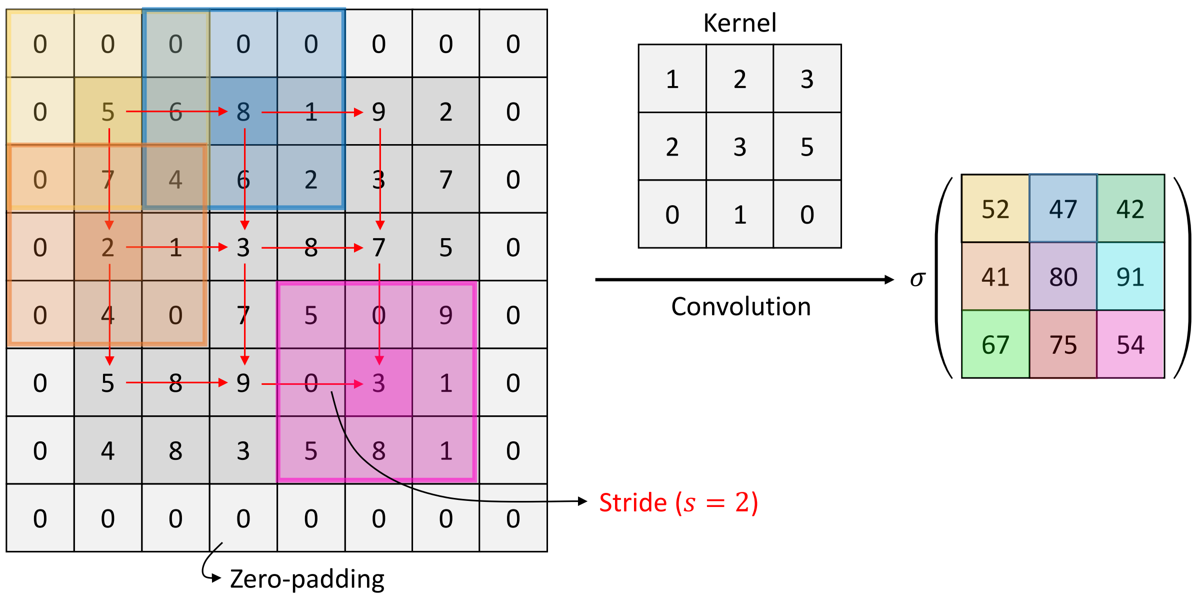


FIGURE 2.8: Illustration of stride and padding (assuming a zero bias).

non-linearity:

$$\mathbf{z}^{(out)} = h\left(\mathbf{z}^{(in)}\right) = \sigma\left(\mathbf{a}\right) \tag{2.35}$$

where σ is applied to each component of $\mathbf{a}$ individually.

Hyperparameters. Convolutional layers have many hyperparameters including the number of feature maps f_{out} and filter dimensions r_f and c_f (typically $r_f \times c_f = 3 \times 3$ or 5×5) which together directly control the number of learnable parameters of the layer (and therefore its complexity). In addition to those, one usually introduces a *stride* s which consists in evaluating the kernel every s pixels (instead of every pixel) and therefore divides by s the size of the feature map for each dimension to which it is applied. It also reduces the amount of computations involved with the layer. The *padding* defines how convolution behaves near the edges of the image where all the considered pixels might not exist. Typical padding options are considering missing pixels to have value 0, mirroring the image at the edge, *etc*. Padding and stride are illustrated in Figure 2.8.

Transposed convolution layer. This type of layer can be seen as the opposite operation of convolution. It is commonly used to upsample a signal with a learnable filter. Its hyperparameters are similar to those of regular convolutional layers and include kernel size $r_f \times c_f$, stride s and padding. For instance, supposing a transposed convolution with no padding and a kernel size equal to the stride ($K = r_f = c_f = s = 2$, *i.e.* no overlap between transposed convolution window), the activation map is given by:

$$a_{ijk} = \theta_{k;0} + \sum_{f=1}^{f_{in}} z^{(in)}_{\lfloor \frac{i}{2} \rfloor, \lfloor \frac{j}{2} \rfloor, f} \theta_{k; i \text{ mod } 2, j \text{ mod } 2, f} \tag{2.36}$$

where $\mathbf{z}^{(in)} \in \mathbb{R}^{r_{in} \times c_{in} \times f_{in}}$ are the input feature maps and $\theta_{k;0} \in \mathbb{R}$ and $\boldsymbol{\theta}_k \in \mathbb{R}^{r_f \times c_f \times f_{in}}$ are the learnable bias and weights. The output feature maps $\mathbf{z}^{(out)} \in \mathbb{R}^{r_{out} \times c_{out} \times f_{out}}$ are computed similarly as in Equation 2.35 with $r_{out} = 2 \times r_{in}$ and $c_{out} = 2 \times c_{in}$.

Properties of convolution. In addition to exploiting locality and stationarity, CNNs have other interesting properties which contribute to their efficiency for processing structured data. They have *equivariance to translation* which means that translating the convolutional layer input along some of the dimensions only translates the output along the same dimensions. For tasks like image classification, object location is often irrelevant for inferring the class. Therefore, the network does not have to learn this invariance which makes training easier. It has also been shown that, for image classification, the learned kernels are similar to classical computer vision filters capturing edges or color patterns in early layers and, as one ascends the network, kernels show compositionability, invariance and class discrimination [233] confirming the ability of such a network to learn and use the inherent hierarchical structure of the data. Stacking convolutional layers also brings the benefit of increasing the receptive field of late layers. The receptive field of an activation $a^{(l)}_{ijk}$ in a feature map k at layer l is the set of pixels of the input that influence this activation. The larger the receptive field, the more context is considered by the activation.

There is an interesting parallel to draw between the random subwindows tree-based method presented in Section 2.5.4 and convolutional neural networks. Indeed, whereas the random subwindow algorithm randomly selects windows of varying scales in an input image, a convolutional layer actually processes all windows exhaustively at a given scale. In the case of a convolution layer, the window is obviously smaller (*i.e.* few pixels wide) but because convolutional layers are stacked, the actual window processed for an activation of the last layer (*i.e.* the receptive field) is larger and comparable to the size of window of the random subwindows algorithm. Obviously, both methods differ by how they exploit these windows and deep networks are able to learn more expressive models from them because of the hierarchical structure and learning.

2.6.1.2 Pooling layer

Pooling layers are another common components of CNNs. Their role is to reduce the dimensionality of feature maps by aggregating close activations. In a way, a pooling layer can be seen as a special convolutional layer with stride. However, unlike the convolutional layer, the kernel is not learned but rather computes a (possibly non-linear) statistic of its input. The depth of the feature maps before and after pooling does not change (*i.e.* $f_{in} = f_{out}$). Pooling share some hyperparameters with convolution including stride and padding. Regarding the aggregation function, the most common choices are *max pooling* or *average pooling*. The former will output the largest activation among the one covered by the kernel:

$$a_{ijk} = \max \left\{ x_{i+r,j+c,k} \mid r = 1, ..., r_f; c = 1, ..., c_f \right\} \tag{2.37}$$

The latter will average the activations covered by the kernel:

$$a_{ijk} = \frac{1}{r_f \times c_f} \sum_{r=1}^{r_f} \sum_{c=1}^{c_f} x_{i+r,j+c,k} \tag{2.38}$$

Pooling, similarly to convolution, brings equivariance to translation and increase the receptive field of deeper layers. The reduction of dimensionality also reduces the computational cost of the network as deeper layers have smaller feature maps to process. There exist other types of pooling layers which are described in [62].

2.6.2 Neural network optimization

Remains the question of how to find the best parameters for the neural network $h(\cdot;\theta)$ using a training set. Obviously, these weights should be tuned so that the final model minimizes the expected risk (*i.e.* the generalization error) which is not directly possible as explained in Section 2.3.1. Following the ERM principle, $h(\cdot;\theta)$ can be optimized by minimizing its error on the training set measured by a loss $\mathcal{L}$. In most cases, it is not possible to find the solution analytically and, therefore, one resorts to numerical optimization and variants of *gradient descent* algorithms.

Starting from initial model parameters θ_0[2], gradient descent is an algorithm that iteratively builds a sequence of parameters θ_i such that the loss should ideally decrease when i increases. Every iteration, the parameters are updated by applying them a correction in opposite direction of the gradients of $\mathcal{L}$ in the parameters space:

$$\theta_i = \theta_{i-1} - \gamma \nabla \mathcal{L}(h(\cdot;\theta_i)) \tag{2.39}$$

where γ is the learning rate, an hyperparameter for tuning the amplitude of the parameters update.

Gradient descent does not guarantee convergence to a global minimum in general as the optimization can either reach a local minimum or diverge. The choices of θ_0 and γ are important to avoid divergence. Regarding the local minima issue, the very high-dimensional nature of neural networks makes it unlikely to have no dimension along which the model can improve at a given iteration. Moreover, finding parameters that achieve the global minimum of the training loss is generally unwanted as it might lead

[2]Starting from this section, the notation θ will refer to the set of parameters of a neural network. Depending on the context, θ_i will refer to either a part i of this neural network or the state of these parameters at a time step i.

to overfitting. Nevertheless, this last remark can be questioned by the recent research on over-parametrization discussed in Section 2.3.2.1.

At every iteration, the original gradient descent algorithm computes gradients over the whole training set which is particularly inefficient when $n \gg$. A way of improving this consists in using *stochastic gradient descent* (SGD) where the gradients are approximated using a subset $\mathcal{B}$ of training samples called a *batch* instead of the whole set. This approach has been shown to be particularly efficient for both generalization of the model and computational cost of training [25]. When $1 < |\mathcal{B}| < n$, SGD becomes *mini-batch gradient descent* which is the most used optimization strategy for training neural networks nowadays. The batch size is often chosen based on practical considerations like the amount of memory available on the hardware running the learning algorithm.

Several adjustements to the gradient descent formulation of Equation 2.39 have shown to be effective for convergence and stability of the optimization. In the thesis, whenever we optimize a neural network by gradient descent, we use the Adam optimizer [100] approach which normalizes the gradients using moving estimates of their mean and uncentered variance. This method is efficient and does not require much tuning of the hyperparameters yet makes training more robust.

2.6.2.1 Backward propagation

Gradient descent, as its name suggests, heavily relies on gradient of parameters for updating the model. The success of deep learning can be partly attributed to an efficient algorithm for computing these gradients called *backward propagation* [174] (*a.k.a.* backpropagation). The algorithm is based on the *chain rule* which states that the derivative of a composition of functions $y(x) = (h^{(L)} \circ h^{(L-1)} \circ ... \circ h^{(1)})(x)$ with respect to its input can be broken down into a product of derivatives. Given $y_i = (h^{(i)} \circ h^{(i-1)} \circ ... \circ h^{(1)})(x)$, the composition of the i^{th} first functions ($i = 1, ..., L$), the chain rule states that:

$$\nabla h = \frac{\partial h(x)}{\partial x} = \frac{\partial y_L}{\partial x} = \frac{\partial y_L}{\partial y_{L-1}} \times ... \times \frac{\partial y_2}{\partial y_1} \times \frac{\partial y_1}{\partial x} \tag{2.40}$$

This applies directly to feedforward neural networks which are compositions of functions and implies that computing the parameters gradients can be broken down into computing local gradients at every layer. The backward propagation algorithm starts from the loss and iteratively evaluates local gradients going backward in the network until all parameters have been reached. These local gradients can then be combined using the chain rule and the resulting gradients can be used to update the parameters as dictated by gradient descent. This obviously requires that all functions $h^{(l)}$ are differentiable as introduced in Section 2.6.1.

Relying on gradients of a long chain of functions for the optimization can be troublesome at times. Indeed, when a neural network is not build carefully, *vanishing* or *exploding gradients* can appear. Some activation functions such as the hyperbolic tangent or the sigmoid have saturating modes as they converge to constant values. As these fonctions converge towards a constant, their derivatives become smaller and smaller and converge to zero. Because of the multiplicative nature of the chain rule, one gradient in the chain is enough to cancel all gradients upstream which happens when the activations start working in their saturating regime. There exist best practices to reduce the likelihood of vanishing gradients: weights must be carefully initialized, it is better to avoid activation functions with saturating modes (ReLU is a good candidate) and some architectural tricks can help (see Sections 2.6.2.2 and 2.6.3). In opposition, exploding gradients cause divergence because gradients are too large. Again architectural tricks, careful weigths and learning rate initilization makes this issue less likely to occur.

Nowadays, all deep learning frameworks implement backward propagation and also feature automatic differentiation. Automatic differentiation means that every basic mathematical function present in the library is associated with its analytical derivative. Therefore, during backpropagation, the algorithm can compute the gradients automatically and exactly based on this information. The combination of these two features makes training a neural network particularly easy as everything is automated.

2.6.2.2 Batch normalization

The *batch normalization* (BN) [83] layer is part of the family of normalization layers. This layer acts as a regularizer and is usually placed before activations in neural networks. Given a signal $\mathbf{x} = \left(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_f\right) \in \mathbb{R}^f$ (*e.g.* the output of a convolutional layer), the BN layer maintains an estimate of the mean μ_i and variance σ_i ($i = 1, ..., f$) of the preceding layer output computed over the batch. These estimates are used to normalize the inputs into an intermediate signal $\hat{\mathbf{x}}$. Obviously, whether this normalization is actually beneficial for the network is task-, layer- and feature-dependent and there are probably scenarii where normalizing would hurt performance. Therefore, the signal $\hat{\mathbf{x}}$ is then followed by a learnable linear layer allowing the network to learn to revert this normalization if necessary:

$$BN(\mathbf{x}) = \left[\alpha_1 \left(\frac{\mathbf{x}_1 - \mu_1}{\sqrt{\sigma_1^2 + \epsilon}}\right) + \beta_1 \quad ... \quad \alpha_f \left(\frac{\mathbf{x}_f - \mu_f}{\sqrt{\sigma_f^2 + \epsilon}}\right) + \beta_f\right] \tag{2.41}$$

where the α_i and β_i are learnable parameters. During training, the input statistics μ_i and σ_i are computed on-the-fly using moving averages. At inference, the statistics are frozen as well as the learnable parameters α_i and β_i.

Batch normalization greatly helps network optimization by maintaining intermediate activations within acceptable ranges of values and therefore reducing the risk of vanishing or exploding gradients, narrowing the parameter search and allowing higher learning rates.

Unfortunately, the use of batch statistics can also be an issue. When there is a sudden change in the model input distribution, the batch statistics are likely to change quickly but the linear layer, restricted by the learning rate, will probably not be able to adapt as quickly. This can cause instabilities while the network adapts its batch

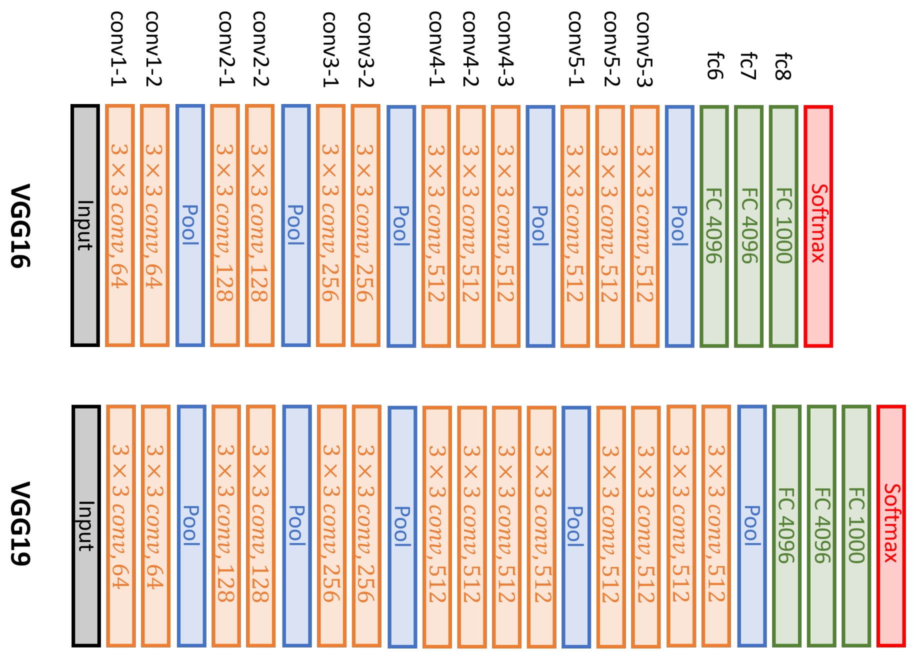


FIGURE 2.9: VGG16 and VGG19 architectures (source: [47]).

normalization layers for the new distribution. This situation can occur for instance with transfer learning where source and target tasks can have drastically different input distributions.

2.6.3 Modern network architectures

This section presents different architectures used in our contributions.

Simple layer-stacking architecture After the success of AlexNet, researchers investigated the effect of model depth on classification performance. An architecture in particular caught the attention by ending up a runner up for a later edition of ILSVRC in 2014. This architecture is named VGG [183] after the team who participated to the challenge. It has two implementations that we use in this thesis: VGG16 and VGG19. The number is the count of layers with trainable weights in the architecture. These architectures are pretty simple (see Figure 2.9) as they only stack blocks of few convolutional layers with ReLU activation followed by pooling. The last convolutional layer is followed by a 3-layers MLP.

Residual architecture Although improving over AlexNet, very deep networks like VGGs are subject to training difficulties: beyond a certain model depth, accuracies sometimes start to decrease. This can be attributed to the fact that, for a given task, adding layers beyond a certain depth is not necessary. Moreover, it can be difficult for gradients to flow back given the depth of the architecture (*i.e.* vanishing gradients). In order to address those two issues, ResNet [77] introduces the *residual mapping*, or

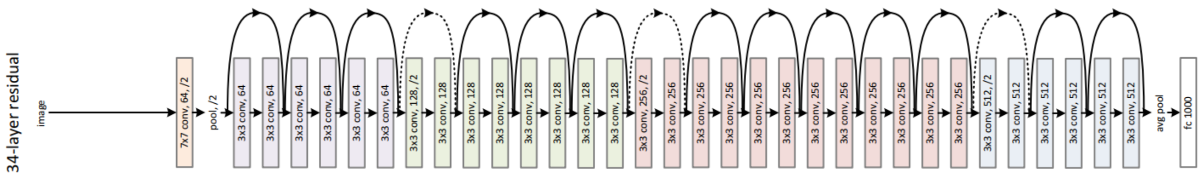


FIGURE 2.10: ResNet34, same design as ResNet50 but less layers. Residual connections are placed every two convolutional layers (source: [77]).

skip-connection. Given a layer $h^{(l)}(\mathbf{x})$, the residual mapping $r^{(l)}(\mathbf{x})$ would be:

$$r^{(l)}(\mathbf{x}) = \mathbf{x} + h^{(l)}(\mathbf{x}). \tag{2.42}$$

This residual mapping makes it easier for the optimizer to simply ignore the layer as it just has to tune all the weights in $h^{(l)}(\mathbf{x})$ to 0 in which case $r^{(l)}(\mathbf{x}) = \mathbf{x}$. Moreover, gradients are still able to flow freely through the skip connection. This increased trainability has been empirically confirmed as very deep residual network (up to 150 layers) have won ILSVRC in 2015 and 2016 and have shown impressive performance in many other contexts. Another advantage of residual networks is that, even though they are deeper than their VGG counterparts, they need much less parameters to reach better performance[3] making ResNets more lightweight. Nowadays, residual connections are a common building block in deep architectures. In this thesis, we use one of the first iterations of ResNets, namely ResNet50 (see Figure 2.10 for ResNet34, a similar architecture).

Inception architecture The winning method of ILSVRC2014 was based on an architecture called GoogLeNet [197] (*a.k.a.* InceptionV1). This architecture is based on *inception modules*. An inception module is a composition of convolutional layers built to maintain a certain level of representativeness while reducing the number of training parameters compared to a block of plain convolutional layers. An example of a trick used to reduce the number of parameters is to use two consecutive 3×3 instead of one 5×5 convolutional layers. In this thesis, we use InceptionV3 [198], the third iteration of the architecture, which essentially scales it up by using factorized convolutions and regularization. We also use InceptionResNetV2, a version of the Inception architecture using residual connections [196].

Dense architecture The densely connected convolutional networks [81] push further the idea of residual connection. Let us suppose a group g of L consecutive convolutional layers $h^{(l)}$ ($l = 1, ..., L$) taking $\mathbf{x}$ as input. In this group, each layer $h^{(l)}$ receives as input the outputs of all preceding layers $h^{(i)}$ ($i = 1, ..., l-1$) and the input signal

[3]VGG16 and 19 have more than 140M parameters whereas ResNet50 has 25M and the largest ResNet152 has 66M parameters.

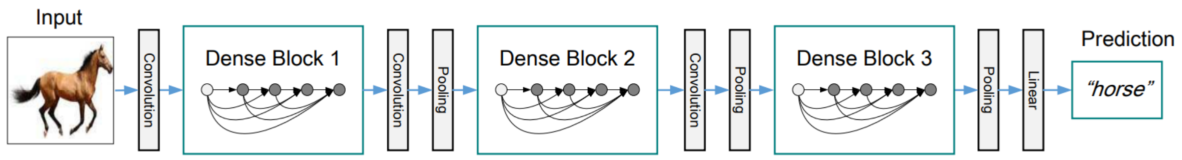


FIGURE 2.11: A densely connected network (source: [81]).

x. This group g is called a *dense block* and is the basis for the densely connected networks. Such networks typically stack several dense blocks connected together with one convolution and one pooling layers (see Figure 2.11).

A segmentation architecture, UNet For tackling image segmentation tasks, one generally uses *fully-convolutional networks* (FCN) which is a kind of network where (almost) all layers are convolutional. One of the most popular architectures for segmentation is a U-shaped network called U-Net [167]. This architecture is composed of a contracting and an expanding paths (hence the "U", see Figure 2.12). Similarly as for classification networks, the contracting path is composed of convolutional and pooling layers that progressively encodes the input images into high-level context features. The expanding path combines decoded high-level features with low-level features from the contracting path to eventually generate a segmentation map. The decoding of low-level features is learned using transposed convolutions. In the original U-Net implementation, padding is disabled and kernel size is set to be equal to the stride (same setup as presented in Section 2.6.1.1). In order to combine decoded high level features and low-level features, U-Net uses skip connections from the contracting to the expanding paths.

2.6.4 Deep transfer learning

As presented in Section 2.2.5, transfer learning regroups several families of methods which have been and continue to be explored through the prism of deep learning [201]. In this thesis, we focus on supervised model-based (or network-based) transfer learning applied to image classification. With this approach, knowledge is encoded through the parameters of a model. This model is first trained on a source task, usually a large image classification dataset, then transferred to the target task.

There are two approaches for transferring a deep neural network to a target task. Let us consider three networks h_s, h_o and h_n respectively parametrized by θ_s, θ_o and θ_n. In the remainder of this section, we will designate a network interchangeably by its set of parameters θ_i or its function $h_i(\cdot;\theta_i)$. The network θ_s is shared between the source and target tasks and θ_o and θ_n are the task-specific networks. Both approaches first require the network $h = h_o \circ h_s$ to be pre-trained on the source task. At the end of pre-training, θ_s encodes the knowledge to be transferred.

The first transfer approach is *feature extraction* (see Figure 2.13) and simply consists in generating a new representation for samples of the target task using θ_s as an encoder. For each sample $\mathbf{x}_i$ of the target task, this encoder generates a feature vector

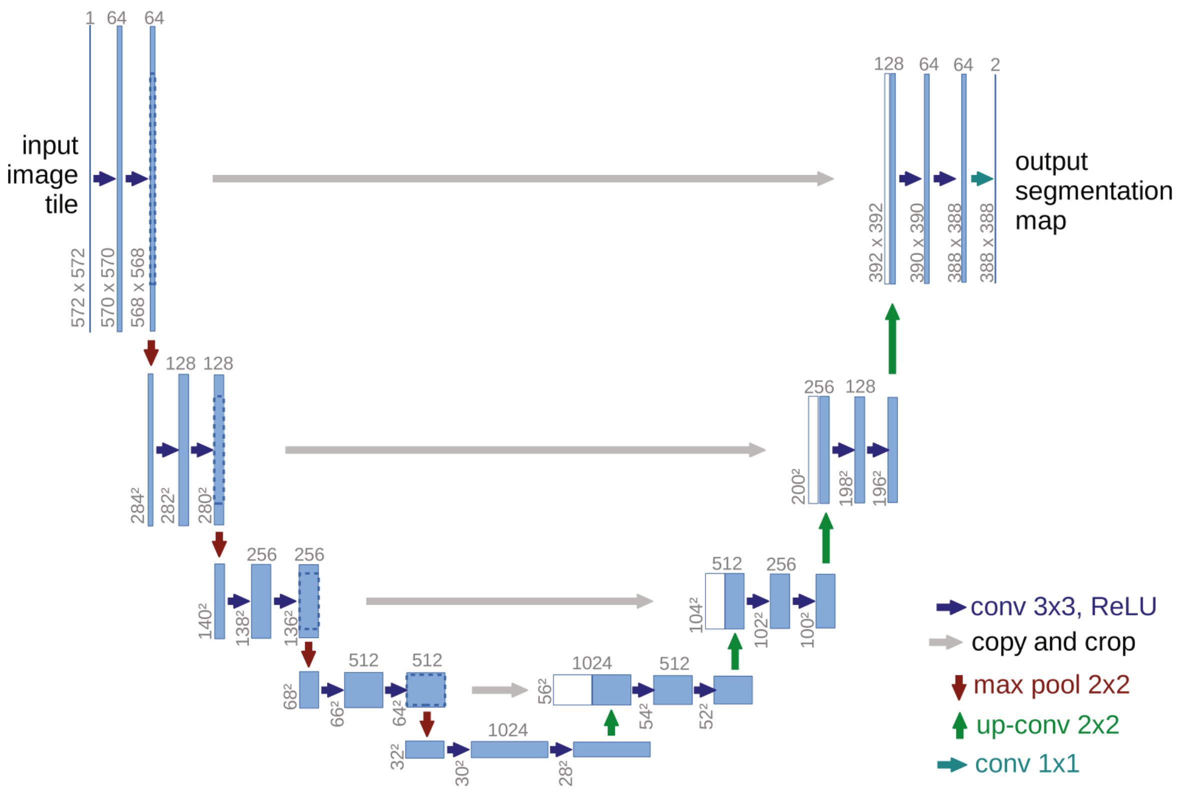


FIGURE 2.12: U-Net architecture (source: [167]).

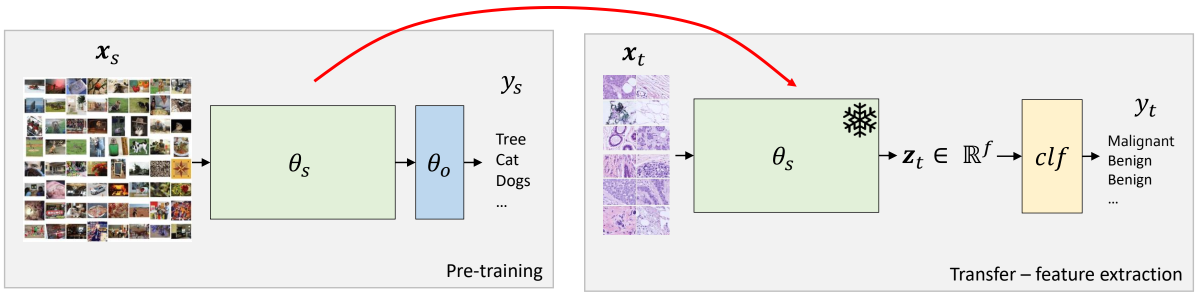


FIGURE 2.13: Supervised transfer learning by *feature extraction*. The shared network is trained by supervised learning on the source task. Then, the shared part θ_s is extracted and used as an encoder for images of the target task. A third-party classifier clf can be trained and used to classify the encoded target data.

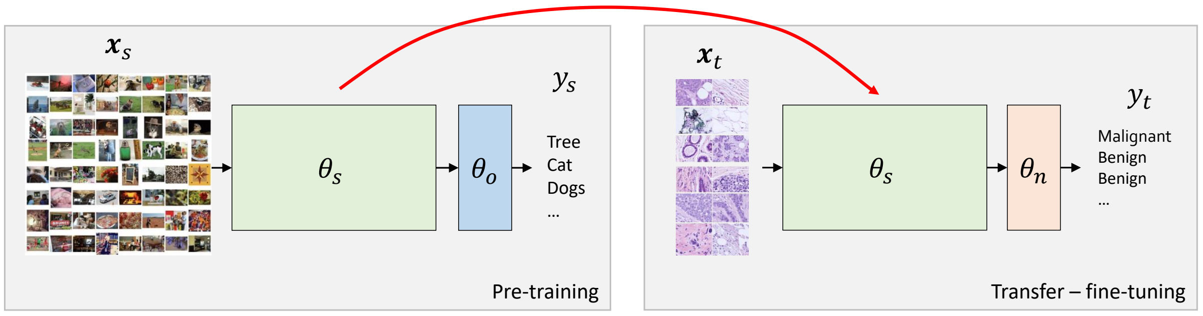


FIGURE 2.14: Supervised transfer learning by *fine-tuning*. The shared network is trained by supervised learning on the source task. Then, the shared part θ_s is extracted and attached to θ_n. The network $h_n \circ h_s$ is trained by gradient descent on the target task.

$\mathbf{z}_i = h_s(\mathbf{x}_i; \theta_s) \in \mathbb{R}^f$ where f is the number of features. The encoder is said to be *"frozen"* as it is not updated after transfer. The generated features can be used to train a third-party classifier on the target task which can be as simple as a linear model. A common choice of classifier is a linear SVM.

The second transfer approach is *fine-tuning* (see Figure 2.14) where the task-specific part of the network θ_o is replaced by a new classification network θ_n and the resulting network $h = h_c \circ h_s$ is further trained by gradient descent on the target task. Subjecting the whole network to gradient descent will change the features learned on the source task and can cause catastrophic interference. Forgetting what the model has previously learned might not be an issue if this knowledge is not useful and prevents the model to reach the best possible performance on the target task. However, this phenomenon can also cause features that are good for both the source and target tasks to be forgotten which is undesirable. Therefore, a way of alleviating the issue is to use a small learning rate (compared to the initial training learning rate). Another way of preventing this issue consists in freezing a part of θ_s, usually the first layers.

The feature extraction approach only emerged during the last decade with works on the Overfeat [179, 162] and Decaf [49] convolutional architectures. The authors trained their AlexNet-based networks in a supervised manner on the large ImageNet source dataset composed of 1.2 millions natural images and 1000 distinct classes. They both showed that these networks were able to produce discriminative features for tasks they were not trained on.

Building upon these results, Yosinski et al. [228] analyzed the transferability of deep neural networks using both approaches in more details. Their most prominent conclusion is that features in a neural network become less transferable as one moves deeper in the network indicating that the network features shift from generic to specific along the network. This result is consistent with Zeiler and Fergus [233] as early layers typically learn classical computer vision filters. This motivates freezing only the first layers of the network to avoid catastrophic interference as they are more generic and therefore should not require much tuning. They also studied how task similarity impacted transfer performance and found that similar tasks indeed benefited better from transfer than dissimilar ones. This was confirmed in Mensink et al. [137] who showed that for a variety of tasks, there exists a source task that outperforms ImageNet, especially when this task incorporates in some ways the target domain. More recently, Kornblith, Shlens, and Le [105] showed that there is a correlation between model performance on the source task (in this case ImageNet) and transfer performance on the target task.

Whereas the versatility of ImageNet initially fueled research in deep transfer learning with supervised model pre-training, several recent contributions have shown that pre-training could be performed differently. For instance, SimCLR [39] uses contrastive learning to learn the model. This is a self-supervised approach which consists in generating pairs of similar and dissimilar images using data augmentation and training a network to discriminate these pairs. They have shown that the resulting network has learned discriminative features that can be transferred quite efficiently to new tasks. The *vision transformer* (ViT) [51] is another approach based on transformers, a recent and popular family of attention-based methods. Attention is a mechanism of selection of information which can be implemented in different ways in neural networks [147]. It has initially been applied with great successes on *natural language processing* (NLP) problems but ViT has shown that attention-based architectures were also applicable to vision. In this framework, the transfer process is very similar to supervised pre-training as the model is first trained in a supervised manner on a large database, then transferred to the target task. The only difference is that the attention-based network does not use convolution and takes as input a sequence of non-overlapping patches of the image. Other contributions focus on improving classical supervised pre-training approach. For instance, Wang et al. [216] combine transfer learning and model compression as the fine-tuning process excludes non-informative feature maps from the final model based on their *attentive feature distillation and selection* (AFDS) mechanism.

2.6.5 Multi-task learning

Multi-task learning, introduced in Section 2.2.6, has been applied with great successes for a wide-range of application [238] such as image and video understanding [101], natural language processing [45], face recognition [241, 37, 239, 160], information retrieval [125], art classification [194], *etc.* The success of multi-task learning is notably due to the fact that leveraging several tasks and/or datasets alleviates the need for large amounts of data. Moreover, training in multi-task has a regularization effect preventing the model to overfit a particular task therefore yielding a better generalizable model. The modularity of neural networks also allows to embed multi-task specific components hence facilitating its application to deep learning [33, 238]. There exist many ways how multi-task learning can be implemented within deep learning with, for instance, architecture tricks [139, 193] and weights sharing [33].

2.6.6 Self-training

As introduced in Section 2.2.3, self-training is a family of semi-supervised learning methods. Self-training is an iterative process where a typical iteration consists in using a teacher model to produce pseudo-labels for unlabeled samples from $\mathcal{D}_u$ which are then combined with labeled samples from $\mathcal{D}_l$ to train a student model.

There are several elements that characterize a self-training algorithm. An important element is the way the student and teacher models interact during the process. A common approach consists in using the current student as a teacher for the next training round, during which a new student is trained from scratch [226, 222]. A different approach consists in building a teacher of which the model parameters are a moving average of the student parameters at the end of each training round [205]. Pham et al. [155] update both the student and teacher networks by back-propagation during the self-training round.

Another characteristic element is how the algorithm exploits the pseudo-labels. Some techniques use pseudo-labels *uncertainty* to select which samples should be included for the next training rounds by excluding high entropy samples for instance [69, 116]. Another way of exploiting the pseudo-labels in the training signal is to exploit *consistency*. Several self-training approaches ensure consistency between the predictions of the teacher and of a noisy student [222, 242, 186, 205]. The student is said to be noisy because of the stochastic nature of the data augmentation and network training (*e.g.* caused by dropout, stochastic depth). Laine and Aila [112] propose a variation of this by enforcing consistency between the current model and the pseudo-labels generated and aggregated over the past training rounds (so-called temporal ensemble).

2.7 Wrapping up

In the previous sections, we gave an overview of different topics. We can now position our work in this context. In Chapters 4 and 5, both contributions explore heterogeneous model-based transfer learning by transferring deep learning classification models. We use several target tasks to study how transfer learning performs in the context of digital pathology (more on this in Chapter 3). The first contribution (see Chapter 4) studies transfer from ImageNet. Motivated by the fact that transfer works better when the source and target tasks are related, the second contribution (see Chapter 5) uses homogeneous feature-based multi-task learning as a way to pre-train a model for transfer on digital pathology data directly. In Chapter 6, we move to a different type of task: image segmentation. We investigate a semi-supervised learning approach leveraging self-training to train U-Net segmentation model from sparsely-labeled digital pathology datasets. The model is used to complete the unlabeled area which are then included in the training set.

Before diving into these contributions, we review in the next chapter relevant background related to our application domain, digital pathology. This chapter also extends our presentation of related works covering similar methods of transfer learning, multi-task learning and self-training (see Sections 3.4.4, 3.4.5 and 3.4.6 respectively) applied in the context of medical and pathology image analysis.

3 Chapter Digital pathology

Overview

This chapter presents an overview of digital and computational pathology from the perspective of a computer scientist and aims at providing basic understanding of what makes computational pathology a challenging but promising topic. Section 3.2 presents a typical histological glass slide preparation procedure, how this glass slide is transformed into an image and what could go wrong during the process. Section 3.3 discusses briefly how slides are used and analyzed by practioners and provides three use cases where computational pathology could greatly help pathologists in their day-to-day work. Finally, in Section 3.4, we present some of the challenges specific to the application of machine learning to computational pathology including data leakage and data scarcity and how to tackle them. This last section also presents the related works of our contributions.

3.1 What is digital pathology?

Nowadays, medicine and healthcare rely heavily on analysis of body samples to study and diagnose diseases. The branch of medicine focusing on this analysis is called *pathology* which includes histology-based pathology and cytology-based pathology (*a.k.a.* histopathology and cytopathology respectively). Both of these sub-branches involve the study of microscope glass slides containing samples (see Figure 3.1). Histology samples are tissue sections cut from a bodily specimen. Cytology is concerned with samples of free cells or tissue fragments.

The trend of digitalization affecting our societies also impacts pathology as, using dedicated scanners, a glass slide can now be digitized into large image file called a *whole-slide image* (WSI). These files associated with subject metadata are stored in computer systems commonly called picture archiving and communication system (PACS) or laboratory information systems (LIS). In this context, *digital pathology* (DP) can be defined as "*the acquisition, management, sharing and interpretation of pathology information - including slides and data - in a digital environment*" [50]. Working with WSI instead of physical slides has several advantages and drawbacks. Aside from easier sharing and storing of slides, digitization also opens the way for automated analysis

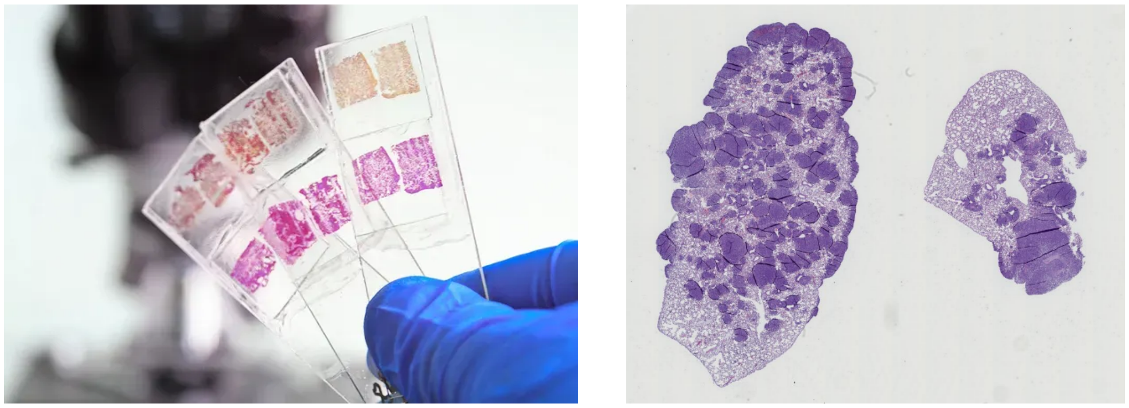


FIGURE 3.1: *Left:* microscope slides with tissue samples (source: [44]). *Right:* a whole-slide image of dimensions 30720 x 25600 pixels.

sofware to automatically extract relevant information, typically with the help of artificial intelligence and machine learning. The branch of digital pathology interested in such analysis is called *computational pathology* (CPATH) and holds great promises for the future. Indeed, computational pathology techniques have the potential to relieve pathologists from easy but time-consuming tasks allowing them to focus on challenging cases and research therefore reducing healthcare cost and improving diagnosis quality. On a larger scale, they hold promises for increased coverage and quality of healthcare around the world and especially in low incomes countries where the number of pathologists per inhabitant is typically insufficient. According to the WHO cancer report [220], the ratio of pathologists per inhabitant was approximately 1 per 15000 in high income countries in 2020 but dramatically drops to 1 per million or less in many low income countries. A throrough list of advantages and drawbacks of digital pathology can be found in Table 1 of [84].

Interestingly, although digitization technologies are quite mature, adoption of digital pathology in healthcare facilities is not simple. Many still heavily rely on glass slides for day-to-day operations. Indeed, the transformation requires to modernize the whole hardware (scanners, workstation) and software (slide viewers, information system) infrastructures and to re-think entirely the processes of the facility [191, 53, 207]. This obviously requires significant investments both in time and money and careful planning to carry it out successfully which is not always compatible with the workload of pathology services or research laboratories. Other difficulties might arise, slowing down the transformation, such as reluctance to change and lack of confidence in modern tools for slide visualization and analysis.

As far as automated analysis is concerned, it remains quite a challenge. Whole-slide images typically contain several billions of pixels at full resolution which implies longer processing times and memory issues compared to classical images. Moreover, for most tasks, the image content is complex and traditional computer vision methods (*e.g.* thresholding) would often fail to distinguish structures of interest. This complexity is increased by the presence of artifacts [204] appearing during the conversion process of a bodily specimen to an image. An artifact is a visual or physical alteration of a sample that can hamper its analysis, automated or not. Artifacts introduce a source of variability that algorithms must learn to deal with. At worst, they can prevent any meaningful analysis by hiding, destroying or changing the appearance of the structures of interest (more on artifacts in Section 3.2). The ability of learning techniques to train models that capture complex relationships in data makes machine learning an ideal candidate to tackle computational pathology tasks. However, data scarcity is a prevalent issue in the field as quality data, especially annotated, can be difficult to obtain for various reasons: privacy concerns, time-consuming and expensive nature of the annotation process, *etc*.

Overall, digital pathology holds great promises but presents significant and interesting challenges on several fronts. This thesis focuses on the ML-based automated analysis aspects of computational pathology and studies how to tackle data scarcity in particular.

3.2 A journey from the body to the computer

Turning a bodily specimen into whole-slide images is a long and complex multi-step process typically involving the work of several highly-specialized technicians and machines. Some steps can nowadays be automated but the chain remains mostly manual. In this section, we describe the different steps of this procedure which is summarized in Figure 3.2. An alternate presentation of the process can be found in [134] (including illustrations). Sample preparation can differ more or less dependending on the nature of the sample (*e.g.* histology, cytology, hematology) or target imaging technique (*e.g.* brightfield, fluoresence, multispectral). For the sake of brevity, our description focuses on histology with a tissue section prepared for brightfield microscopy and scanning, brightfield being one of the most common modalities used in histo- and cytopathology. We will also present a few technical details related to WSI files structure and visualization.

Throughout the section, we will discuss some of the possible artifacts resulting from the transformation procedure. Our presentation of artifacts will not be exhaustive and few visual examples can be found in Figure 3.3. A more thorough list of pre-scan artifacts with illustrations can be found in [204].

3.2.1 Specimens collection, fixation, cutting and dehydration

Whether it is for research or diagnosis, the slide preparation process starts with a specimen, a piece of human or animal body for which a question must be answered. The specimen can be as large as a whole organ but can also be as small as a drop of bodily material commonly referred to as a *biopsy*. Before going through the preparation process, a specimen must be fixated. The goal of fixation is to put a stop to the natural decay of the specimen and increase its structural stability [166]. This can be achieved, for instance, by immersing the specimen in a formaldehyde bath (*i.e.* the fixative solution) for period of time depending on its size (*i.e.* few hours to a whole day).

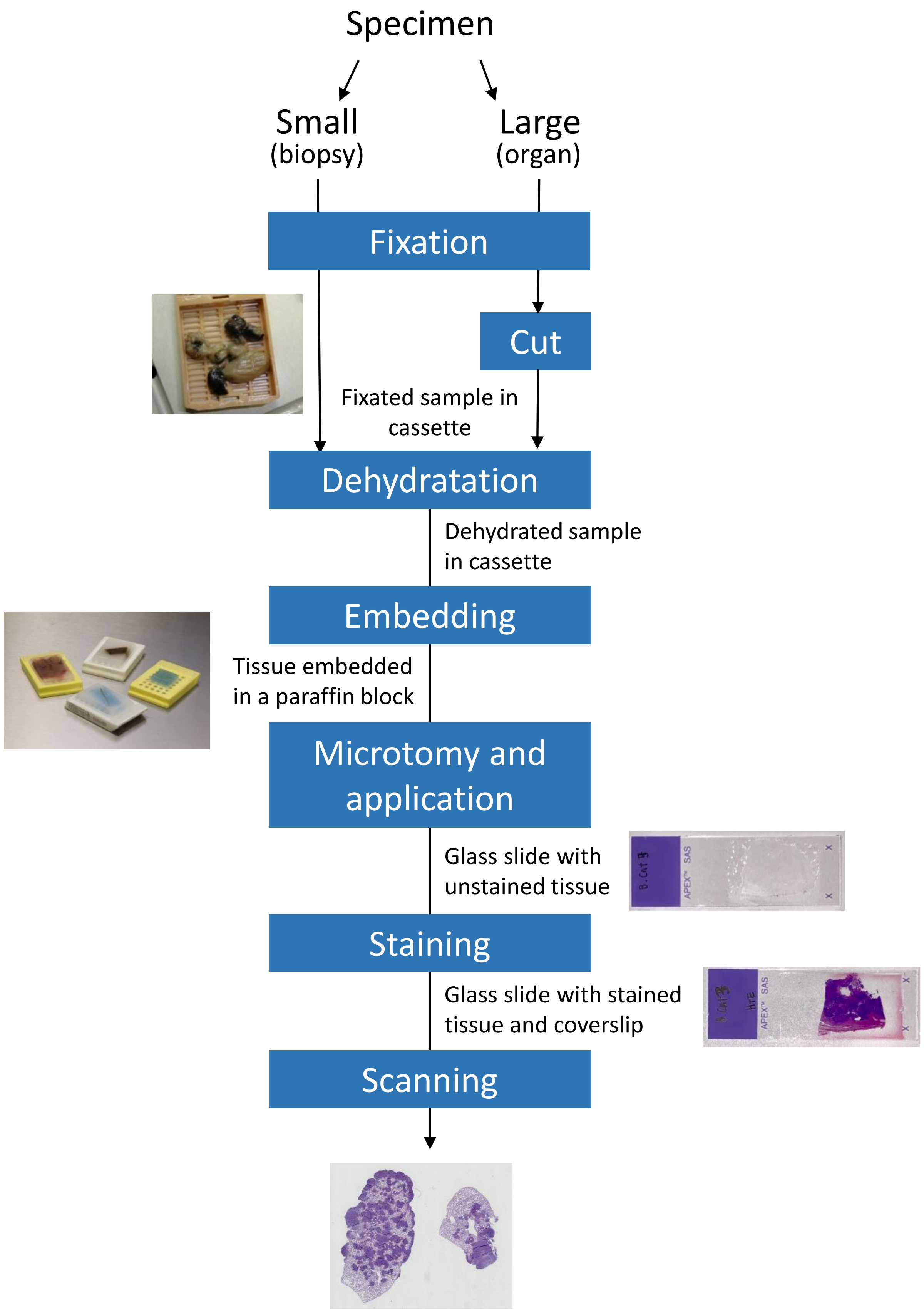

Specimen
Small
(biopsy)
Large
(organ)
Fixation
Cut
Fixated sample in cassette
Dehydratation
Dehydrated sample in cassette
Embedding
Tissue embedded in a paffin block
Microtomy and application
Glass slide with unstained tissue
Staining
Glass slide with stained tissue and coverslip
Scanning

FIGURE 3.2: Summary of the transformation of a specimen into a whole-slide image.

When the specimen has been fixated, it is then placed into a standardized container called a *cassette* (see Figure 3.4a). If it is too large for the cassette (*e.g.* an organ), one or more volumes of interest are cut from the specimen. Depending on the later examination, the orientation of the cut can be crucial to exhibit relevant tissue structures of the specimen. In the remainder, we will call a *sample* the content of this cassette.

For a proper analysis, the tissue morphology of the sample must be preserved. This is most commonly achieved by infiltrating the tissue with paraffin wax. Infiltration however does not work on a raw fixated tissue because paraffin is hydrophobic. Therefore, one must first perform *dehydration*, that is, replacing water naturally present in the sample with a product miscible with paraffin. This is done by first immersing the sample into a succession of alcoholic solution baths. Although this process achieves dehydration, alcohol does not mix with paraffin neither. Therefore, the sample is then immersed into one or more xylene-based solutions baths, xylene being miscible with both alcohol and paraffin. The sample, infiltrated with xylene, is finally immersed in a paraffin bath under vacuum. The dehydration process takes few hours and is often automated using dedicated machines.

The earliest source of artifacts is the specimen extraction process itself. The specimen can indeed be damaged by the use of certain tools (*e.g.* burned by an electrical scalpel) or treatment at the extraction site. Unlike these, the following artifacts are caused by the early stages of the slide preparation process. Bad fixation can lead to decaying tissue (*i.e.* autolysis, see Figure 3.3a) and structural degradation (*e.g.* tissue shrinkage, see Figure 3.3b). Improper cutting can also cause tissue damage like tearing and squeezing. Improper dehydration can leave some parts of the sample with remaining water, alcohol or xylene. Tissues can also be exposed to the different solutions for an excessive duration. These processing errors can for instance cause tearing, shrinkage, interference with the staining process (see Section 3.2.3) and affect the structural properties of the tissue (*e.g.* tissue becomes brittle).

3.2.2 Embedding, microtomy and glass-slide application

At this point, the sample in the cassette has been infiltrated with paraffin. The next step consists in embedding the infiltrated sample in a block of paraffin to allow easier cutting. The sample is placed in a small container which will serve as a mold for casting the block of paraffin. The cassette is then directly placed on top of the container so that, when the block solidifies, it is attached to the back of the cassette (see Figure 3.4b). When it has indeed solidified, the sample can now be cut into thin slices to be applied on the glass slides. Cutting is performed with a dedicated tool called a *microtome* (see Figure 3.5). Operated by a technician, the microtome allows slices to be cut to an extremely small and precise thickness of around 3 or 4 μm. The slices are then floated onto a water bath which helps mounting them on glass slides. Although there exist equipment that automate the embedding and microtomy steps, to the best of our knowledge, they are not widespread and these steps are still mostly performed

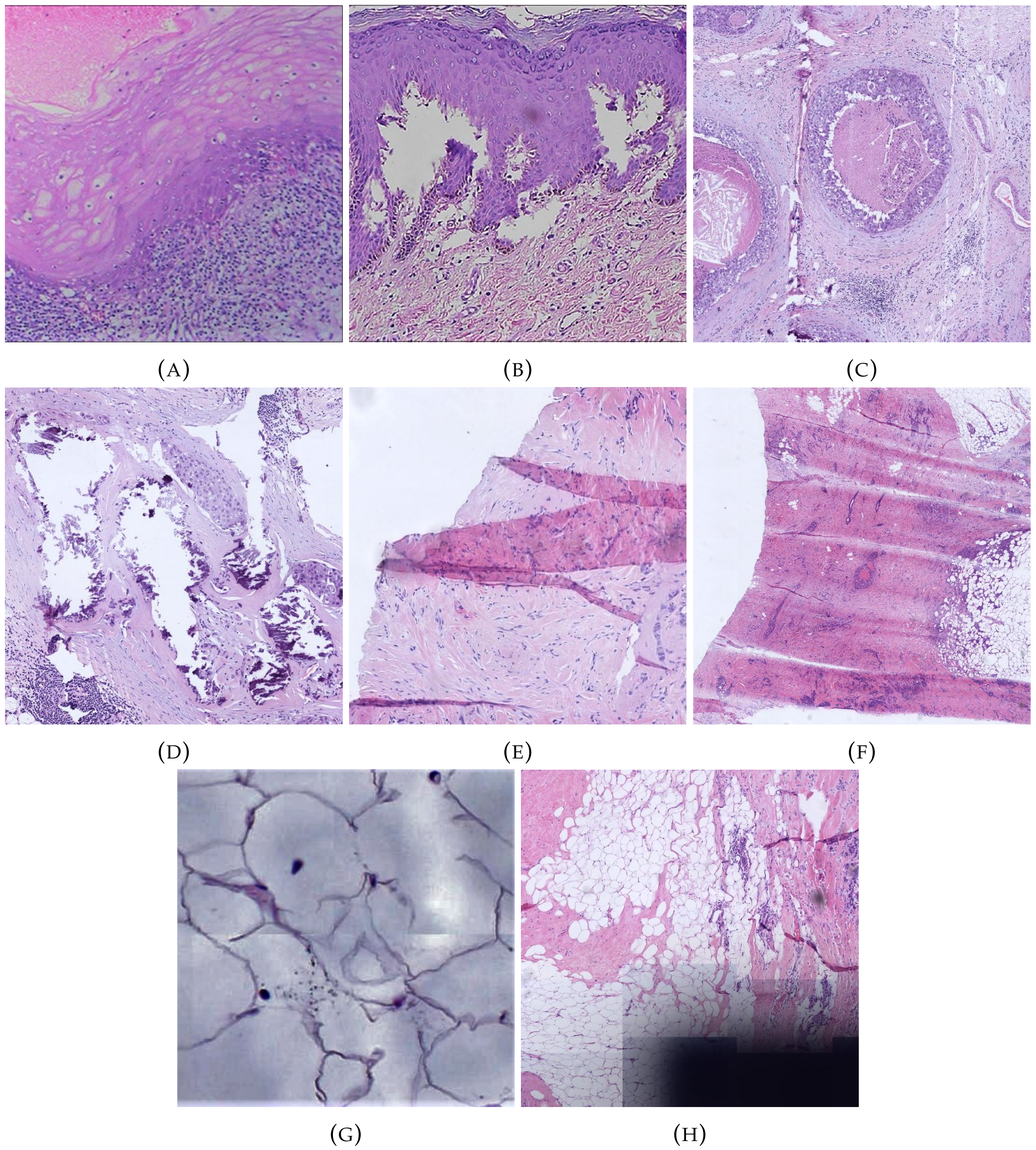
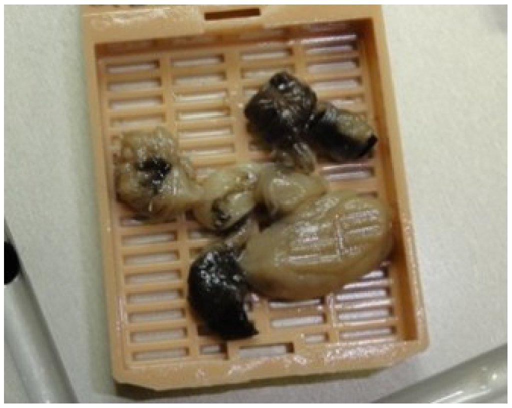


FIGURE 3.3: *Autolysis artifacts*: 3.3a (poor cellular differentiation), 3.3b (tissue separation). *Microtomy artifacts*: 3.3c (nick or blemish in the blade), 3.3d (calcification pushed through the tissue by the blade). *Folding artifact*: 3.3e. *Staining artifact*: 3.3f. *Scanning artifacts*: 3.3g (stitching issue), 3.3h (scanner failed to capture part of the tissue) (sources: 3.3a, 3.3b [204]; others from Cytomine).


(A) Cassette with fixated samples (source: [192])

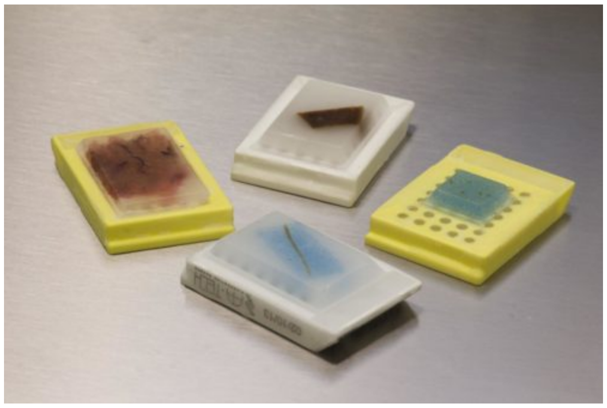

(B) Cassette with paraffin-embedded samples (source: [22])

FIGURE 3.4: Tissue cassettes.

by technicians. Even if operated manually, modern microtomes are equipped with automation and ease-of-use features to improve ergonomy and convenience, hopefully improving consistency (*i.e.* slice thickness, *etc.*).

These steps should be performed carefully not to introduce artifacts. For instance, a warm and soft paraffin block, a dull microtome blade (see Figure 3.3c) or a calcification in the tissue can cause compression artifacts (*i.e.* tissue displacement causing material accumulation). A calcification is solid calcium deposit present in the tissue that the blade cannot cut through. This deposit is therefore pushed through the tissue, compressing underlying tissues (see Figure 3.3d). Another possible source of artifact is contamination of the water bath with previous samples, hair or dust which can in turn contaminate the floating slices.

3.2.3 Staining

Mounted tissue slices are almost completely transparent which would prevent any meaningful analysis. They must therefore be stained to highlight structures of interest (see Figure 3.6). Similarly as for dehydration, this process consists in immersing the slide into a succession of stainning solutions. The nature of these solutions will depend on the content that should be highlighted for the future analysis. The most common and standard staining in histology is called *hematoxylin & eosin* (H&E). Hematoxylin stains nucleic acids in a deep blue-purple color (typically cell nuclei) and eosin non-specifically stains proteins in a pink color (typically extracellular matrix and cytoplasm). The H&E stain, although most common, is not the only one available. There exist many other staining techniques such as *immunohistochemistry* (IHC) which exploits the binding nature of some antibodies with specific proteins. Markers can then be used to highlight the antibodies, hence the proteins of interest they have bound with. Because the IHC staining can be very selective, a counterstain (*e.g.* hematoxylin)

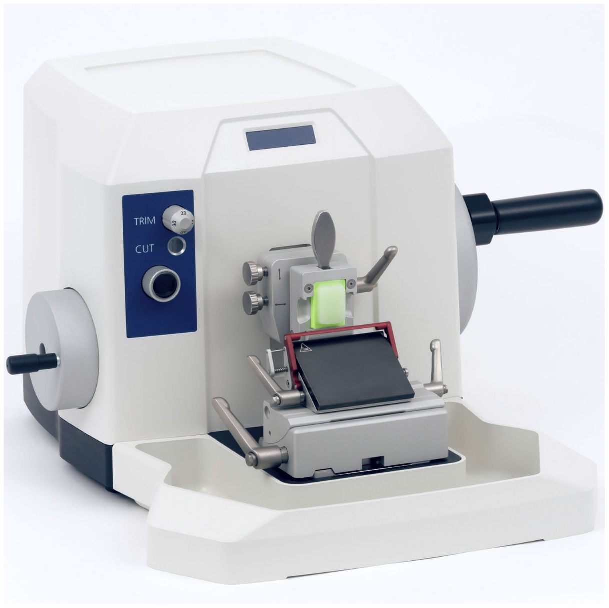


FIGURE 3.5: A microtome.

is often applied to highlight the rest of the tissue. Counterstain is particularly important when different slices of a tissue stained with different techniques must be compared together because it helps matching the slices spatially. An example of a tissue stained with H&E and IHC is given in Figure 3.7. When the sample has been stained and cleaned from remaining excess of staining solutions, one must apply a cover slip on the sample in order to ensure that sample lies in one single plane as the focal plane of microscopes and scanners is usually quite narrow. The cover slip also protects the sample from external contamination and degradation.

The choice of a staining and its clean application are crucial for an efficient analysis. The staining baths can become contaminated with samples, by-products resulting from chemical reactions (*e.g.* precipitation or crystallization of chemical components resulting in the presence of pigments in the sample) or external objects (*e.g.* hair, dust). The baths tend to degrade over time as moving slides from one bath to another transfer some staining solutions as well. The degradation of the staining solutions can cause variation in staining intensities between earlier and later samples. The bath duration is important for proper staining and bathing samples for less or more time than recommended can respectively cause under- or over-staining (see Figure 3.3f). Moreover, insufficient cleaning after staining can leave spots of stain on the slide. Improper application of the cover slip can for instance cause the presence of air bubbles.

Nowadays, the staining process can be automated with automated slide stainers. These systems move the slide automatically from a bath to the next ensuring stable dipping durations for each stain.

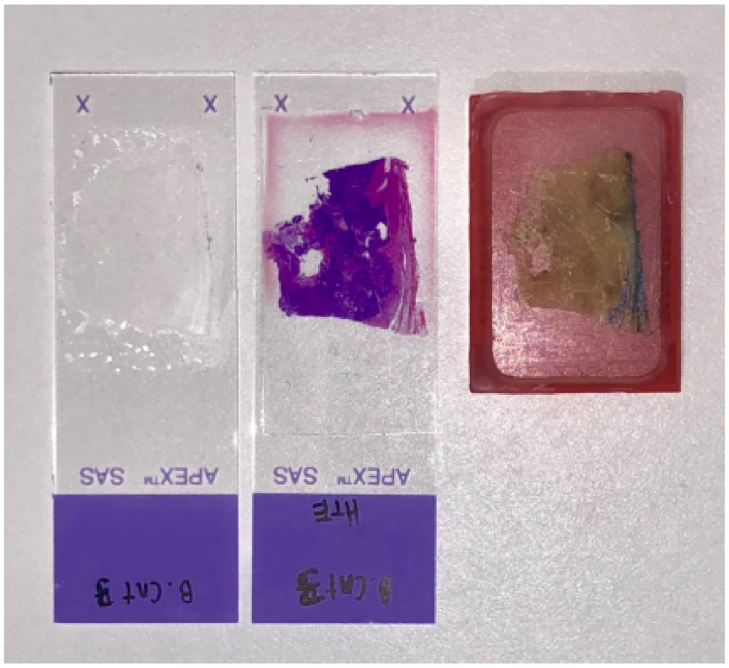


FIGURE 3.6: *Left:* an unstained tissue section mounted on a glass slide. *Middle:* H&E-stained section coming from the same tissue. *Right:* the source tissue embedded in a paraffin block (source: [1]).

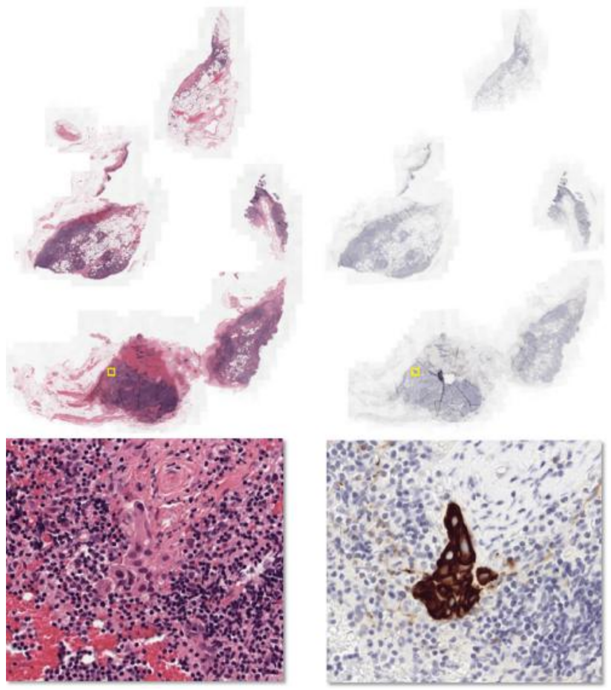
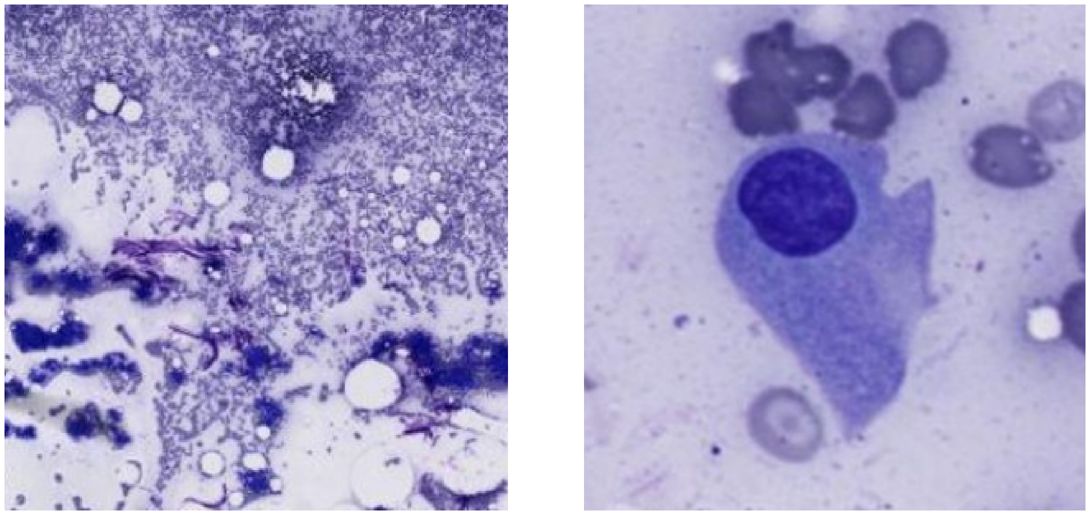


FIGURE 3.7: Two slices of the same tissue stained with H&E (left) and IHC (right) (source: [123]).



(A) ×1.25, 7.24 μm/pixel (B) ×40, 0.226 μm/pixel

FIGURE 3.8: Two images (approximately 200 × 200) extracted from the same WSI at different magnification and resolution (reported in subcaptions). They are both centered on the same cell.

3.2.4 Scanning

A glass slide coming out of the staining process described in Section 3.2.3 is ready to be analyzed with an optical microscope but can also be digitized by a slide scanner into a WSI. In this section, we will briefly describe few key elements related to slide scanning. A more thorough technical presentation and discussion of scanning technologies can be found in [150].

Scanners are equipped with high-precision lenses and sensors allowing them to generate very high-resolution images required for a proper analysis. Magnification and resolution are two popular metrics used to describe the visual quality of the resulting image. Magnification advertised by scanner vendors refers to the maximum size of the objects in the scanned image with typical values being ×20 or ×40 (*i.e.* objects appear 20 or 40 times larger than they actually are). Resolution determines the extent to which smaller objects can be resolved. Resolution is reported for a given magnification level and often lies around 0.25 μm/pixel at ×40 for modern scanners but higher magnification and resolution are possible. These magnifications are standard and, coupled with an adequate resolution, are usually sufficient for routine analysis of H&E or IHC slides [232].

Given the need for high magnification and resolution, it is not possible to capture the whole slide in a single shot with currently available sensors. Therefore, scanners typically capture a sample step by step either tile by tile or in an in-line fashion and then assemble together the different parts using a stitching algorithm. Obviously, as a slide is scanned in several shots, the scanner must ensure that focus is correct for all the shots in order to avoid blur. Common focus strategies are for instance re-focusing every tile, or every n^{th} tile. When it comes to the focus strategy, there is usually a trade-off between scanning time and focus precision: focusing every tile allows for ideal focus but takes time, whereas focusing every n^{th} is faster but incurs a risk of incorrect focus and blur between the re-focusing steps. Modern scanners implement efficient strategies to reduce the scanning time per slide which nowadays ranges from 30 seconds to several minutes (at $\times 20$ or $\times 40$ magnification). These strategies include improved focus strategies and automatic tissue detection allowing to skip the empty parts of the slide during scanning. Most scanners also allow to load batches of few hundreds of slides at once therefore reducing the time spent by the operator interacting with the machine.

For histology, it is usually sufficient for scanners to scan one focal plane of the slide (*i.e.* that of the tissue slice). For cytology, however, it is not always enough. Indeed, the slide preparation process for a cytology sample differs and the material is not always aligned within a single focal plane. With an optical microscope, it is possible to navigate continuously over the focal planes by adjusting the lenses positions. When it comes to scanning such samples, one has to resort to a technique called "*z-stacking*". It consists in capturing the material at different depths along the z-axis. This process can be significantly longer than single focal plane scanning, as it multiplies this time by the number of slices of the stack. The result is a finite number of images which can be viewed in different ways (*e.g.* one by one, grid view, as a video sequence, *etc.*) but does not offer the simplicity or continuous exploration provided by optical microscope for this task.

Scanning can also introduce artifacts including stitching problems (*i.e.* misalignment between scanned tiles or lines, see Figure 3.3g), blur due to incorrect focus, tissue detection failure causing parts of the slide to be missing from the WSI (see Figure 3.3h). The scanner should obviously remain as clean as possible to avoid external components to pollute the image (*e.g.* dust, glass shards, hair, *etc.*).

3.2.5 File formats and compression

For each glass slide, a scanner generates one or more files to store the image data but also any metadata related to the case (information or identifiers of laboratory, patient, specimen, staining, *etc.*). How the data is organized in such a file is specified by a *file format* of which there exist many. Some file formats are closed and proprietary (*e.g.* scanner- or vendor-specific file formats) but others have open specifications (*e.g.* DICOM, OME-TIFF). The most involved formats usually combine a descriptive part to store case-related metadata using, for instance, the XML language (*e.g.* in OME-TIFF) and a subfile format for the image itself (*e.g.* TIFF).

Regarding the internal structure of the latter, the image is usually splitted into a set of tiles (*e.g.* 1024×1024) rather than being stored as a single image array. The file contains metadata to provide efficient access to these tiles. This organization makes sense because, in practice, one rarely has to load the whole image in memory at full resolution at once which, given the size of a raw image (*e.g.* Figure 3.9), would be impossible on most computers anyway. It is however a common use case for a practitioner to look at an entire slide (or a large region of interest) at a lower magnification and resolution.

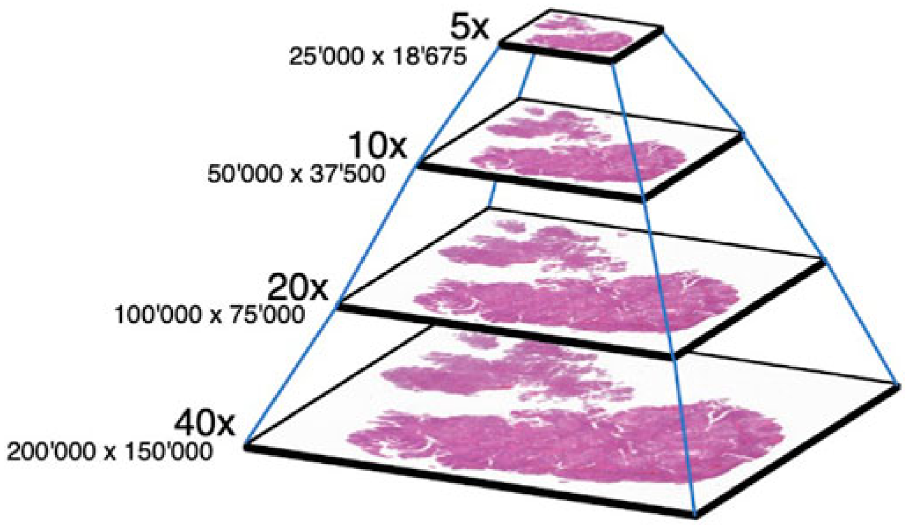


FIGURE 3.9: Pyramidal view of a whole-slide image (source: [133]).

Downsizing, downsampling and aggregating tiles to obtain a low resolution view of a slide (or a region of interest) is an expensive operation and cannot realistically be performed on-the-fly without significantly increasing loading times. Therefore, a typical image files also stores versions of the image at different zoom levels. The i^{th} zoom level ($i \in \mathbb{N}_0$) is a version of the image which has been downsized by a factor 2^i (or 3^i, 4^i depending on the image format). The number of zoom levels varies depending on the size of the image at full resolution and is such that the image at the lowest zoom level (*i.e.* largest i) has a reasonably low size (*e.g.* when it fits in a single tile). Similarly as for zoom level 0, other zoom levels are stored as set of tiles. Such a file is called *pyramidal* because of this structure (see Figure 3.9).

A whole-slide image file is not a lightweight one. For example, a raw $10^5 \times 10^5$ RGB image (3 bytes per pixels) contains 30 gigabytes of information. This is massive and not scalable if one has to consider the multitude of slides to be scanned and stored by an hospital for instance. Therefore, compression algorithms are very frequently used to reduce the size of image files. Popular choices are JPEG (lossy compression) and JPEG2000 (lossless compression). Compared to its lossless counterpart, lossy compression allows for better compression rate and disk usage reduction. However, the loss of information results in visual alterations that become more and more severe as the compression rate increases. Therefore, lossy compression should be used carefully to avoid destroying image features relevant to the analysis.

3.2.6 Visualization and hardware considerations

An image file is not worth much if it cannot be viewed. Nowadays, there exist many open or proprietary software and tools for visualizing WSI. Some are desktop applications designed to run on personal computers and workstations (*e.g.* QuPath [15], ASAP [46]) and some others are directly provided by scanner vendors to run on dedicated hardware alongside the scanner. Some others are web-based (*e.g.* Cytomine

[132], OMERO [3]) for which the viewer can be accessed through a web browser and content is served over the HTTP protocol. The most popular tools are not limited to a viewer but usually include many more features. For instance, Cytomine includes image annotation tools, integration of computer vision and machine learning algorithms, authentication, project-based content organization, an API to interact with data programmatically, *etc*.

Whereas it is included by-design in web-based tools, remote slide viewing is also possible with some desktop applications. This feature is particularly important in a laboratory environment. Without it, the system would need to download an entire WSI file before being able to display. This would add a significant latency in the slide reviewing process. Remote slide viewing allows for a seamless and more efficient interaction between the practitioner and the viewing system.

Beyond software, hardware considerations are important for efficient viewing, especially when implemented at the scale of a laboratory or pathology service [207]. Personnal workstations should be equipped with a quality screen able to render image without loss of quality (color and resolution) [165]. They also must be equipped with adequate computing resources (memory and CPU/GPU) in particular if analysis algorithms are supposed to be executed on them. The network should be robust enough to support the exchange of information (*i.e.* download and upload of large files and data). Servers should have enough disk space to store the digitized slides and data and implement mechanisms in order to minimize the risk of data corruption or loss.

3.2.7 Discussion

As presented in the previous sections, the preparation process has a significant impact on the final quality of a slide. Every step in the process introduces variability and has a chance of creating artifacts that can be both harmful to the analysis. Fortunately, automation reduces significantly the variability and makes the risk of artifacts acceptably low, although not absent. The dehydration, paraffin embedding and staining steps can all be automated with dedicated machines nowadays but microtomy remains mostly manual due to the high-precision requirements of the process. For scanned slide, there exist automated quality control tools such as HistoQC [87] or PathProfiler [73] to verify that a WSI has a sufficient quality to be considered for analysis.

The preparation process also impacts significantly the diagnosis time as overall, from fixation to scanning, it can take from 12 to 24 hours (sometimes more) to obtain a slide depending on the nature and state of the original specimen. There exist faster processes such as cryosectioning (*i.e.* the sample is frozen then sectioned) for intra-operative consultation during which a surgeon requests an analysis to decide how to proceed with a surgical operation. Such process can provide an answer within the hour but the resulting slides are usually of lesser quality than traditionally prepared histological slides.

3.3 Whole-slide analysis

When a slide has finally been prepared, a pathologist, a biologist or a technician can take over and start the analysis, whether on a computer screen or through the lenses of an optical microscope. There exist many active research domains which rely on slide analysis. Similarly, there exist many pathologies that can be discovered from the same source. For the sake of simplicity, the remainder of the section will focus on oncology and cancer diagnosis. In this field, one of the most frequent conditions that has to be evaluated is malignancy of tumors. The indicators of malignancy can vary greatly from a disease to another and even from one type of cancer to another. The approach of the slide by the pathologist and the analysis can therefore vary accordingly.

Two important tools for diagnosis and prognosis of cancer are grading and staging systems. A grading system is a set of mostly objective criteria for evaluating how abnormal cancer cells and tissues are compared to their healthy counterparts. Similarly, a staging system defines a set of criteria for quantifying the dimensions of the primary tumor and determining how far the cancer has spread in the patient's body. These criteria can be quantitative (*e.g.* cell counts, tumor area dimensions) or qualitative (*e.g.* cell or tissue morphology, presence of metastases) and consider macroscopic or microscopic elements. Beyond grading and staging, slide analysis can also provide direct information for establishing a proper treatment. For instance, localizing a malignant tumor in its support tissue can be important for guiding a future surgical operation.

All the types of tasks involved in slide analysis are not equal. They have varying degrees of complexity and tediousness. Some tasks are rather simple but take a significant amount of time (*e.g.* counting mitosis, see Section 3.3.1), others require strong focus for an extended period of time which increases the risk of error (*e.g.* thyroid nodule malignancy, see Section 3.3.3). Moreover, some diagnoses require to screen more than just one slide, in which case the volume of information to review can be significant.

When a task has to be performed frequently and is tedious, error-prone or involves reviewing a large amount of data, there is a great potential for computational pathology methods to improve analysis quality and speed. These methods can either provide quantitative or qualitative information that directly contributes to the diagnosis or simply assists the practitioner during the reviewing process.

Beyond automating existing tasks, computational pathology could also give rise to new approaches and standards for diagnosis. For instance, computational methods could be used to extract new descriptive features such as exhaustive statistics about cell types and structure in a sample or new cell morphology descriptors. This information could be used alongside non-imaging data and correlated to patient outcome [2].

In the longer term, new computational methods coupled with new imaging techniques could make obsolete the slide preparation process as it is performed now as well as the current diagnosis systems based on histology and cytology. Indeed, in the future, entire specimen could be scanned directly to produce 3D volumes with slide-free microscopy techniques [70, 126]. These image volumes could be analyzed directly by computational methods to provide fast and hopefully reliable indicators for pathologists to diagnose cases in a shorter time frame than ever before. However, it will take a significant amount of time before such technology is sufficiently developed, validated and becomes reliable enough to be used in clinical practice on a daily basis.

In the following sections, we provide some examples on diagnosis tasks that would greatly benefit from automation with computational methods. For each of them, we also provide examples of contributions of the computational pathology community to the automation of these tasks. The datasets presented in these sections were used in the contributions of the thesis.

3.3.1 Mitotic count

Mitosis is one of the phases of the cell cycle during which the chromosomes (replicated in a previous phase) are separated into two new nuclei. Tumor often exhibit a larger mitosis rate compared to healthy tissues. Many grading systems therefore use a measure of the mitosis rate as an indicator of cancer severity. For instance, the Bloom-Richardson system [171] for histologic grading of breast cancer includes mitotic count as one of its three indicators. In particular, mitosis must be counted in 10 fields of view. The final grade is determined based on a point system. Each indicator provides 1 (least severe) to 3 (most severe) point(s) which are added together (3-5 pts = grade I, 6-7 pts = grade II and 8-9 pts = grade III). For the mitosis indicator, 1 point is given if the mitotic count is 5 or less, 2 points if it is between 6 and 10 and 3 points if it is 11 or more[1].

Counting mitosis is cumbersome. It is no surprise that computational pathology methods have been investigated to automate the task. For example, in 2014 was organized the MITOS-ATYPIA-14 challenge [170] where participants had to come up with the best machine learning model to count the number of mitosis in different fields of view (see Figure 3.10). In addition to the mitotic count, they had to predict a global nuclear atypia score for each field of view, nuclear atypia being another indicator of the Bloom-Richardson system.

3.3.2 Breast cancer staging and sentinel lymph nodes

Axillary lymph nodes are structures of the immune system located near the armpit. They drain a large proportion of the lymph coming out of the breast. They contain lymphocytes and constitute a barrier that eliminates bacteria, viruses and other foreign particules (including metastases) from the lymph. Because they are the first recipient of metastases originating from breast tumors, these lymph nodes are an important element to consider when staging breast cancer. In particular, in the context of TNM, a standard internationally accepted cancer staging system, the pN-stage [163] quantifies cancer spread through the presence of metastases in regional lymph nodes. Its

[1]The recommended numbers might change based on the microscope used for the analysis.

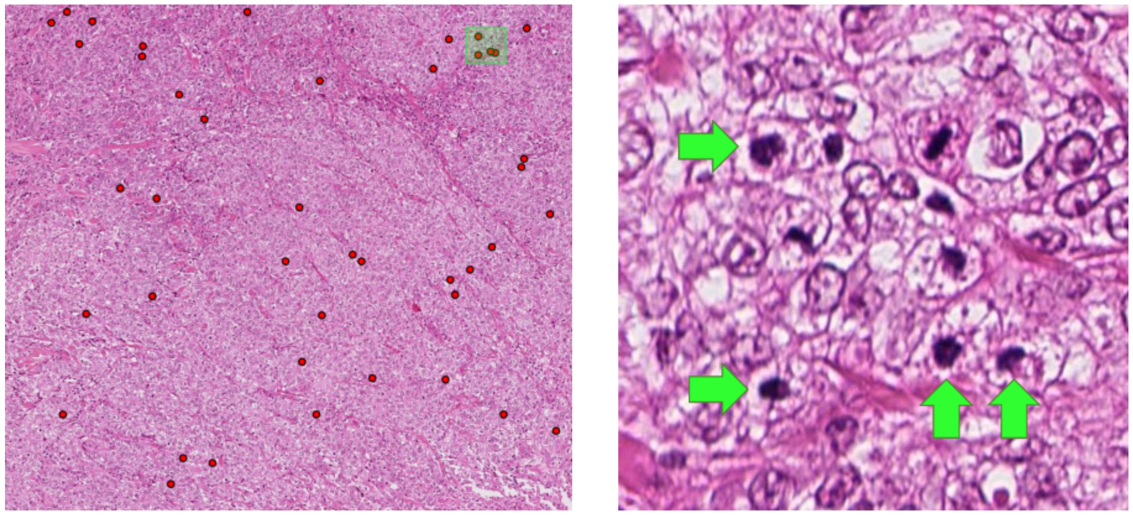


FIGURE 3.10: *Left:* a training image provided for the MITOS-ATYPIA-14 challenge viewed from the Cytomine viewer. Red dots are mitosis present in this image. *Right:* A close up view of four mitosis (in the green square from the left image).

histological evaluation requires to screen one section of up to 10 axillary lymph nodes and several sections of the "sentinel" lymph node which is the most likely to contain metastases [218]. Sometimes, additional slides counter-stained with IHC must be analyzed to confirm the diagnosis. The staging system requires to evaluate the number of nodes contaminated with metastases and the dimensions of these metastases (*i.e.* their size and the number of cells they contain).

Evaluation of the pN-stage has many characteristics that qualifies it to be a good candidate for automation: it is time-consuming, tedious and error-prone. The computational pathology community has therefore considered this problem notably through the Camelyon16 and Camelyon17 challenges [123]. For the first iteration of the challenge, the participants had to predict a slide-level pN-stage. For the second iteration, the organizers moved to a patient-level prediction. The participants were provided with 899 training and 500 testing WSIs which were collected in 5 different hospitals and medical centers from the Netherlands. All training slides were also coming with a slide level pN-stage label. Moreover, 209 of those WSIs contained detailed hand-drawn contours for all metastases (see Figure 3.11). The top-performing solutions typically used a combination of techniques and algorithms including traditional computer vision pre- and post-processing methods, deep learning to segment the metastases and rule-based or random forests methods to predict the final stage.

This challenge was a success as many teams participated despite the complexity of handling such a massive dataset (3 terabytes of slide data). The top-performing algorithms were quite successful at predicting the pN-stage. However, as stated by the organizers, the task was not adequatly solved and there was room for improvement [14] as a combination of the best algorithms still misclassified the pN-stage for 23 patients out of 100.

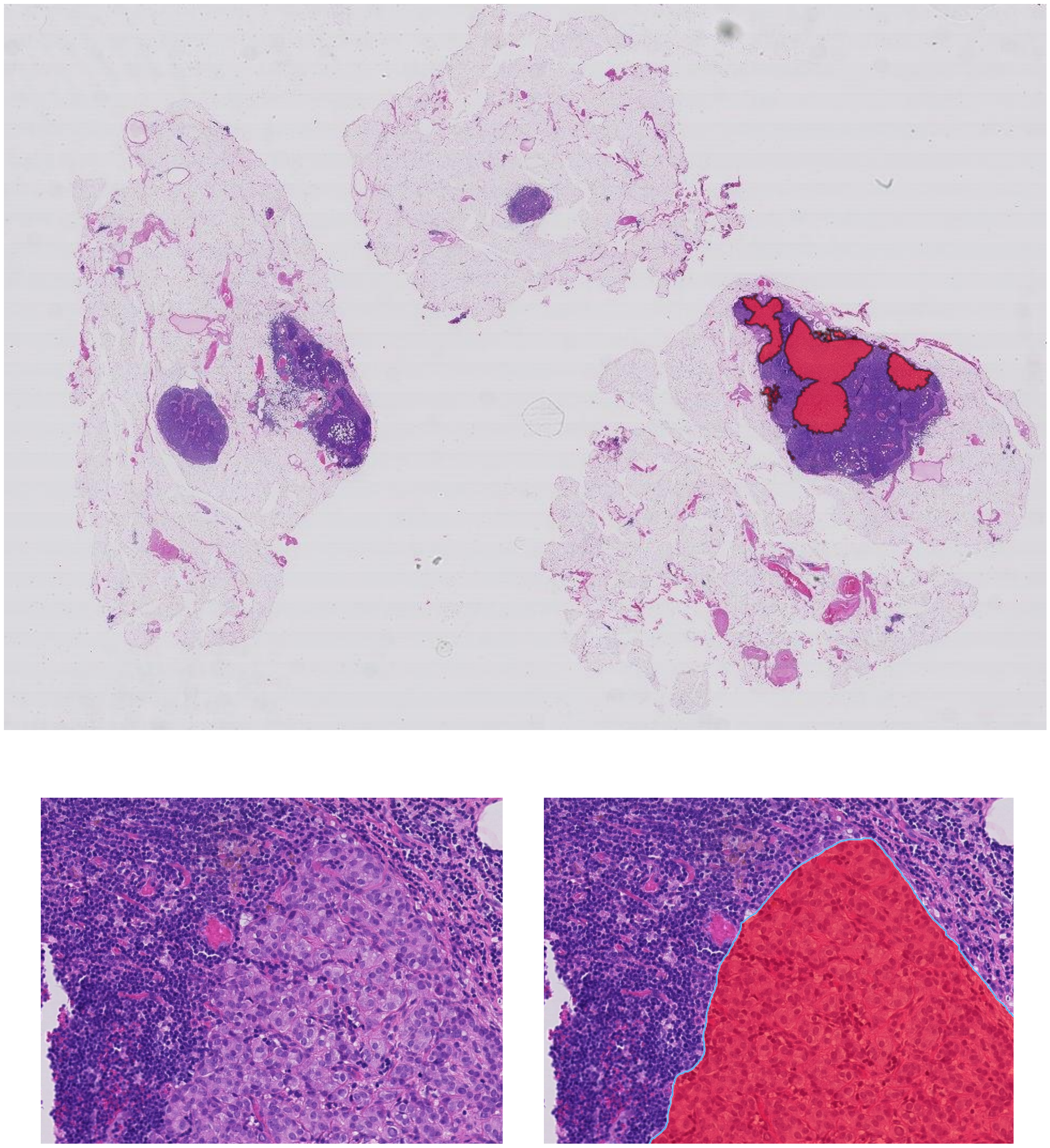


FIGURE 3.11: A WSI from the Camelyon16 challenge training set viewed from the Cytomine viewer. *Top:* the whole-slide image with metastases highlighted in red. *Bottom:* A close-up view of a metastase with (*right*) and without (*left*) annotation mask.

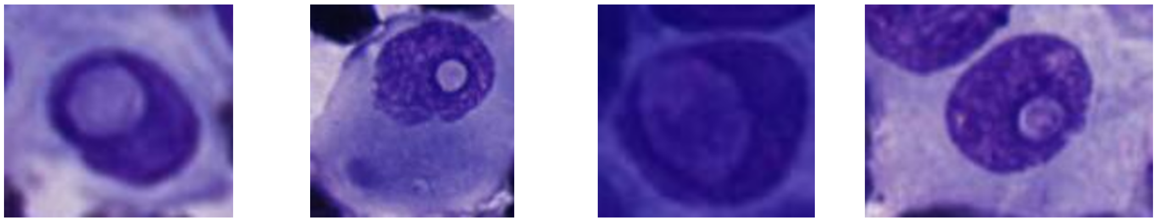


FIGURE 3.12: Example of cells with an intranuclear inclusion from thyroid nodule FNAB smears.

3.3.3 Thyroid nodule malignancy

A thyroid nodule is a small lump that can form within the thyroid gland. Most nodules are benign but up to 5% of them are malignant [227]. Malignancy diagnosis is based on a wide variety of information including patient history (past exposure to radiation, family history of thyroid cancer, *etc.*) and different technical examinations (*e.g.* echography or radiography). A crucial step in the process is the *fine-needle aspiration biopsy* (FNAB) during which cell material is extracted from the nodule. The resulting cytology sample is smeared on a glass slide (this slide preparation process is different from the one described in Section 3.2), stained and examined under a microscope. Cytology analysis is an integral part of the Bethesda diagnosis system [32] which grades a nodule into six categories going from nondiagnostic (TBS I) and benign (TBS II) to malignant (TBS VI). One of the elements that define the TBS category is the presence of cells with intranuclear inclusions (see Figure 3.12) which are highly suggestive of malignancy [7]. Relatively to the size of a slide, these cells are tiny and the problem of finding them comes down to searching a needle in a haystack (see Figure 3.13). However, these cells are not the only indicators of thyroid cancer and considering this is only a part of what must be investigated in a thyroid nodule smear, it would greatly benefit from the use of computational methods. An algorithm could either directly help pathologists in finding the cells or provide a more comprehensive diagnosis system that would make the search of such cells irrelevant.

Although the earliest application of artificial intelligence to nodule malignancy assessment dates back to the 1990s [92], the topic is still quite overlooked and currently available algorithms are mostly benign *vs.* malignant classifiers [96] which are too simplistic to be used in clinical settings. One recent contribution, however, considered the problem of predicting directly one of the categories of the Bethesda diagnosis system [52]. This approach is more realistic and in-line with the pathologists' diagnosis process but still requires further examination to assess its wider applicability.

3.4 Computational pathology and machine learning

Digital pathology has opened the way for the application of machine learning to automate analysis and diagnosis. Albeit promising, the application of machine learning remains challenging for various reasons. In this section, we discuss some of these

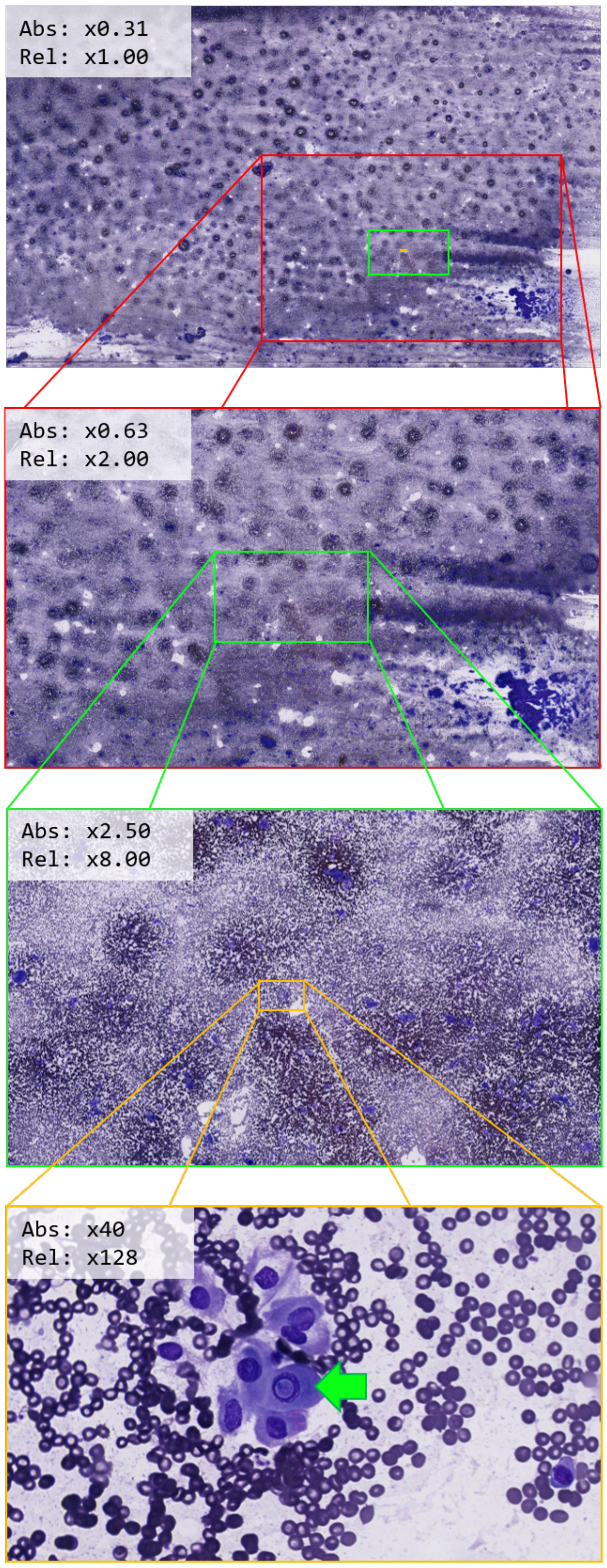


FIGURE 3.13: Views of an intracellular inclusion at different zoom levels. In the top left corner of each image are the absolute and relative magnifications.

challenges then present some techniques that can be used, if not to overcome them, at least to alleviate them.

3.4.1 Data leakage

In Section 2.3.3, we introduced the notion of data leakage that occurs when samples in different splits of a dataset are not independent from each other. Data leakage usually results in poor evaluation of the generalization performance, leading to a poor choice of hyperparameters. Obviously, this should be avoided at all cost especially when the prediction of this model impacts the patient diagnosis and treatment. It is worth noting that data leakage is considered an important obstacle for the application of machine learning in biology and medicine in general, not only in digital pathology [40].

In digital pathology, data leakage can occur in many, sometimes subtle, ways. Therefore, learning pipelines should be built carefully. One important potential source of data leakage is due to the fact that a whole-slide image does not only convey information about the tissue it contains but also about the slide preparation process (see Section 3.2). For instance, staining solutions decay over time. Therefore, the staining intensity can reflect whether the sample has been dipped in a recently-changed or an heavily-used staining bath. When staining is performed manually, the intensity might also reflect the identity of the technician who performed the staining operation. Indeed, some technicians might dip slides for a little longer than others resulting in a stronger intensity. When a dataset is built by different laboratories, the origin of a slide is traceable due to differences in the slide preparation processes between sites.

These slide idiosyncracies are harmless as long as they are not correlated with the learning problem target. Whereas it might seem unlikely to happen, some simple though unfortunate choices can lead to correlation. For instance, supposing a problem of malignancy assessment, if one laboratory provides all the healthy samples and another all the malignant samples, there is significant risk that the learning algorithm would exploit the differences resulting from the preparation processes. This would obviously lead to poor generalization when the model will be applied to slides coming from another laboratory for instance.

The slide preparation process is not the only culprit for data leakage. Another possible source occurs at the patient level. Bussola et al. [31] have empirically studied patient-wise and random dataset splitting strategies. They have shown that overfitting and data leakage indeed occur when splitting samples randomly.

In general, it is difficult to completely prevent data leakage as one does not always have control over the whole WSI generation process. However, good practices surely help reducing the problem to an acceptable minimum. A detailed list of guidelines and good practices to reduce the risk of data leakage can be found in [129]. This includes collecting data as representative as possible of the different variations that could naturally occur during the generation process. For model evaluation and selection, splitting the dataset into subsets should be performed considering the characteristics of the samples that could correlate with the target (*i.e.* patient, laboratory, technician, staining, equipement, time of the day/week, *etc*.).

3.4.2 About annotations

Annotations are important to train machine learning models in a supervised manner. They provide information to the algorithm that can adapt the model to fit more accurately the target data. An annotation typically links two elements: an annotation construct and a semantic information.

An annotation construct concerns the format of the annotation. For instance, in classification, the annotation is a label usually encoded as an integer. For object detection, a common construct is the bounding box framing an object of interest. In image segmentation, the construct is often a hand-drawn polygon covering all pixels belonging to a structure of interest. The semantic information links an annotation to the semantic of the tackled problem and is necessary to differentiate one annotation from another. For instance, in the case of tissue segmentation, the semantic information would be the type of tissue delineated by the hand-drawn polygon (*e.g.* adipose or conjonctive tissue, tumor, metastase).

When it comes to annotating a WSIs dataset, the level of annotation must also be considered. Indeed, annotation can be performed at slide, region or cellular level. The combination of annotation construct, semantic information and level are an important choice in an annotation project and must be chosen carefully to match the target application and fit the allotted budget [215]. Indeed, some constructs are more time consuming and therefore expensive to obtain than others (*e.g.* hand-drawn polygons at the cellular level). The cost of annotation is a challenge for computational pathology and is one of the causes of data scarcity which is discussed in the next section.

3.4.3 Data scarcity

As introduced in Section 2.2.1, *data scarcity* refers to a context where data is lacking which usually hampers the performance of machine learning methods, especially deep learning. Data scarcity is often cited as one of the major challenges in computational pathology [208, 124, 164, 102]. A misconception would be to consider that data simply do not exist in a sufficient quantity. Indeed, hospitals and research institutes have accumulated a significant amount of data in different formats over the years (imaging, text, *etc*.). What makes computational pathology but also the whole field of medical and biological image analysis a data scarce domain is a combination of factors preventing these data to be usable for machine learning in a straightforward way.

Many successes in the application of machine and deep learning to natural images problems were made possible by the availability of numerous large and exhaustively annotated datasets. For instance, the ImageNet classification dataset features

Challenge			# WSI	# ROI	ROI size	Crowd	AI-assist.	Task	# targ.
		tils	82	/	/	yes	yes	REG	$r \in [1, 100]$
[211]	TIGER	rois	195	2032	$< 1.5k \times 1.5k$	no	no	SEG	7
		bulk	93	/	/	no	no	SEG	2
[64]	CoNiC		/	4981	256×256	no	yes	SEG, CLF	6
[10]	MIDOG 2021		50	200	$< 5k \times 5k$	no	yes	DET, CNT	2
[224]	BCNB		1058	/	/	no	no	CLF	$16^{(1)}$
[75]	WSSS4LUAD		87	10k	$< 300 \times 300$	no	no	CLF	2
[5]	BCSS		/	151	$< 7k \times 10k$	no	no	SEG	7
[4]	NuCLS		/	3944	$< 300 \times 300$	yes	yes	SEG, DET	12
[90]	PAIP 2021		150	/	/	no	no	$\text{SEG}^{(2)}$	4

TABLE 3.1: List of challenges published in 2021 with the "histology" modality on the Grand Challenge website. The "Crowd" and "AI-assist." columns relate to the annotation process and how it was performed. The former indicates whether or not non-pathologists were involved in the process (*i.e. yes* for crowdsourcing). The latter indicates whether or not the annotation process involved some kind of AI assistance. The "# targ." indicates the number of target categories/classes of the underlying machine learning problem. REG, CLF, SEG, CNT and DET respectively stand for regression, classification, segmentation, counting and detection. (1) The BCNB challenge proposes 6 different classification tasks each with up to 4 classes (for a total of 16 classes). (2) The expected output is not a classical segmentation mask but rather the boundary of the structure of interest.

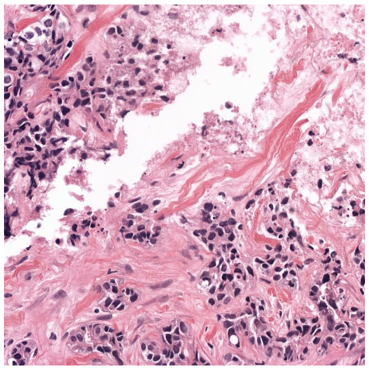
1000 fine-grained classes for 1.2 million images (see Figure 3.14) and has been a key element in deep learning innovations since AlexNet. Another example is *Common Objects in COntext* (COCO) by Microsoft [121], a large-scale dataset for segmentation, detection and captioning (*i.e.* assigning a descriptive sentence to an image). It contains more than 200k images annotated with fine segmentation masks over objects from 91 distinct categories (*i.e.* humans, furniture, animals, *etc.*). It counts more than 800k unique annotated instances of these categories. Initially published by Google in 2016, the most recent iteration of Open Images Dataset [110] contains approximately 9.2 million images each annotated with one or more labels from 19.8k concepts for a total of 30 million image-level labels. JFT-3B is an unpublished dataset from Google introduced in [234] that contains 3 billion images with noisy labels from a hierarchy of 30k items.

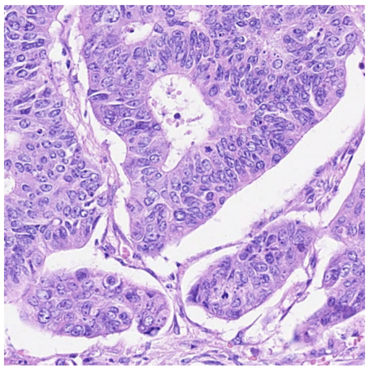
Those were only few examples of the plethora of datasets available in the natural image domain. In computational pathology, the growing interest for ML-based solutions has encouraged researchers and practitioners to increasingly share their data and annotations. Although the data scarcity situation is slowly improving, dataset size, versatility and variety are still subpar compared to the natural image domain.

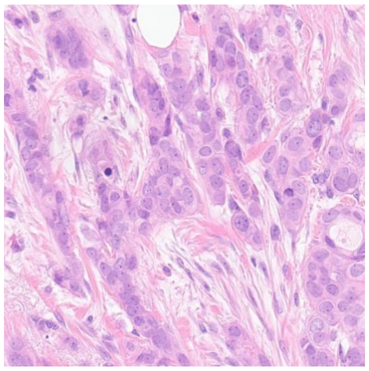
3.4.3.1 A mini-review of Grand Challenge pathology datasets from 2021

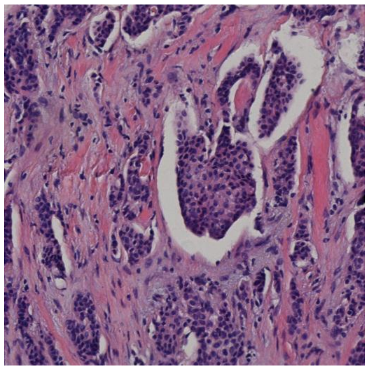
In order to illustrate this point, we performed a search on the Grand Challenge website [68], a popular platform which runs machine learning challenges related to biomedical images, each challenge coming with an open-access dataset. We searched for *histology*

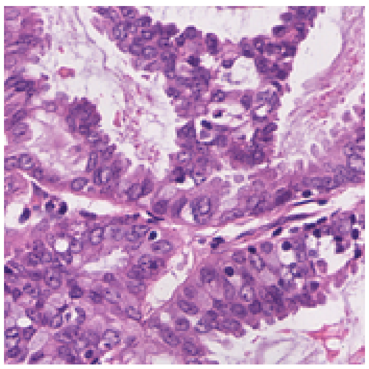
FIGURE 3.14: Samples from ImageNet (source: [93]).

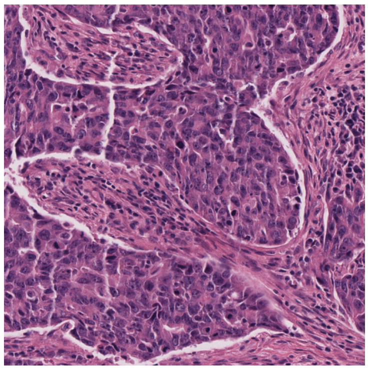

(A) TIGER

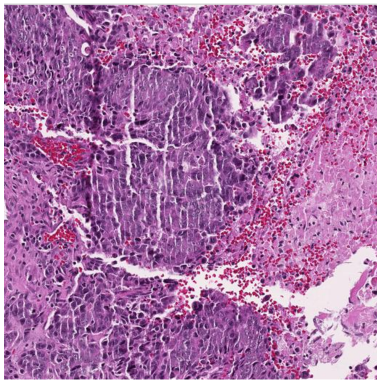

(B) CoNiC (Lizard dataset)


(C) MIDOG 2021


(D) BCNB


(E) WSSS4LUAD


(F) BCSS


(G) NuCLS

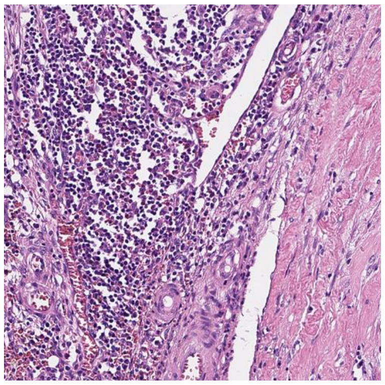

(H) PAIP 2021

FIGURE 3.15: Samples from the Grand Challenge datasets of our mini-review.

challenges published in 2021 and found 8 results (see Table 3.1 for a brief description of relevant aspects and Figure 3.15 for selected samples of these datasets).

A first observation is that, whether the dataset is a set of WSI or *region of interests* (ROI), the number of provided images is several orders of magnitude below the number of images available in natural image datasets. It can be argued that the dimensions of WSI and ROI images in digital pathology datasets is significantly larger that natural images (*e.g.* average image size in ImageNet is 469×387 pixels, the largest of the 151 ROI of BCSS reaches approximately 7k $\times$ 10k pixels), however this must be put in perspective in relation to the prediction task. For instance, hundreds of WSIs represent a very large amount of raw data (*e.g.* few terabytes) but, when the target task is whole-slide classification, this only amounts to a hundred of annotated samples which is significantly fewer compared to ImageNet or others and can be considered a rather small sample size from a ML perspective.

Beyond the size of datasets, it is interesting to note that most prediction tasks in digital pathology only feature few classes (at most 12 in our Grand Challenge sample). It is common to encounter tasks presented as binary (*e.g.* malignant *vs.* benign) even though it is often a simplification of the underlying biological or medical problem. Datasets with a large variety of classes have shown to be effective for learning efficient models on natural images. Therefore, on this matter, computational pathology is still lagging behind significantly.

It is also interesting to consider how these datasets were built. Among them, four datasets use internal data acquired and annotated specifically for the challenge (PAIP 2021, MIDOG 2021, BCNB, BCSS), two of them mix both internal and external data (TIGER, WSSS4LUAD) and the last two use exclusively data from external sources (CoNiC, NuCLS). Regarding the use of external data, it either means that external WSI were imported and annotated for the challenge or that data from other datasets were combined together. For instance, the CoNiC challenge is based on the Lizard dataset [66] which combines data from the *The Cancer Genome Atlas* (TCGA) [219], PanNuke [58], CRAG [67], CoNSeP [65], GlaS [184] and DigestPath [118]. The TCGA is an open platform gathering various data related to cancer genomic including a little more than 30k whole-slide images, making it one of the largest open databases of WSI to date. It is no surprise that, over the years, many datasets have been built using the TCGA as one of their sources. This also includes WSSS4LUAD, TIGER and NuCLS from our Grand Challenge sample. Interestingly, one of the components of Lizard, PanNuke has also been built from The Cancer Genome Atlas (TCGA) and three external datasets of which two were built on top of the same platform (MoNuSeg [109] and CMP17 [214]). Although it raises the question of the risk of data leakage, assembling existing datasets to form a larger one certainly helps fighting data scarcity.

3.4.3.2 Causes

The mini-review performed in the previous section was not aimed at being a thorough evaluation of DP datasets characteristics. It rather serves as a way to highlight different consequences of data scarcity in the field: small dataset size, lack of variety and versatility, *etc*. The scarcity in digital pathology is caused by a combination of factors.

One of the main causes of scarcity is the cost of the annotation process. Pathology is not a simple subject and slide evaluation requires years of training and experience. Whereas classifying pictures into object categories can be done by mostly anyone, creating a ground truth digital pathology dataset requires trained pathologists for whom time is a precious and expensive resource. Therefore, annotation cost in digital pathology is significantly higher compared to the natural image domain. This is aggravated by the fact that disagreement between pathologists is not uncommon and quality ground truth usually requires confronting and aggregating annotations by several experts.

Privacy and ethical concerns also play a role. Indeed, medical data is sensitive and cannot be shared without patient consent, rightfully so, preventing data to be made available for machine learning. Privacy might incur additional costs as it might be necessary to keep track where data records have been used, for instance, in the context of the European General Data Protection Regulation (GDPR). Indeed, in case a patient revokes his data sharing agreement, under the right-to-be-forgotten, models might have to be re-trained [82]. Overall, it can discourage researchers and practitioners to make their data available.

The use of data-hungry deep learning algorithms does not help and is aggravated by the need for a dataset to contain enough samples to account for the variability incurred by the slides content and preparation process.

3.4.3.3 How to work around data scarcity ?

The causes of data scarcity are numerous and it remains a challenge for computational pathology but the situation is evolving positively.

Nowadays, there exist annotation strategies which allow to cut the cost per annotation and therefore produce larger datasets given the same budget. These strategies include crowdsourcing through citizen science [154] or the intervention of medical students (*e.g.* NuCLS dataset). These approaches obviously require either supervision by pathologists or more annotations per image to average out the inaccuracies (or even both) but these are in general less expensive than direct annotation by trained pathologists. It is also possible to indirectly reduce the annotation cost by reducing the time spent per annotation. One way of achieving that is to use the AI to assist experts and accelerate the annotation process (see NuCLS, CoNiC, MIDOG 2021 from our Grand Challenge sample) [34]. A weak but fast algorithm could for instance suggest cell boundaries. The annotator could then correct the boundaries if necessary which is less time-consuming than drawing them from scratch. The use of dedicated and intuitive user interface with efficient drawing tools can also help in that regard.

If annotating new data is not an option, it is also possible to combat data scarcity by the use of existing external data and proper computational methods [210]. There exist several open resources for digital pathology [128] providing access to slides and annotations. We have already introduced TCGA which provides access to a spectacular amount of 30k WSI. The Camelyon dataset introduced in Section 3.3.2 contains

1399 WSI of which 209 were annotated at full resolution with detailed hand-drawn contours of all metastases. This is one of the largest most-precisely annotated datasets in the field to date. The Lizard dataset (subject of the CoNiC challenge) contains almost 500k segmented nuclei all labeled into one of 6 classes. Some promising projects are also ongoing such as Bigpicture [145] of which the goals are to "*create the first European, ethically compliant, and quality-controlled whole slide imaging platform, in which both large-scale data and AI algorithms will exist*". This project aims at having the same impact on computational pathology as ImageNet had on the natural image domain by constructing a database of millions of digital slides.

Sometimes the lack of data is such that the dataset is not representative enough of the different variations that could occur in a realistic context. The impact of this issue can be alleviated during the training process directly with an ad-hoc data augmentation procedure that would automatically generate the kind of variations that can be expected (*e.g.* staining intensity, deformation, *etc.*). Data normalization during pre-processing such as stain normalization can also help in that regard [91, 175, 240].

Regarding the computational methods, there exist machine and deep learning algorithms which can make use of external or sparse data. We explore some of them in the thesis and present related works in the following sections.

3.4.4 Transfer learning

Transfer learning (see Sections 2.2.5 and 2.6.4) is one of the most popular techniques for tackling data scarcity. In this section, we explore how deep transfer learning was applied to biomedical images. Some of the first applications were reported in [16, 43, 63] for pulmonary nodule detection in chest x-rays and CT-scans using Decaf and OverFeat (feature extractors based on ImageNet and AlexNet) as feature extractors. While those works have revealed the potential of deep transfer learning in that field, the performances were not significantly better than those of previous methods. Ravishankar et al. [161] compared the performance of a pre-trained AlexNet-based CaffeNet [88] to well-established computer vision like *histogram of oriented gradient* (HoG) [135] and found the former to outperform the latter.

Later, as new ImageNet architectures emerged, their transfer potential from ImageNet was also evaluated on various biomedical imaging tasks. Antony et al. [6] evaluated the use of fine-tuning and feature extraction of different architectures including VGG16 for quantifying knee osteoarthritis severity on radiography images. Kieffer et al. [98] compared training from scratch to both feature extraction and fine-tuning for classification and retrieval of pathology images using InceptionV3 and VGG16. Shin et al. [182] studied the interest of transfer of different architectures including GoogLeNet and AlexNet for thoracoabdominal lymph node detection and interstitial lung disease classification. Some of these works and others [156, 199] have shown that transfer learning usually yielded better performance than training neural networks from scratch.

As introduced earlier, some works have shown that transfer learning is likely to provide better performance when the source and target tasks are close [228] or if the source task includes the target domain [137]. This implies that pre-training models on pathology data directly is a sound approach although it requires a significant amount of source data to make a proper source task. This was confirmed in several contributions.

For instance, Khan et al. [97] pre-train an InceptionV3 network on a custom dataset generated from Camelyon16 and then transfer the resulting model to a prostate cancer classification task. They show that their pre-trained model outperforms both training from scratch and using an ImageNet pre-trained model. Medela et al. [136] also use transfer learning between two pathology tasks but rather than following a classical supervised pre-training approach, they adopt a self-supervised algorithm by training a siamese network to distinguish between different parts of colorectal tissues. The network is then transferred as a feature extractor on the target task (tumor classification). Shang et al. [180] use several datasets (including some unrelated to their target task such as *Dogs vs. cats*) and compare ImageNet and domain-specific pre-training in order to tackle colonoscopy image classification. They also show that pre-training on domain-specific data yield superior performance compared to using ImageNet. Kraus et al. [106] train a custom deep neural architecture, DeepLoc, for classifying protein subcellular localization in budding yeast. Then, they assess the transferability of their pre-trained DeepLoc by fine-tuning it on different image sets, including unseen classes, and show that the pre-training is indeed beneficial.

3.4.5 Multi-task learning

Multi-task learning (see Section 2.2.6 and 2.6.5) has been applied to medical imaging. Samala et al. [177] jointly train a classifier on three mammography image datasets (digitized screen-film and digital mammograms) and compare it to single-task training and transfer learning. They show that a multi-task trained network generalizes better than a single-task one. Zhang et al. [237] use transfer and multi-task learning to derive image features from Drosophila gene expression. MTL has also been applied more specifically to computational pathology. Pan et al. [148] apply MTL for breast cancer classification by using a classification loss and a verification loss. The role of the latter is to ensure that features produced by the network differ for images of different classes. Arvaniti and Claassen [9] use both weak and strong supervision at once to classify prostate cancer. Shang et al. [180] evaluate multi-task learning which is the best performing approach on their target task. However, they suggest that more experiments would have to be carried out to assess whether their conclusions are generalizable.

3.4.6 Self-training and weakly supervised learning

We have introduced self-training in Sections 2.2.3 and 2.6.6. It is not surprising that self-training has also been applied to medical image tasks to combat data scarcity [200, 153] and to computational pathology in particular. Most works in this domain currently treat with image classification [152, 195, 104, 85, 181] but detection and segmentation have also been explored. Li et al. [119] combine weakly-supervised learning and self-training to predict per-pixel Gleason score across entire WSIs. Li et al. [118] build a signet ring cell detector using self-training. To mitigate the impact of erroneous pseudo-label, their approach features a cooperative training step where two models are trained on the pseudo-labels generated by one another during the previous round. Self-training has also been applied to segmentation for other image modalities. To segment cardiac MR images, Bai et al. [12] propose a self-training approach to train a segmentation architecture. In particular, the teacher and student are the same model and the student is not reset between training rounds. Additionally, they apply a *conditional random field* (CRF) to refine the model predictions on the unlabeled images. Fan et al. [54] focus on lung infection segmentation in the context of the COVID-19 pandemic. Their approach features a self-training protocol where at every training round, they pseudo-label K new unlabeled images which will be added to the learning set for the next round.

Semi-supervised literature also concerns the use of imperfect data which relates more to weakly-supervised learning. This approach has also been explored for biomedical tasks. Wolny et al. [221] learn from sparse instance segmentation masks using their sparse single object loss which combines an instance-based loss and an embedding consistency loss. They also evaluate their method on two microscopy images datasets. Bokhorst et al. [23] segment tissues from colorectal cancer patients into 13 classes representing different types of tissues. Their dataset is composed of both exhaustively- and sparsely-labeled images. During training, they apply a weight map to tune and balance the contribution of individual pixels to the loss. The masks ignore unannotated pixels. On the GlaS dataset [184], Foucart, Debeir, and Decaestecker [56] study how performance of different segmentation methods including semi- and weakly-supervised techniques are impacted by dataset imperfections (*e.g.* deformation, missing annotations, *etc*.). They show that fully supervised approaches are able to cope with noise up to a certain level but quickly degrade after that and that semi-supervised methods are able to partially recover from these degradations.

3.5 Wrapping up

This chapter introduced the domains of digital and computational pathology and gave an overview of different challenges encountered in these fields: high variability in the slide preparation and scanning processes, lack of annotated data, high data volumes, *etc*. These challenges for sure have slowed down the advances in automated image analysis techniques but the situation is evolving quickly as many initiatives are under way to change that. Given the high practical relevance of automated tools in the domain, one can only expect that the interest in computational pathology will continue to grow and envision a future where new image acquisition and analysis techniques will not only revolutionize image analysis itself but also the pathology and diagnosis workflow as a whole. In this thesis, we explore transfer, multi-task and semi-supervised learning in order to tackle computational pathology problems and attempt to work around data scarcity.

Part II

Transfer learning

4 Chapter Transfer learning from ImageNet

Overview

In this chapter, we investigate deep heterogeneous model-based transfer learning as a way to overcome object recognition challenges encountered in the field of digital pathology. Through several experiments, we explore various uses of pre-trained neural network architectures and different combination schemes with random forests for feature selection. Our experiments on eight classification datasets show that densely connected and residual networks consistently yield best performances across strategies. It also appears that network fine-tuning and using inner layers features are the best performing strategies, with the former yielding slightly superior results.

References: this chapter is an adapted version of the following article
Romain Mormont, Pierre Geurts, and Raphaël Marée. "Comparison of deep transfer learning strategies for digital pathology". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 2262–2271

Supplementary materials can be found in Appendix A.

4.1 Introduction

Data scarcity in computational pathology is a recurrent problem which can be tackled in several ways, one of them being transfer learning (see Section 3.4.3). In this chapter, we explore how convolutional neural networks pre-trained on ImageNet [48] (source task) can be transferred to computational pathology target tasks as these architectures have shown interesting transfer properties [49, 228, 179]. At the time of writing the article this chapter is based on (early 2018), the feature extraction and fine-tuning transfer approaches had not been compared thoroughly yet in computational pathology and there was no consensus about whether one was better than the other (see related works in Sections 2.6.4 and 3.4.4). Moreover, most works involving transfer learning in medical imaging used old architectures such as AlexNet [182, 18, 6, 161, 199, 108, 99], GoogLeNet [182, 18] or VGG [98, 18, 6, 231, 79, 108]. To the best of our

Dataset	Domain	Classes	Images			
			Train	Validation	Test	Total
CellInclusion (C)	Cyto	2	1644	173	1821	3638
ProliferativePattern (P)	Cyto	2	1179	167	511	1857
Glomeruli (G)	Histo	2	12157	2448	14608	29213
Necrosis (N)	Histo	2	695	96	91	882
Breast (B)	Histo	2	14055	4206	4771	23032
MouseLba (M)	Cyto	8	1722	716	1846	4284
Lung (L)	Histo	10	4881	562	888	6331
HumanLba (H)	Cyto	9	4051	346	1023	5420

TABLE 4.1: Sizes and splits of the datasets. The "Glomeruli" dataset was first used in [130].

knowledge, only one article used residual networks for transfer learning at that time [231] and none were using dense networks.

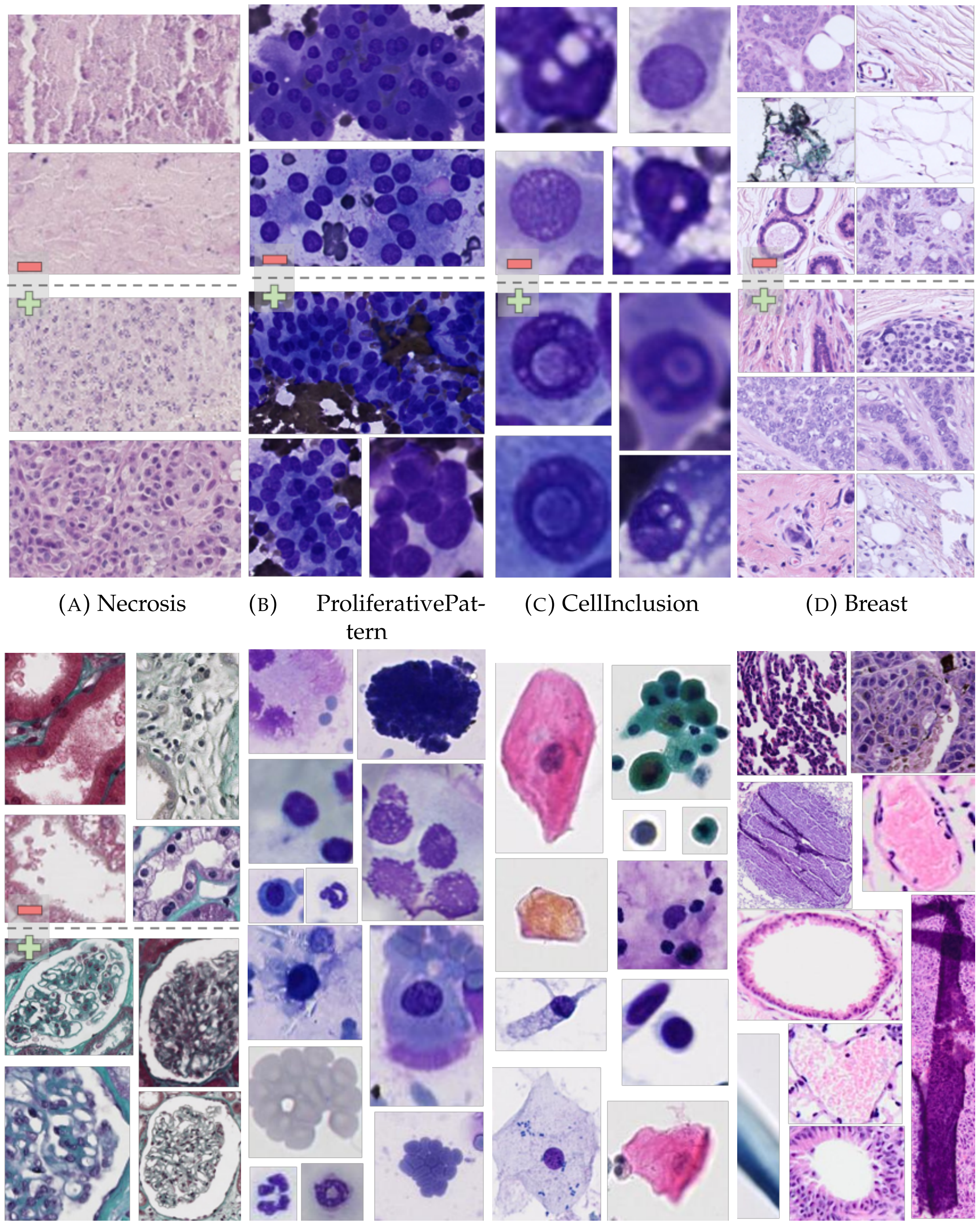
Therefore, we study thoroughly and compare several strategies that involve feature extraction and fine-tuning. We carry out several experiments over eight object classification histology and cytology datasets. Different combinations of state-of-the-art networks and feature selection techniques using random forests are proposed in order to answer questions of high pratical relevance: which network provides best-performing features? How should those features be extracted and then exploited to get the best performance? Is fine-tuning better than using features from off-the-shelf networks? More generally, our empirical study also contributes to confirm the interest of deep transfer learning for tackling the recurrent data scarcity problem in computational pathology.

This chapter is organized as follows. We first introduce the eight datasets we have used in Section 4.2. We present how we implement feature extraction and fine-tuning in Section 4.3. Our experiments and results are presented and discussed in Section 4.4. We finally conclude in Section 4.5.

4.2 Datasets

Our experimental study uses datasets collected over the years by biomedical researchers and pathologists using the Cytomine [132] web application. Using this platform, eight image classification datasets were collected which are summarized in Table 4.1. These contain tissues and cells from human or animal organs (thyroid, kidney, breast, lung, *etc.*).

For all datasets except Breast, each sample image is the crop of an annotated object extracted from a whole-slide image, a crop being a rectangle image containing exactly the object. The Breast dataset is composed of patches for which the label encodes the type of tissue in which the central pixel is located. Selected image samples for each dataset are shown in Figure 4.1.



(A) Necrosis (B) ProliferativePattern (C) CellInclusion (D) Breast

(E) Glomeruli (F) MouseLba (G) HumanLba (H) Lung

FIGURE 4.1: Overview of our eight classification datasets (the display size does not reflect actual image size). For binary classification datasets, negative and positive samples were respectively placed at the top and bottom of the figures.

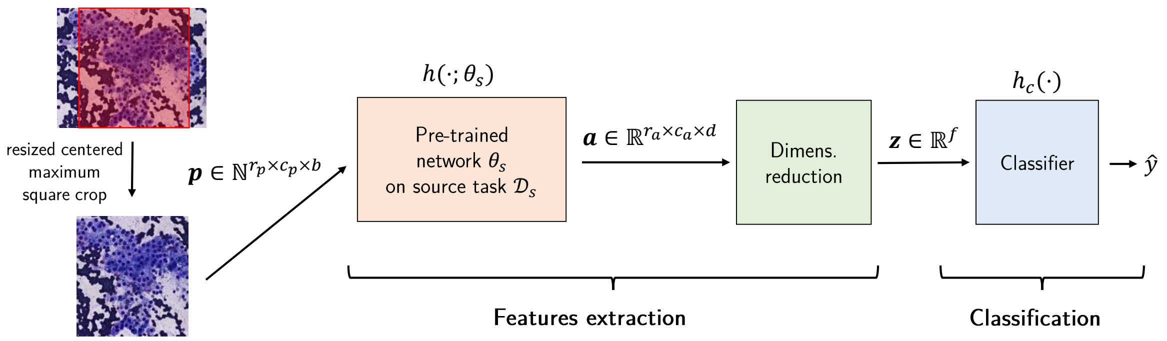

resized centered maximum square crop
$p \in \mathbb{N}^{r_p \times c_p \times b}$
$h(\cdot;\theta_s)$
Pre-trained network θ_s on source task $\mathcal{D}_s$
$a \in \mathbb{R}^{r_a \times c_a \times d}$
Dimens. reduction
$z \in \mathbb{R}^f$
$h_c(\cdot)$
Classifier
$\hat{y}$
Features extraction
Classification

FIGURE 4.2: Feature extraction from pre-trained convolutional neural networks

4.3 Methods

In this section, we introduce the architectures we have used in Section 4.3.1, we describe how we have implemented feature extraction in Section 4.3.2 and fine-tuning in Section 4.3.3. We also provide details about the training procedure of the final classifier in Section 4.3.4 and the inference procedure in Section 4.3.5. As discussed in Section 2.6.4, the interest of feature extraction is that it is not as computationally demanding as fine-tuning but, because the transferred model has not been retrained on the target, the features might not be specific enough. Fine-tuning is much more computationally demanding as it implies retraining a usually large model (*i.e.* tens of millions of parameters or more) but make the features more specific and therefore more relevant to the target which usually has a positive effect on performance.

4.3.1 Deep networks

We follow the feature extraction and classification process presented in Figure 4.2, which starts from a deep convolutional neural network θ_s pre-trained on a source task $\mathcal{D}_s$. In particular, we use ImageNet as the source dataset $\mathcal{D}_s$ and, for each network, the pre-trained weights were retrieved from Keras [41]. For the network θ_s, we evaluate several architectures that have been state-of-the-art on the ImageNet classification dataset [48] or that present interesting trade-off between computational requirements and performances: VGG16, VGG19 [183], InceptionV3 [198], ResNet50 [77], InceptionResNetV2 [196], DenseNet201 [81], and MobileNet [80]. In the sequel, those networks will be respectively referred to as VGG16, VGG19, IncV3, ResNet, IncResV2, DenseNet, and Mobile. These networks can be used as feature extractors or fine-tuned, as explained in the following sections.

4.3.2 Feature extraction

Images are first resized to match the input dimension of the network. Respectively denoting by $r_I \times c_I \times b$ the height, width, and number of channels of the input image I, we extract a square patch $\mathbf{p}$ of (maximum) height and width $min\ (r_I, c_I)$ in the center of the image, which is then resized to the network input size $r_p \times c_p \times b$. This extraction process is parameter-free and preserves the aspect ratio of the image, since all pre-trained networks takes square images as inputs (*i.e.* $r_p = c_p$).

The resized patch is then forwarded through θ_s (loaded with pre-trained weights). Given this input, the output $\mathbf{a}$ of an arbitrary layer l (*i.e.* a set of feature maps) of dimensions $r_a \times c_a \times d$ is extracted where d is the number of feature maps and r_a and c_a are respectively their height and width. Because this tensor can be high-dimensional, one usually applies a dimensionality reduction procedure (*e.g.* global average pooling, principal component analysis, *etc.*) to reduce it to f features, yielding a feature vector $\mathbf{z} \in \mathbb{R}^f$. Here, we limit our analysis to global average pooling (*i.e.* feature maps averaging), which, unlike principal component analysis for example, has the advantage of being parameter-free.

4.3.3 Fine-tuned feature extraction

For our experiments on network fine-tuning, we replace the final fully connected (FC) layer by a fully-connected layer with as many neurons as there are classes in the current dataset. This newly-appended layer being randomly initialized, it is not correlated at all with the transferred network θ_s yet. In general, it is a good idea to freeze θ_s and to train this new layer for a few epochs to correlate them. In our case, we perform this "warm-up" phase for 5 epochs with a learning rate of 10^{-2}. Then, we train the whole network for 45 epochs with a learning rate of 10^{-5}. We use the Adam optimizer [100] with parameters β_1 and β_2 respectively set to their recommended values 0.99 and 0.999 and no weight decay. We use the categorical cross-entropy as a loss function. Fine-tuning is performed using the training set exclusively. Optionally, we use data augmentation to virtually increase the size of the training set. First, a maximum square crop is taken at a random position in the input image. Then, random flips and rotations are applied to the resized patches before they are forwarded through the network. The model is evaluated on the validation set and then saved at the end of each epoch. When the fine-tuning is over, we select among the saved models the one that performed the best on the validation set.

4.3.4 Final classifier learning

When the features have been extracted for all images of a dataset (either using off-the-shelf or fine-tuned networks), they can be used for training the classifier h_c. We use either linear SVM [24] (with a one-vs-rest scheme for multi-class problems), extremely randomized trees (ET) [61], or a fully connected single layer perceptron, all as implemented in `scikit-learn` [151]. SVM is the most popular classification method when it comes to classifying extracted features. ET are incorporated mainly for their ability to compute feature importance scores. FC is a natural choice to mimic how the pre-trained network exploits the features. Hyperparameters of all methods were tuned by slide-wise (or patient-wise) n-fold cross-validation on the merged training and validation sets. Namely, they are the penalty C for SVM, the maximum number of features for ET, and the learning rate and number of iterations for the single layer perceptron. Selected values for tuning are given in Appendix Section A.1.

4.3.5 Prediction

For prediction, the process is similar: patches are extracted from target images, forwarded through θ_s, reduced by global average pooling and classified with the learned classifier h_c. As for the fine-tuning, we make additional experiments using directly the fine-tuned network for classifying the images (see Section 4.3.3 for parameters).

4.4 Experiments

In this section, we propose and thoroughly compare different strategies for extracting and using features from the deep networks introduced previously. We follow a rigourous evaluation protocol described in Section 4.4.1. Strategies are presented and evaluated one after the other in Sections 4.4.2 to 4.4.7, then an overall comparison of strategies is discussed in Section 4.4.8.

4.4.1 Performance metrics and baseline

For performance evaluation, each dataset (presented in Section 4.2) is randomly splitted into training, validation and test sets. Following the guidelines in [129], image patches from the same slide (or patient when this information was available) are all put in the same set to avoid any data leakage (see Section 3.4.1).

To evaluate the different transfer strategies, we use two different metrics: the area under the receiving operating curve (ROC AUC) for binary classification problems and the classification accuracy for the multi-class problems. For the sake of readibility, a summary of the scores for all experiments and datasets is given in Table 4.3 whereas the detailed scores are only given in Appendix Section A.5. In all figures, we plot instead for each method its rank among all methods compared in the same graph averaged over all eight datasets. To associate high rank with best results, we compute the rank when methods are sorted in reverse order of performance (AUC or accuracy). For example, since ten methods are compared in Figure 4.3, the maximum average rank is 22, corresponding to a method being the best one on all eight datasets, and the minimum average rank is 1, corresponding to a method always worse than all others.

As a baseline for comparison, we use the ET-FL variant of the tree-based random subwindows classification algorithm presented in Section 2.5.4 because it is fast, generic and requires only a slight tuning of the hyperparameters which are either set to their default value or tuned by cross-validation (see in Appendix Section A.1 for default values and ranges).

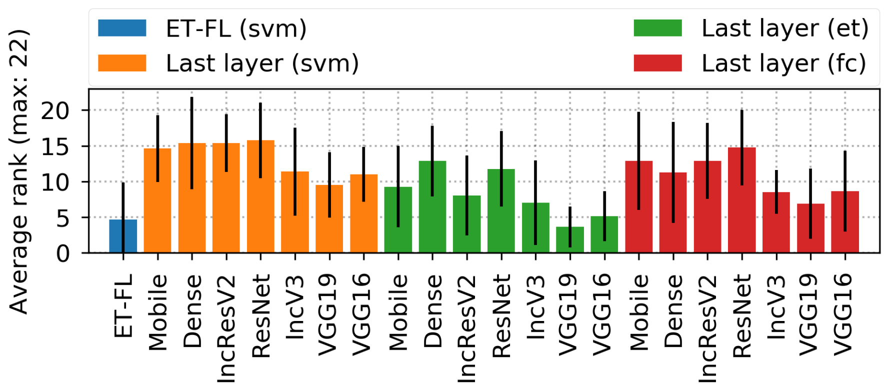


FIGURE 4.3: Average ranks of the methods for the "*Last layer features*" experiment. Colors encode the choice of classifier for h_c (orange for SVM, green for ET and red for FC).

$\mathcal{N}$	**Last layer**	**Merged layers**	
	# feat.	# feat.	# cut
Mobile	1024	/	/
DenseNet	1920	7744	9
IncResV2	1536	17088	12
ResNet	2048	15168	17
IncV3	2048	/	/
VGG19	512	/	/
VGG16	512	/	/
Total	9600	/	/

TABLE 4.2: Number of features extracted for the "*Last layer*" experiment. Total number of features for the "*Merging features across networks*". Number of features and cut points for the "*Mering layers features*" experiment.

4.4.2 Last layer features

Our first strategy follows the common approach where features are extracted at the last layer of a pre-trained network. In our case, we take the features from the last feature maps before the first fully connected layer (the numbers of extracted features per network are given in Table 4.2). For each dataset and network, we then tune and train the three types of classifiers h_c (ET, SVM, and FC) with the extracted features, on the union of the training and validation sets, and then we evaluate them on the test set. The resulting average ranks for all classifiers and all datasets are given in Figure 4.3.

We observe that SVM and single layer perceptron are more efficient at classifying from deep features than extremely randomized trees, with a slight advantage to SVM. Mobile, DenseNet, IncResV2 and ResNet yield best performances when combined with SVM or single layer perceptron, while for extremely randomized trees, only DenseNet and ResNet are leading the way. Last layer features from VGG16, VGG19 and IncV3 allow most of the time to beat the baseline but they are clearly not

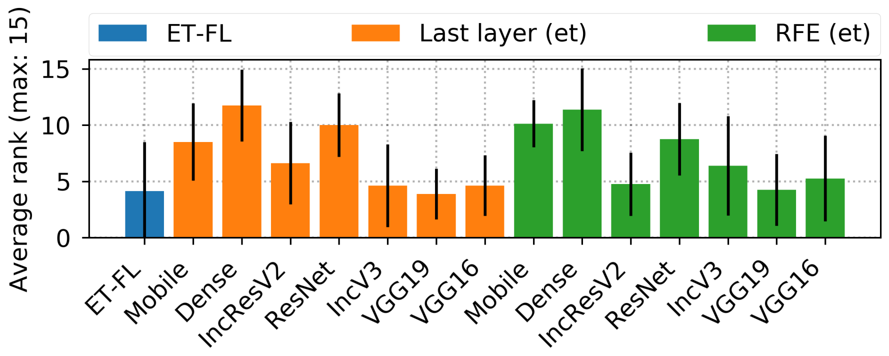


FIGURE 4.4: Average ranks for last layers' features classified with ET before (orange) and after (green) selection with recursive feature elimination.

competitive with features from the other networks whatever the classifier. Overall, the best performance is obtained by combining ResNet features with SVM.

4.4.3 Last layer feature subset selection

Our second strategy aims at checking whether all features extracted from the last layer of the network are important for the final classification or if a small subset of them would be sufficient to obtain optimal performance. To answer this question, we use cross-validated *recursive feature elimination* (RFE) [72] using importance scores derived from extremely randomized trees to rank the features (where the importance of a given feature is computed by adding up the weighted impurity decreases for all tree nodes where the feature is used, averaged over all trees). This method outputs a sequence $\{(S_t, s_t)|t = 1, \ldots, T\}$ (built in reverse), where S_t are nested feature subsets of increasing sizes (with $|S_1| = 1$ and $|S_T| = k$) and s_t is the cross-validation performance of an ET model trained on S_t. From this sequence, we compute for each dataset:

$$k_{min} = \min_{\{t=1,\ldots,T: s_t \geq \max_{t'}(s_{t'} - l_a)\}} |S_t|,$$

where l_a is a small performance tolerance (set to 0.005 in our experiment). k_{min} is thus the minimum number of features needed to reach a performance not smaller than the optimal one by more than l_a.

On average across datasets and networks, this method selected 7.5 % of the features (detailed numbers are given in Appendix Tables A.6 and A.7). The models re-trained using the selected features yielded comparable performance when using ET as classifier (see Figure 4.4). Feature selection even improved ET performance on the IncV3, VGG19, and VGG16 networks. Using SVM on the selected features however leads to a performance drop compared to SVM with all features (see Appendix Table A.10). We believe that this difference is due to the fact that the selection is optimized for ET and it is thus likely to remove features that are useful for linear SVM and not for ET, which are non-linear approximators.

	Datasets							
Strategy	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
Baseline (ET-FL)	0.9250	0.8268	0.9551	0.9805	0.9345	0.7568	0.8547	0.6960
Last layer	0.9822	0.8893	0.9938	0.9982	0.9603	0.7996	0.9133	0.7820
Feat. select.	0.9676	0.8861	0.9843	0.9994	0.9597	0.7438	0.8941	0.7703
Merg. networks	0.9897	0.8984	0.9948	0.9864	0.9549	0.8169	0.9155	0.7928
Merg. layers	0.9808	0.8906	0.9944	0.9964	0.9639	0.7941	0.9268	0.7977
Inner ResNet	0.9748	0.8959	0.9949	0.9964	0.9664	0.8131	0.9291	0.8113
Inner DenseNet	0.9862	0.8984	0.9962	0.9917	0.9699	0.8012	0.9268	0.7967
Inner IncResV2	0.9873	0.8948	0.9962	0.9982	0.9720	0.8137	0.9234	0.7713
Fine-tuning	0.9926	0.8797	0.9977	0.9970	0.9873	0.8727	0.9405	0.8641
Metric	Roc AUC					Accuracy (multi-class)		

TABLE 4.3: Best score for each strategy and each dataset. The best and second best scores are respectively highlighted in green and orange.

These experiments show that, among the available features from the last layer, most of them are uninformative or redundant and therefore only few of them are actually needed for the prediction when using ET as classifier. This conclusion can also be drawn by observing the recursive feature elimination cross-validation curves (see Appendix Figures A.1 and A.2). For all datasets and networks, the accuracy converges very abruptly to a plateau when the number of selected features increases, which indicates that removing features from the learning set does not impact negatively the predictive power of the models.

It is interesting to note that the selected features are not the same across datasets. For instance with DenseNet, for a feature to appear in the subset of best features (determined with the importances obtained during the *"Last layer features"* experiment) for all datasets, we need to consider a subset of size 1477 (*i.e.* 77% of the features). We observe similar results for the other networks (see Appendix Table A.2). On the other hand, there can be a significant overlap between features selected by RFE for specific pairs of datasets (see Appendix Tables A.8 and A.9). These results suggest that the best features are task-dependent and that there would be no interest in restricting a priori the subset of transferred features, even when focusing on the domain of computational pathology.

These conclusions hold when using ET as a classifier. The fact that we observe a drop of performance when using the best features subset with SVM suggests that, although there might exist a different optimal subset for other classifiers, we cannot prove it with our experiments. To achieve that, additional experiments would need to be carried out by applying RFE (or a similar feature selection technique) to those classifiers.

4.4.4 Merging features across networks

The third strategy consists in merging features from the last layer of all the studied networks. Aggregating all features results in a feature vector of size 9600. We observe

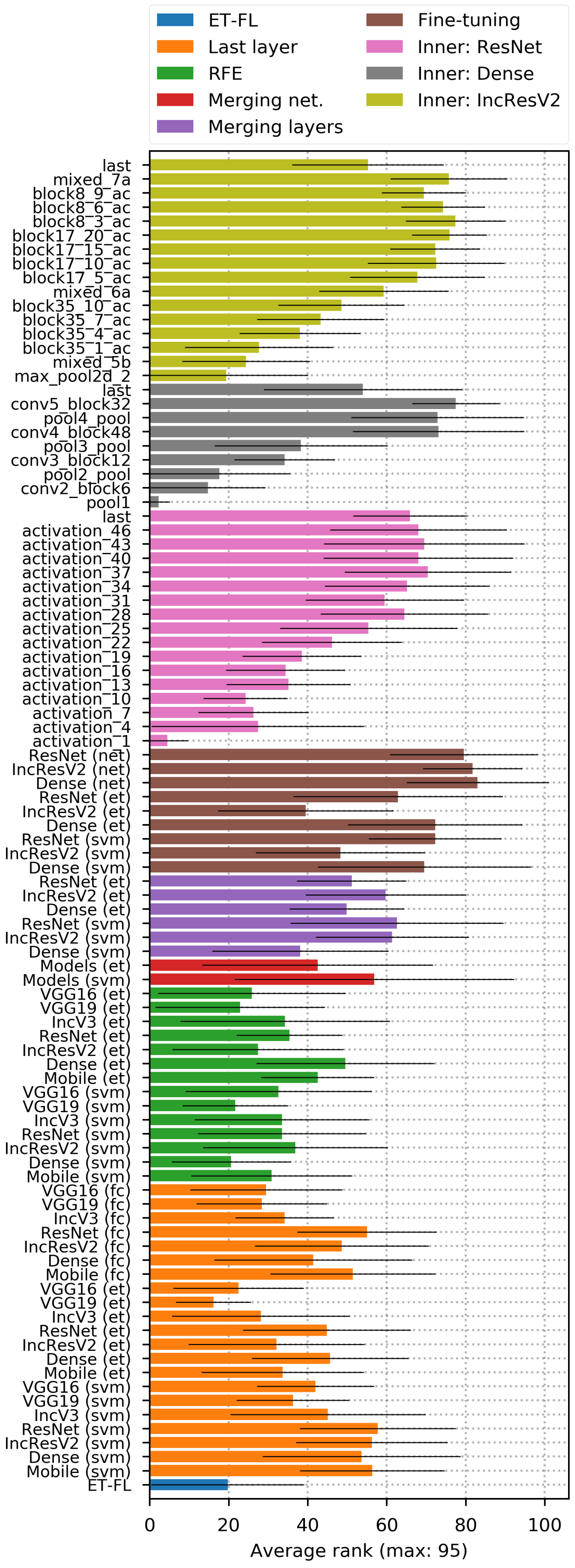


FIGURE 4.5: Average ranks for all the evaluated methods.

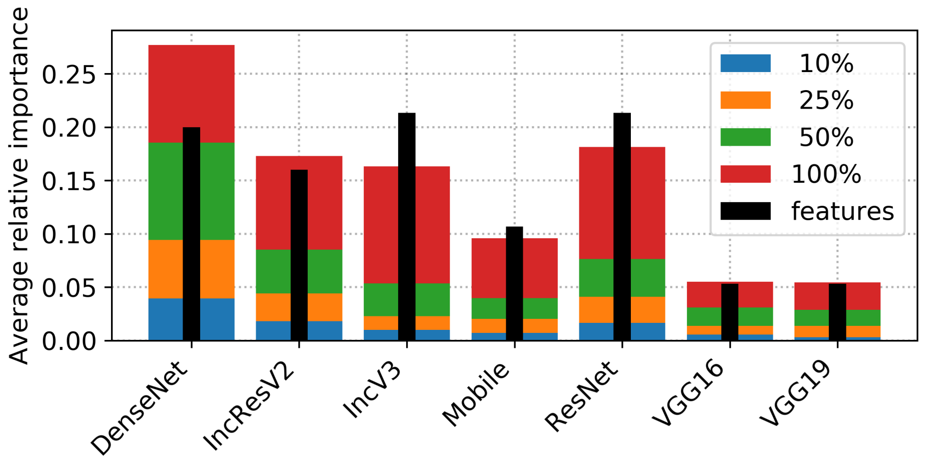


FIGURE 4.6: Average relative importances (across datasets) brought by each studied network when all their last layer features are aggregated. The black bars quantify the proportion of features of each network. The colors indicate the information brought by features of decreasing importances: blue and red features are respectively the most informative and least informative ones. Blue, orange, green and red bars regroup importances of features that respectively and cumulatively bring 10%, 25%, 50% and 100% of the information for predicting the outcome. For example, the DenseNet bar reaches approximately 26%, meaning that DenseNet features brings about 26% of the total information for predicting the outcome. Moreover, the orange and blue segments for DenseNet together reach approximately 10%, meaning that, when selecting the subset of most informative features bringing 25% of the total information (orange + blue), those of DenseNet bring 10% of information (or 40% of the information of the subset).

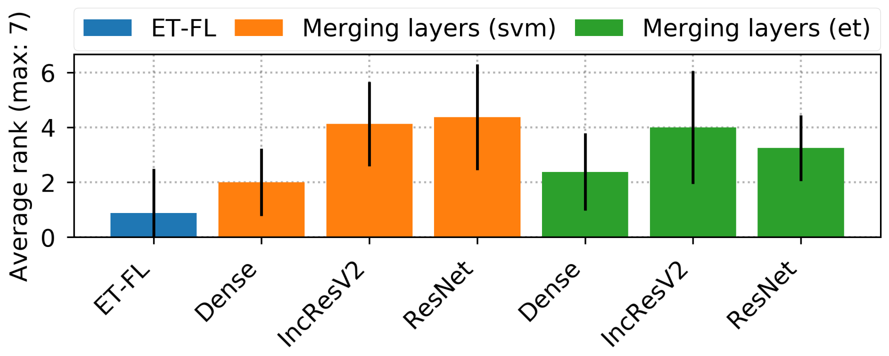


FIGURE 4.7: Average ranks for models learned on merged layers of networks compared to the baseline.

in Table 4.3 and Figure 4.5 that despite the fact that features from all networks are combined, this strategy gives performance results similar to but not better than using the best single network, both with ET and SVM.

Using forest importance ranking procedure described previously, we further analyze the information brought by the last layer of each network (feature importances averaged across datasets are given in Figure 4.6). We observe that the features of DenseNet bring about 26% of information on average while they only account for 20% of all the features. Moreover, the proportion of information brought by the most informative features of DenseNet is higher than for any other network. Following DenseNet, the next most informative networks are IncResV2, ResNet, IncV3 and finally the Mobile, VGG16 and VGG19 networks. Surprisingly, the importances brought by the VGG networks relatively to the number of features is non-negligible and higher than the one of ResNet and IncV3. This may indicate that features of those networks are redundant with features of DenseNet and IncResV2 while features of VGG16 and VGG19 are not.

4.4.5 Merging features across layers

This strategy aims at merging features across layers (at several depths) for a given network. Given the results in Section 4.4.2, we limit our analysis to the IncResV2, ResNet and DenseNet networks. Those three networks have complex structures and many layers which yield plenty of possible cut points for feature extraction. To reduce the number of possibilities, we limit the extraction to bottlenecks of the networks, a bottleneck being a point were several paths are merged into a single one. For ResNet, we have selected the ReLU activations of the merging layer after each residual block. For IncResV2, considering all bottlenecks (following an inception-resnet or a reduction block) yields approximately fifty possible cut points, that is more than 100k features. We have decided to subsample those cut points to obtain a number of features closer to those of other networks while covering the network as uniformly as possible along its depth. As far as DenseNet is concerned, we extract features only at the end of dense blocks and after pooling blocks. Indeed, extracting features inside dense blocks would have resulted in duplicated features because of the dense connections.

In Table 4.2 are given the information about the generated features vectors for the "*Last layer*", "*Merging features across networks*" and "*Merging layers features*" experiments. Information about the dimensions of the extracted features from inside the networks (before global average pooling) for ResNet, IncResV2 and DenseNet are respectively given in Appendix Tables A.3, A.4 and A.5. The layer names given in those tables are the ones given by the Keras package [41].

The average ranks for each layer of all studied networks are given in Figure 4.7. One first observation is that there is no significant difference between SVM and ET in terms of performance, unlike when we use the last layer only. Surprisingly, DenseNet is not performing well with respect to the other network while it was competitive with using the last layer only. Merging the layers actually leads to a drop of performance with respect to using only the last layer for DenseNet, while it leads to a small improvement for the other two networks (see Section 4.4.8).

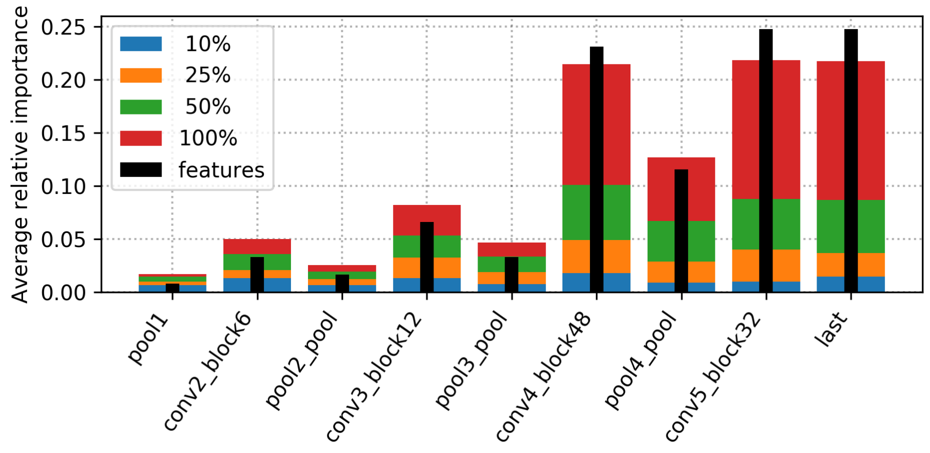
As in the previous section, we use feature importances to identify most informative features. Detailed importances plots for each network are given in Figure 4.8. These plots clearly show that the most informative features are spread over all layers. We also observe that the relative importance of features in the early layers is higher than the ones in deeper layers. One possible reason is that last layers actually have more features than earlier ones and as shown in Section 4.4.3, most of those features are either irrelevant or redundant. Therefore, the extremely randomized trees actually discard most of them them during training.

Merging features from several layers results in large feature vectors for describing the images. Those large vectors make this method less attractive as it results in longer classifier training time. This is especially true for extremely randomized trees. Unlike in the previous strategy however, feature extraction does not increase computation requirements as only one forward pass through the network is needed to extract all the features.

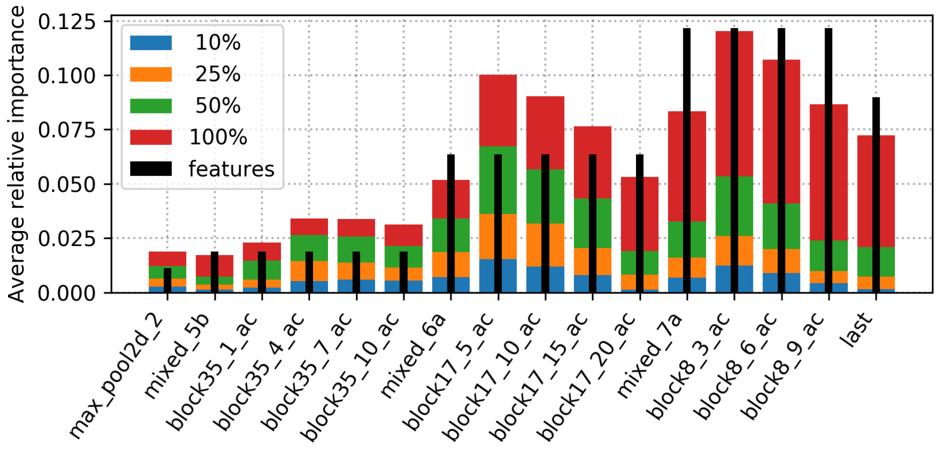
4.4.6 Inner layers features

For this strategy, we assess features extracted from each layer separately. The motivation is to determine if there is a layer that, taken alone, yields better performance than the others, and in particular, the last one. We use the same cut points as for the previous experiment (see Section 4.4.5), we learn as many SVM classifiers as there are layers, each using the features of a single layer.

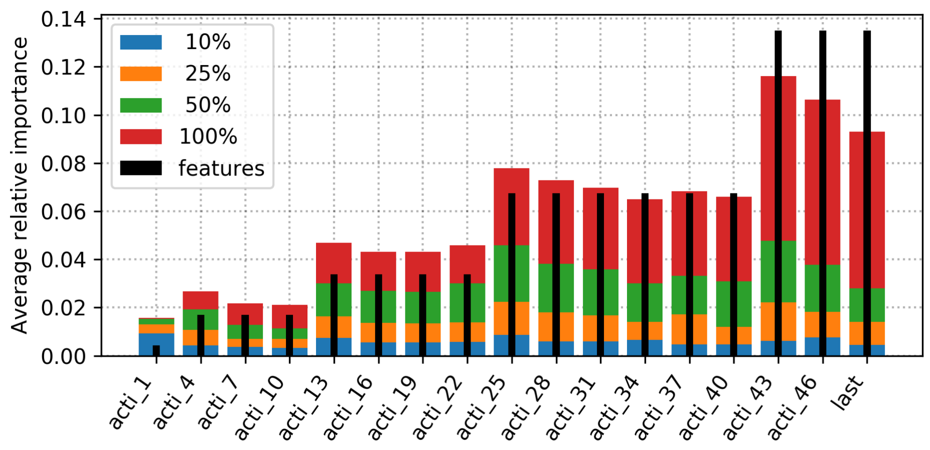
Average ranks for each inner layer and each network are given in Figure 4.9. In all cases, the last layer features are always outperformed by features taken from an inner layer of the network. The optimal layer is however always located rather at the end of the network, while the first layers are clearly never competitive. Unfortunately, we have not found that a specific layer was better for all datasets, so in practice the choice of the layer should be determined by internal cross-validation as we did. Interestingly, the baseline either outperforms the early layers of the networks or yield comparable



(A) DenseNet



(B) IncResV2



(C) ResNet

FIGURE 4.8: Average relative importances (across datasets) brought by each extracted layer of a network. See Figure 4.6 for explanation.

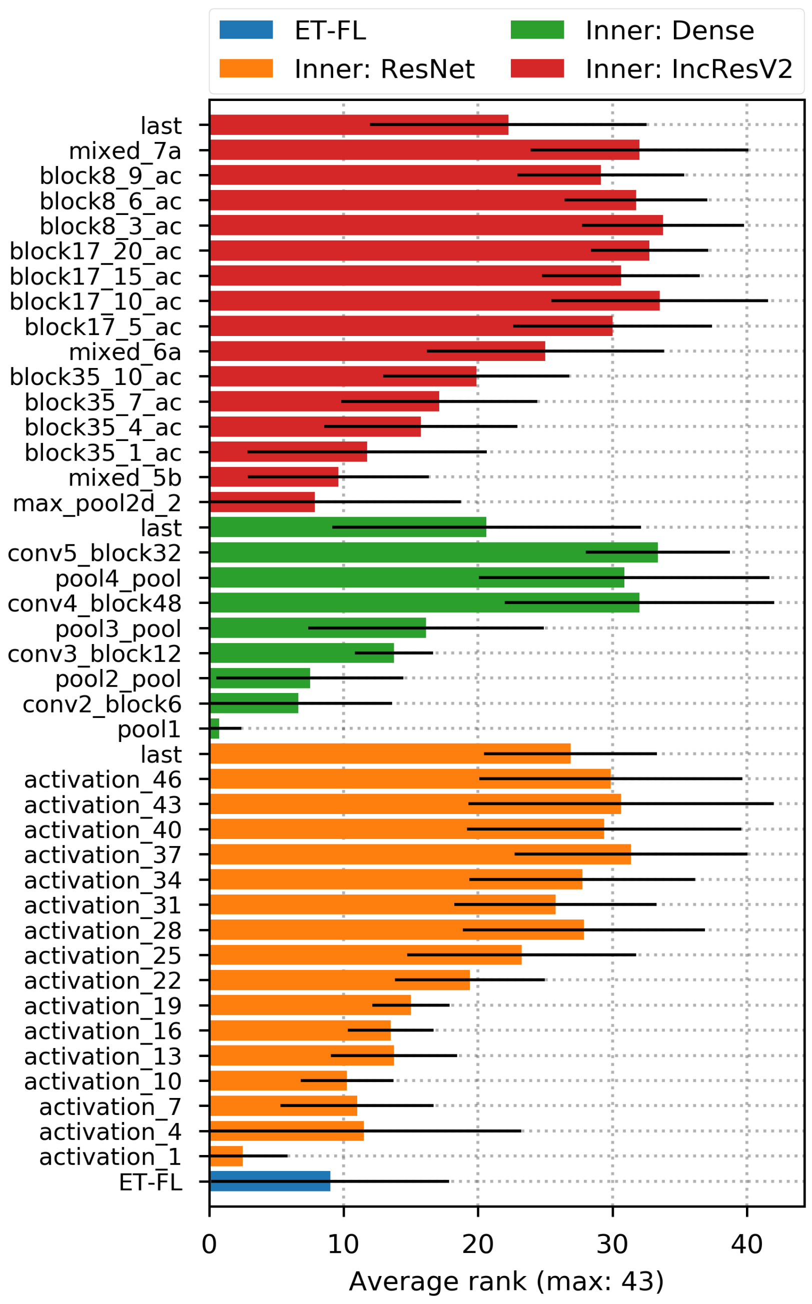


FIGURE 4.9: Average ranks for the baseline and the models trained using features from layers inside the networks. For each network, the layers are sorted by decreasing depth (from top to bottom).

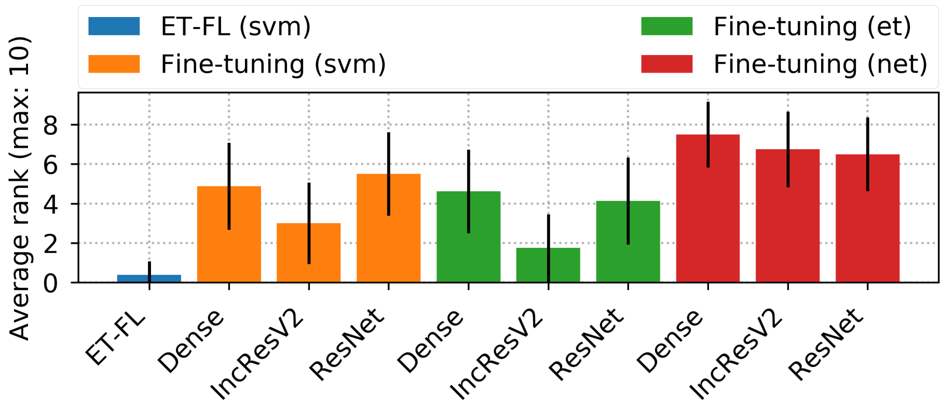


FIGURE 4.10: Average ranks for fine-tuned networks compared to the baseline. Evaluation was done either by using SVM (orange) or extremely randomized trees (green) on the fine-tuned features or by predicting the outcome using the fine-tuned fully connected layer directly (red).

results which tends to indicate that the features provided by ET-FL are somewhat generic, *i.e.* they could be assimilated to features that you would for instance find in the early layers of a classification CNN (see Section 2.6.4).

4.4.7 Fine-tuned features

All previous experiments explored strategies using features extracted from off-the-shelf networks. In this last strategy, we investigate fine-tuning as described in Section 4.3.3. We focus on the same three networks as in the previous sections (ResNet, IncResV2 and DenseNet).

The average ranks for the different fine-tuning methods are given in Figure 4.10. With the three networks, best performances are obtained by making predictions directly from the fine-tuned fully connected layer. SVM and ET trained on the features extracted from the fine-tuned networks are clearly inferior, in particular with IncResV2. Note that, for the other two, last layer features extracted from the fine-tuned network are nevertheless better than last layer features from the original network, when used as inputs to SVM (see Figure 4.5). Fine-tuning is thus globally improving the quality of the features for these two networks. Overall, the best performance is obtained with fine-tuned DenseNet.

4.4.8 Discussion

To allow comparison of all strategies, the best scores per strategy and dataset are summarized in Table 4.3 and the average ranks of all methods evaluated in the previous experiments are given in Figure 4.5.

Concerning the networks, ResNet and DenseNet often yield the best performing models whatever the way they are exploited. They are followed by the IncResV2, Mobile, and IncV3 networks. Performances obtained with the VGG networks are below those of the others.

Regarding the methods, fine-tuning (and predicting with the network) usually outperforms all other methods whatever the network. Especially, this strategy yields significant improvements for the multi-class datasets. For binary datasets, the improvement is often not as impressive, but on three of these datasets, the performances of all methods are already very high (greater than 0.9).

Moreover, for each dataset, there is at least one inner layer that yields the best or second best scores and this best layer is never the last one. This is confirmed by the rank plot (see Figure 4.9) that shows that the ranks of the models learned on last layer features are below those using inner layer features. This might be explained by the fact that last layer features are too specific to the source task (*i.e.* natural images).

Merging features across networks and layers yield results similar to using last layer features but they are outperformed by the best inner layers and also by fine-tuning. The inferior performances of these methods could be attributed to the fact that important features are lost among many redundant and/or uninformative ones and they are thus suffering from overfitting. One way to improve these methods could be to perform feature selection. However, given that these methods are already more computationally demanding, they are definitely less interesting than fine-tuning and selecting the best inner layer. In particular, merging features across networks requires a forward-pass through all the selected networks. Although only one pass is needed when merging the layers, it still yields a large feature vector which makes further training and tuning slower, especially for ET.

Throughout the experiments, we have also gained more insights about the extracted features. By performing feature selection, we have discovered that very few features are actually useful for learning efficient ET classifiers on our datasets and the best features are task-dependent. This conclusion might extend to other classifiers as well but it would require additional experiments to prove it.

4.5 Conclusion

We have empirically investigated various deep transfer learning strategies for recognition in computational pathology. We have observed that residual and densely connected networks often yielded best performances across the various experiments and datasets. We have also observed that fine-tuning outperformed features from the last layer of off-the-shelf networks. It also appeared that using one network's inner layer features yielded performances slightly superior to using those of the last layer and inferior to fine-tuning but with the advantage of not having to re-train the network.

Retrospectively, we believe it would have been interesting to include models trained from scratch in our comparison. They were excluded to reduce the cost of the experiments and because several studies already showed that pretraining on ImageNet was beneficial [182, 199]. Note also that such comparison will be carried out in the next chapter.

Chapter

5 Multi-task pre-training from pathology data

Overview

In this chapter, we investigate multi-task learning as a way of pre-training models for classification tasks in digital pathology. It is motivated by the fact that many small and medium-size datasets have been released by the community over the years whereas there is no large scale dataset similar to ImageNet in the domain. We first assemble and transform many digital pathology datasets into a pool of 22 classification tasks and almost 900k images. Then, we propose a simple architecture and training scheme for creating a transferable model and a robust evaluation and selection protocol in order to evaluate our method. Depending on the target task, we show that our models used as feature extractors either improve significantly over ImageNet pre-trained models or provide comparable performance. Fine-tuning improves performance over feature extraction and is able to recover the lack of specificity of ImageNet features, as both pre-training sources yield comparable performance.

References: this chapter is an adapted version of the following article
Romain Mormont, Pierre Geurts, and Raphaël Marée. “Multi-task pre-training of deep neural networks for digital pathology”. In: *IEEE journal of biomedical and health informatics* 25.2 (2020), pp. 412–421

Supplementary materials for this chapter can be found in Appendix B.

5.1 Introduction

In Chapter 4, we have investigated transfer learning using ImageNet [48] as a source task and hightlighted different best practices for transferring a pre-trained model to object recognition and classification tasks in computational pathology. Although transfer from ImageNet is a valid option, it has also been shown that transfer learning works best when the target task is similar to the source task [228, 137]. Whereas task similarity is hard to define formally, it is clear that the natural image domain (*i.e.* the ImageNet task) is very dissimilar to digital pathology tasks. Therefore, this question arises: could we get even better performance from transfer learning by using a digital pathology pre-trained model instead of an ImageNet one? Some works [97, 136, 106, 180] have advanced that domain-specific pre-training is indeed beneficial but to the best of our knowledge, at the time of writing the article this chapter is based on (early 2020), there was no in-depth study that attempted to answer this question in computational pathology.

A major obstacle preventing this question to be answered is the lack of a large and versatile dataset like ImageNet in digital pathology. However, the digital pathology community has made available many small and medium size datasets through challenges and publications over the years. Given its capability to learn from several tasks simultaneously, multi-task learning is a great candidate to answer our research question and, in this chapter, we investigate multi-task learning as a way of pre-training neural networks for computational pathology. Therefore, this work lies at the crossroad of multi-task and transfer learning and differs from typical contributions in those fields mostly on the objective. Indeed, we do not use multi-task learning nor transfer learning for solving a specific task but rather to pre-train a versatile network to be transferred to new tasks.

Our main contributions are as follows. **(1)** We have collected, assembled and transformed heterogeneous digital pathology datasets into a large versatile pool of classification datasets featuring 22 tasks, 81 classes and almost 900k images (see Section 5.2). **(2)** We have developed a multi-task architecture and a corresponding training scheme for creating a transferable model from these 22 tasks (see Sections 5.3.1 to 5.3.3). **(3)** We have developed a robust validation protocol based on a leave-one-task-out scheme for evaluating the transfer performance of our models compared to other approaches (see Sections 5.3.4 to 5.3.6). **(4)** We have evaluated the performance of the resulting multi-task pre-trained models compared to ImageNet ones, both when pre-trained models are used as direct feature extractors and when they are fine-tuned for each target task. We have also compared our approach to a model trained from scratch without any transfer, as well as to a multi-task learning model trained including the target dataset (see Section 5.4). **(5)** Our implementation and multi-task pre-trained models are available on GitHub[1]. These models are evaluated on an independent dataset, BreakHis [188], in Section 5.4.3. Related works can be found in Sections 2.6.4, 3.4.4, and 3.4.5.

5.2 Data

In order to build our pool of tasks, we have collected publicly available datasets (see Table 5.1) from as many sources as possible. We have also leveraged the Cytomine [132] platform to collect additional datasets annotated by our collaborators (see Appendix D). Some publicly available datasets are missing from our pool because either they could not be converted into a relevant classification problem (e.g. KimiaPath24 [11], Janowczyk tutorials 3 & 4 [86]) or we could not actually obtain them from the authors (*e.g.* dead link on download page or datasets not released yet [58]). Most

[1]`https://github.com/waliens/multitask-dipath`

Ref.	**Name**	Type	Task	Organ & pathology	Stains	Images	Classes
[170]	MITOS-ATYPIA 14	DET	Detection of mitosis and grading nuclear atypia	breast cancer	H&E	64873	3
[185]	Warwick CRC	DET	Detection and classification of nuclei	colorectal cancer	H&E	2500	2
[86]	Janowczyk1	SEG	Cell nuclei segmentation	breast cancer	H&E	31725	2
[86]	Janowczyk2	SEG	Identification of epithelium and stroma	breast cancer	H&E	3402	2
[86]	Janowczyk5	DET	Detection of mitosis	breast cancer	H&E	24870	2
[86]	Janowczyk6	CLF	Patch classification for WSI segmentation	breast, invasive ductal carci.	H&E	277524	2
[86]	Janowczyk7	CLF	Identification of lymphoma subtypes	breast, lymphoma	H&E	2244	3
[122]	Stroma LBP	CLF	Identification of epithelium and stroma	colorectal cancer	IHC	2313	2
[213]	TUPAC2016 Mitosis	DET	Detection of mitosis	breast cancer	H&E	77853	2
[8]	BACH18 Micro	CLF	Predominant cancer type classification	breast cancer	H&E	4800	4
[19]	Camelyon16	SEG	Detection of lymph nodes metastases	breast cancer	H&E	2922216	2
[94]	UMCM Colorectal	CLF	Tissue type classification	colorectal cancer	H&E	5000	8
[142]	Necrosis	CLF	Necrosed vs healthy tissue	breast cancer	IHC	882	2
[142]	ProliferativePattern	CLF	Prolif. vs non-prolif. classification	thyroid cancer	Diff-Quik	1857	2
[142]	CellInclusion	CLF	Cell inclusion vs healthy cell classification	thyroid cancer	Diff-Quik	3637	2
[142]	MouseLba	DET	Cell classification in bronchoalveolar lavage	lung cancer	MGG	4284	8
[142]	HumanLba	DET	Cell classification in bronchoalveolar lavage	lung cancer	MGG	5420	9
[142]	Lung	CLF	Tissue subtype classification	lung	H&E	6331	10
[142]	Breast1	CLF	Segmentation of cancer tissue	breast cancer	H&E	23032	2
[142]	Breast2	CLF	Segmentation of cancer tissue	breast cancer	H&E	17523	2
[130]	Glomeruli	CLF	Glomeruli recognition	kidney	M3C	29213	2
[89]	Bone marrow	CLF	Cell type classification	bone marrow	H&E	1291	9
					Total	882800	81

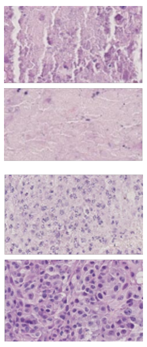
TABLE 5.1: Datasets that were used for multi-task pre-training. CLF, DET and SEG respectively stand for *classification*, *detection* and *segmentation*. H&E, IHC and M3C respectively stand for *hematoxylin and eosin*, *immunohistochemistry* and *Masson's trichrome*. *Images* and *Classes* columns give the number of images and classes of the final (possibly transformed) task for this dataset.

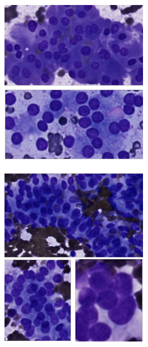
datasets in our pool are H&E stained images of human breast cancer but some other organs, pathology and stains are represented, as well as cytology samples, and animal tissues. Also missing in the pool is the BreakHis dataset which was kept aside during the development of our multi-task training protocol for final model evaluation and comparison to other transfer approaches published using the dataset.

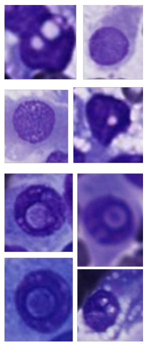
For the collected datasets to be used in a multi-task classification setting, some dataset-specific pre-processing procedures had to be executed on most of them. Applying those procedures, we have built a pool of 22 classification tasks which contains both binary and multi-class classification problems. The different pre-processing are detailed in Appendix Section B.1. Whereas we have tried to avoid intra-dataset class imbalance, there is major inter-dataset imbalance regarding the number of images: the smallest dataset contains 882 images whereas the largest one contains almost 300k. However, we believe it is not an issue and can be made of minor significance by adopting an ad-hoc multi-task training protocol (see Section 5.3.2). Selected samples of the final tasks are provided in Figure 5.1.

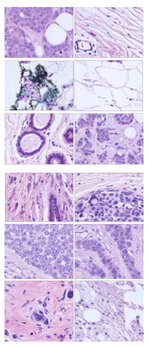
5.3 Methods

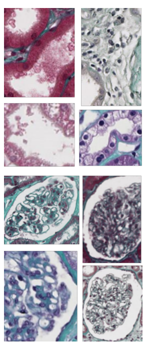
In the following section, we present the training and evaluation protocols and the experiments we have carried out. Those experiments have two main objectives. The first is to evaluate how performance of multi-task and ImageNet pre-trained networks compare when transferred to a target task. The second is to better understand how various training hyperparameters and choices impact the transfer of a multi-task pre-trained network. The multi-task architecture and training are described in 5.3.1 and 5.3.2. We present the different transfer techniques we have used in Section 5.3.3. We

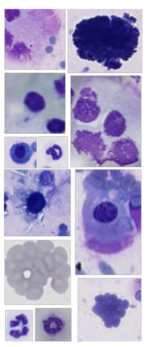

(A) Necrosis

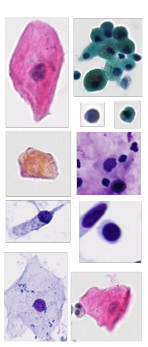

(B) ProliferativePattern

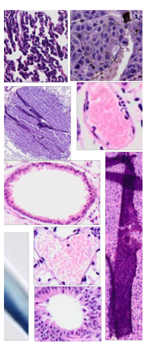

(C) CellInclusion

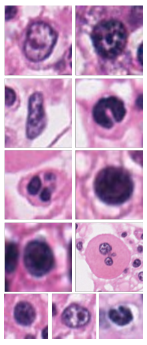

(D) Breast

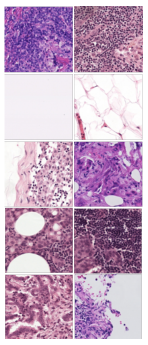

(E) Glomeruli

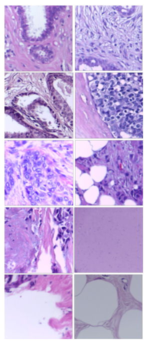

(F) MouseLba

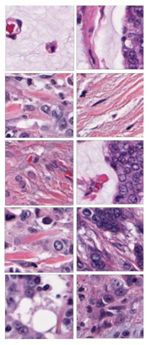

(G) HumanLba

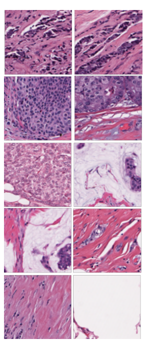

(H) Lung

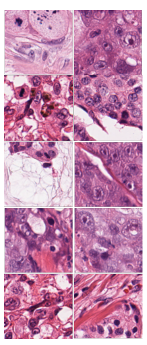

(I) BoneMarrow

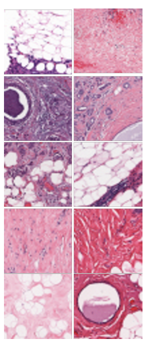

(J) Camelyon 16

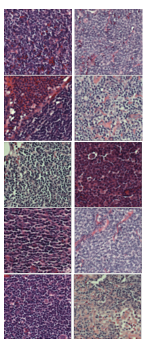

(K) BACH18 Micro

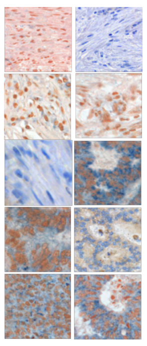

(L) Janowczyk1

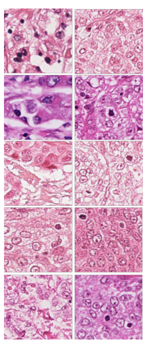

(M) Janowczyk2

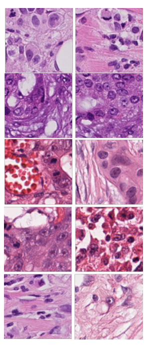

(N) Janowczyk5

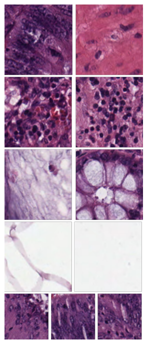

(O) Janowczyk6


(P) Janowczyk7


(Q) Stroma LBP


(R) MITOS-ATYPIA


(S) TUPAC2016 Mitosis


(T) UMCM Colorectal

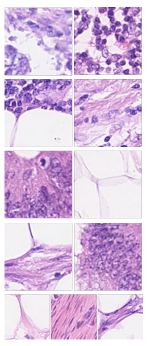

(U) Warwick CRC

FIGURE 5.1: Overview of our final classification tasks (the display size does not reflect actual image size). In this figure, we provide only one set of selected samples for Breast1 and Breast2 as their corresponding tasks are similar and they originate both from the same set of WSIs.

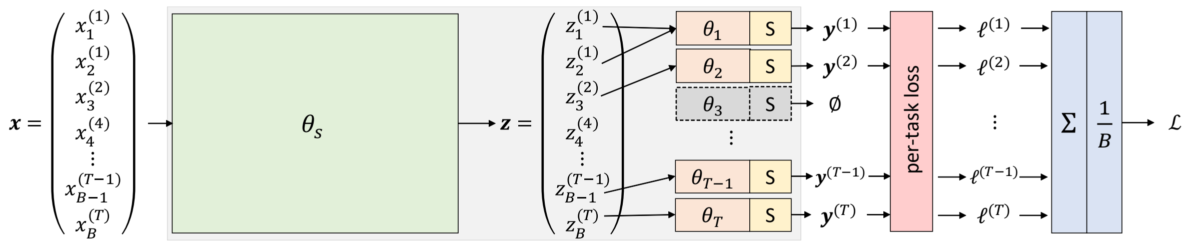


FIGURE 5.2: Multi-task architecture. $\mathcal{L}$ is the multi-task loss (see Section 5.3.2) and S is a softmax layer. $x_j^{(i)}$ and $z_j^{(i)}$ designate respectively the j^{th} sample of the batch $\mathbf{x}$ and its corresponding features produced by θ_s. This sample belongs to task t_i. Features produced for samples of a given task t_i are routed to this task head θ_i. In this example, there is no sample for task 3 in the batch $\mathbf{x}$. Therefore, the corresponding head θ_3 is inactive (*i.e.* produces no output, no parameters update, no gradient computed) for this iteration.

have developed a model evaluation and selection protocol which is described in Sections 5.3.4 and 5.3.5 whereas the various parameters we have chosen and/or evaluated as well as the experiments we have carried out are presented in Section 5.3.6.

Regarding notations, we consider a multi-task setting with a pool $\mathcal{P}$ of T classification tasks. Each task $\mathcal{D}_i$ has n_i training samples and its classification problem features C_i classes. $\mathcal{B}$ represents the set containing all samples from a batch and the batch size is denoted by $B = |\mathcal{B}|$. $\mathcal{B}_i \subseteq \mathcal{B}$ is a set that contains all the samples from task $\mathcal{D}_i$ in batch $\mathcal{B}$.

5.3.1 Multi-task architecture

The structure of our multi-task neural network is similar to those of Shang et al. [180] and Strezoski, Noord, and Worring [193] and is guided by the objective of pre-training a network for transfer. Therefore, we have adopted the architecture presented in Figure 5.2. The to-be transferred network is shared for all tasks and denoted by θ_s. We attach a head θ_i to θ_s for each task $\mathcal{D}_i$ in the pool $\mathcal{P}$. The head θ_i is simply a fully connected layer of dimensions $f_s \times C_i$ where f_s is the number of features produced by θ_s. Using such a simple layer has the benefit of making the learning capacity of the heads much lower compared to the shared network (in our experiments $|\theta_s| \gg |\theta_i|, \forall i$), hence forcing θ_s to learn relevant features for all tasks. In each head, a softmax is attached after the fully connected layer for producing per-task predictions. When forwarding samples in the multi-task network, samples of a given task $\mathcal{D}_i$ are only routed through the head θ_i, which outputs predictions for those samples.

5.3.2 Multi-task training

Classical training choices have to be adapted for the multi-task setting. Regarding batch sampling, we have decided to interleave tasks at the sample level, meaning that a single batch can contain samples from different tasks. Indeed, we believe that if batches containing samples from only a single task were alternated, the network would not see a particular task for $T-1$ iterations, with T the number of tasks, which could make the training harder when T is large.

Given the imbalance in terms of number of images per task, a batch sampling procedure had to be carefully established. Indeed, a simple random sampling across all images would prevent the tasks with fewer images from being seen during training. To overcome this issue, we selected each image in a batch by first randomly sampling a task and then sampling an image from this task, thus giving equal weights to all tasks.

Regarding the training loss, we averaged the categorical cross-entropy over all batch samples taking into account their respective tasks. More precisely, the multi-task loss $\mathcal{L}$ is computed from a batch as:

$$\mathcal{L} = \frac{1}{B}\sum_{i=1}^{T} \ell^{(i)} \text{ with } \ell^{(i)} = -\sum_{j=1}^{|\mathcal{B}_i|}\sum_{k=1}^{C_i} y_{j,k}^{(i)} \log \hat{y}_{j,k}^{(i)} \tag{5.1}$$

where $\ell^{(i)}$ is the loss for the batch samples from task $\mathcal{D}_i$ and $y_{j,k}^{(i)}$ and $\hat{y}_{j,k}^{(i)}$ are respectively the ground truth and model prediction for the j^{th} batch element and the k^{th} class of task i.

When developing a multi-task algorithm, a crucial question is what should be shared between tasks. By reducing the capacity of the heads of our architecture, we wanted to enforce the training algorithm to find generic features in θ_s that work well for all tasks. It might be interesting however to provide a way to slightly relax this implicit constraint, with a hyperparameter. To this end, during training, whereas we train the network with a learning rate γ, we choose to train the heads with a potentially different learning rate given by $\gamma_h = \gamma \times \tau_\gamma$ where $\tau_\gamma \in \mathbb{R}_{\geq 0}$ is a multiplicative factor applied to the global learning rate. This new hyperparameter provides a way of tuning the specificity/genericity of the learned shared features. Indeed, $\tau_\gamma > 1$ makes the heads learning rate larger and gives therefore more flexibility for the heads to adapt, hence relieving the shared network from learning task-specific features. Taking $\tau_\gamma = 1$ results in using the same learning rate for the whole network.

As previously mentioned, each head θ_i is randomly initialized, whereas we initialize θ_s with ImageNet pre-trained weights as it has been shown that doing so accelerates convergence in a single-task setting [142]. However, it means that trained features of θ_s are followed directly by the random layers of the heads. This is known to hurt performance in a single-task transfer setting as reported in Yosinski et al. [228] and is aggravated in a multi-task setting. Indeed, during the first training iterations, the heads gradients will be relatively large and will work to turn each head weights from random to relevant with respect to its task. However, the resulting back-propagated gradients in the last layer of θ_s will be an average of the potentially contradictory signals coming from all the heads. In order to attenuate or eliminate this problem, a simple idea consists in making each head weights relevant to its task before training the whole network by running a warm up phase during which θ_s is frozen (*i.e.* no weights update, no batch normalization update) and only the heads θ_i are trained with a learning rate γ_w.

While preparing our experiments, we have noticed two issues: one with batch normalization (see Section 2.6.2.2) and one with the heads gradients. The former is a consequence of the transfer learning settings whereas the second is a consequence of the task-based routing of samples in the heads. Both are discussed below.

5.3.2.1 Issue with batch normalization

It has been shown that a network equipped with batch normalization can exhibit issues when it is used across different domains [120, 35]. A similar issue occurs when transferring such network to one or several target tasks of which the input distributions differs greatly from the source task. Indeed, the first iteration will propagate through the network samples from an unseen and likely different distribution which will trigger a massive change of batch normalization module statistics ($\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$). However, the batch normalization trainable parameters (β and γ) will themselves be updated much more slowly (especially when the training learning rate is small) preventing them to adapt properly to the shift in distribution and statistics.

In our case, early experiments have shown that it had an undesirable negative effect on training basically destroying the purpose of transfer, as the training curves exhibited a similar behavior to training from scratch. This problem was aggravated in multi-task learning when several tasks, with different input distributions, were used. We have applied a simple procedure to attenuate this effect. Our idea consists in updating parameters β and γ of each batch normalization module before starting training such that the output of the module is preserved when the shift in distribution occurs. Given a source task $\mathcal{D}_s$ and a target task $\mathcal{D}_t$, few batches of the target task are forwarded into the network to estimate the new statistics $\mu_{\mathcal{B}_t}$ and $\sigma_{\mathcal{B}_t}$ of each batch normalization module input. Based on the obtained statistics, the new parameters β_s and γ_s for a module are given by (see below for the derivation of these formulas):

$$\gamma_t = \gamma_s \frac{\sigma^{(\epsilon)}_{\mathcal{B}_t}}{\sigma^{(\epsilon)}_{\mathcal{B}_s}} \tag{5.2}$$

$$\beta_t = \beta_s + \gamma_s \frac{(\mu_{\mathcal{B}_t} - \mu_{\mathcal{B}_s})}{\sigma^{(\epsilon)}_{\mathcal{B}_s}} \tag{5.3}$$

where $\mu_{\mathcal{B}_s}$, $\sigma^{(\epsilon)}_{\mathcal{B}_s}$, β_s and γ_s are the original source task's statistics and parameters. The expression $\sigma^{(\epsilon)}_{\mathcal{B}_x}$ denotes an altered version of standard deviation presented in the original paper which is given by $\sqrt{\sigma^2_{\mathcal{B}_x} + \epsilon}$ with a ϵ constant added for numerical stability. In our multi-task setting, we have used batches containing samples from all tasks during the new statistics estimation in order to mimic the actual inputs distributions at training time.

Deriving the formulas A batch normalization module being a composition of linear functions, it is therefore linear and can be re-expressed as:

$$y_k = BN_k(x) = m_k x + p_k \tag{5.4}$$

where y_k and x respectively denote the output of the batch normalization module for task k (*i.e.* using task k statistics and parameters) and the input of the batch normalization module. Using the definition of the module, Equation 5.4 can be rewritten as:

$$y_k = \underbrace{\frac{\gamma_k}{\sigma^{(\epsilon)}_{\mathcal{B}_k}}}_{m_k} x + \underbrace{\beta_k - \gamma_k \frac{\mu_{\mathcal{B}_k}}{\sigma^{(\epsilon)}_{\mathcal{B}_k}}}_{p_k} . \tag{5.5}$$

The formula in Equations 5.2 and 5.3 are obtained by ensuring $y_t = y_s$ for any x, or similarly solving the following system:

$$\begin{cases} m_s = m_t \\ p_s = p_t \end{cases} \tag{5.6}$$

5.3.2.2 Issue with heads gradients

Because samples are routed through their respective task head in our architecture, all parts of the network do not see the same number of samples from a batch which causes the gradients to be underestimated in the network heads. This can be shown by developing the derivative of our loss with respect to one of the logits of a head. Let $r_k^{(i)}$ be the class k logit of task t_i. The derivatives of the loss $\mathcal{L}$ (see Equation 5.1) with respect to $r_k^{(i)}$ is given by:

$$\frac{\partial \mathcal{L}}{\partial r_k^{(i)}} = -\frac{1}{B} \sum_{j=1}^{|\mathcal{B}_{t_i}|} \frac{\partial \ell_j^{(i)}}{\partial r_k^{(i)}}, \tag{5.7}$$

where $\ell_j^{(i)}$ is the loss term for the jth sample of task t_i in the batch sample. Equation 5.7 shows that the gradients are divided by the batch size although they are estimated using $|\mathcal{B}_{t_i}|$ samples (*i.e.* the number of samples from task t_i in the batch). This applies also to the gradients used for updating the parameters θ_i of the head. Given a task t_i, this magnitude reduction has the same effect as dividing the learning rate for head θ_i by a variable factor that depends on the number of samples of task t_i that are present in the batch. If tasks are sampled uniformly to create a batch, one can expect this factor to be equal to the number of tasks T on average. In order to avoid this phenomenon, we applied a simple trick that consists in re-scaling the gradients of each head θ_i by multiplying them by ϕ_{t_i}:

$$\phi_{t_i} = \frac{B}{|\mathcal{B}_{t_i}|}. \tag{5.8}$$

5.3.3 Transferring a multi-task pre-trained network

We study both deep transfer learning approaches, namely feature extraction and fine-tuning. In both cases, θ_s is pre-trained on some source task(s), either ImageNet or several tasks simultaneously in the MTL setting. For feature extraction, we exclusively use the pre-trained network θ_s to extract features for all images of the target task. The extracted features can then be used to learn a third-party classifier, a common choice being a linear SVM. For fine-tuning, we further train the shared network θ_s on the target task by attaching to it a fully connected layer and a softmax. The resulting network is trained using for instance stochastic gradient descent

Both approaches have their own complementary advantages and drawbacks. As mentioned above, feature extraction uses a linear model which is very fast to train and makes it robust to overfitting when working with small target datasets. However, using fixed pre-trained features makes it possible that the features are not entirely suited for the target task (*e.g.* ImageNet vs. digital pathology), yielding suboptimal performance. Fine-tuning does not suffer from this drawback as the whole network (or a part of it) is retrained on the target task. When using large capacity networks (*e.g.* ResNet or DenseNet), it allows the network to capture and learn task-specific features. This is however an issue when the target dataset is small because the large capacity of the network can lead to overfitting.

5.3.4 Evaluating transferability for hyperparameter tuning

Given a set of tasks available to pre-train a MTL model for future transfer, either by feature extraction or fine-tuning, a question left is how to tune the hyperparameters to train this model. Since we want to optimize the transferability of the model, rather than for example its average performance on the training tasks, we have to design a specific evaluation protocol and a specific metric to assess this transferability for each hyperparameter combination.

For this purpose, we have developed a *leave-one-task-out* (LOTO) cross-validation scheme, inspired from leave-one-out cross-validation. It consists in removing a set $\mathcal{T} \subset \mathcal{P}$ of one or more tasks from the training pool $\mathcal{P}$, training a multi-task model on $\mathcal{P} \setminus \mathcal{T}$ and then evaluating how the learned models transfer to the tasks of $\mathcal{T}$. This operation can then be repeated for different $\mathcal{T}$ to increase the stability of the analysis. In our case, we have picked $\mathcal{T}$ to contain only one task when possible. However, it is important that tasks that are closely related are left out together during LOTO cross-validation to avoid data leakage (see Section 3.4.1). In our case, there are two pairs of datasets that are subject to this exception. The first is CellInclusion and ProliferativePattern which are different classification tasks coming from the same WSIs. The second is Breast1 and Breast2 which are the same classification tasks generated with different rules from the same expert annotations. Therefore, applying LOTO exhaustively in our settings leads to 20 possible left out sets $\mathcal{T}$.

The optimal hyperparameter combination might arguably depend on whether the MTL model will be used for feature extraction or fine-tuning. We have however solely

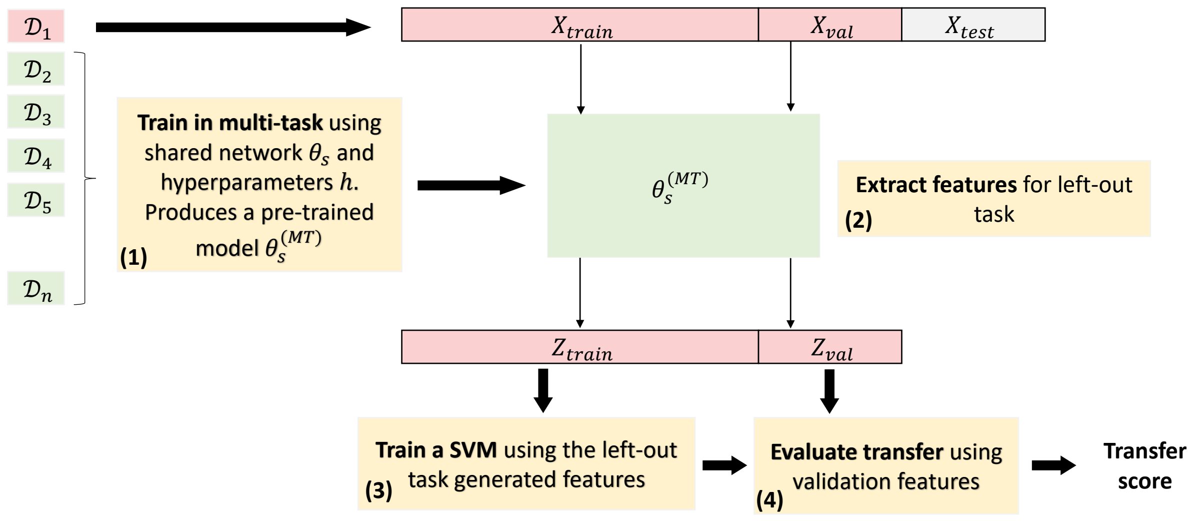
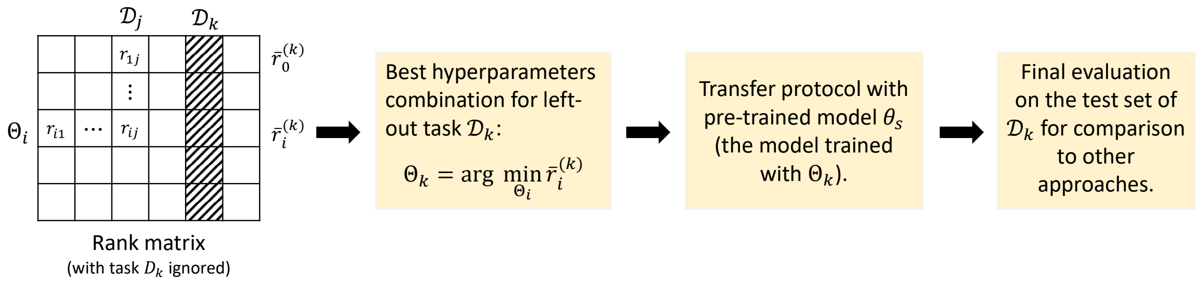

$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_3$
$\mathcal{D}_4$
$\mathcal{D}_5$
$\mathcal{D}_n$
X_{train}
X_{val}
X_{test}
(1) **Train in multi-task** using shared network θ_s and hyperparameters h. Produces a pre-trained model $\theta_s^{(MT)}$
$\theta_s^{(MT)}$
(2) **Extract features** for left-out task
Z_{train}
Z_{val}
(3) **Train a SVM** using the left-out task generated features
(4) **Evaluate transfer** using validation features
Transfer score

FIGURE 5.3: The transfer performance evaluation with LOTO cross-validation.

used feature extraction performance as a proxy to evaluate transferability, mainly because we wanted to release a single MTL model for simplicity, but also to reduce the computational costs of our experiments. More precisely, given a left-out set $\mathcal{T}$ and one of its task $\mathcal{D} \in \mathcal{T}$, we have evaluated transferability of a multi-task pre-trained network trained on $\mathcal{P} \setminus \mathcal{T}$ by using the resulting θ_s as a feature extractor on $\mathcal{D}$. The training set of $\mathcal{D}$ was used to train the features classifier (*i.e.* a linear SVM, see Section 5.3.6 for details) and the validation set was used to evaluate it. Transfer performance was evaluated by the accuracy (ACC) for multi-class classification tasks and the area under the ROC curve (ROC AUC) for binary classification tasks. To cope with the randomness induced by heads initialization and mini-batch sampling, each training of a MTL model was repeated with 5 different random seeds. Note that, at this stage, the test set of $\mathcal{D}$ was kept aside for future comparison of a selected multi-task pre-trained network and comparison to ImageNet transfer (see Section 5.3.5). This process is illustrated in Figure 5.3.

All the scores resulting from the same hyperparameters combination but different random seeds can be averaged and the resulting average performance can be used to assess transfer performance on a given task $\mathcal{D}$. We need however to aggregate these scores over all (left-out) tasks to assess the overall transferability of a given hyperparameter combination. Averaging ACC and ROC AUC scores over tasks is in general not a good idea as these values depends on the task difficulty and are not directly comparable across tasks. We propose instead to aggregate rankings. More precisely, for each task, the combination of hyperparameters leading to the best model on average was assigned rank 1 and the worst was assigned the maximum rank. Applying this procedure to all our sets $\mathcal{T}$ produces a rank matrix where r_{ij} is the rank of the i^{th} combination of hyperparameters evaluated on the left-out task j. The best hyperparameter combination is then defined as the one that minimizes the average rank over



$\mathcal{D}_j$
$\mathcal{D}_k$
r_{1j}
r_{i1}
r_{ij}
Θ_i
$\bar{r}_0^{(k)}$
$\bar{r}_i^{(k)}$
Rank matrix (with task $\mathcal{D}_k$ ignored)
Best hyperparameters combination for left-out task $\mathcal{D}_k$: $\Theta_k = \arg\min_{\Theta_i} \bar{r}_i^{(k)}$
Transfer protocol with pre-trained model θ_s (the model trained with Θ_k).
Final evaluation on the test set of $\mathcal{D}_k$ for comparison to other approaches.

FIGURE 5.4: The final performance evaluation process for comparison of multi-task pre-training to other transfer approaches. $\bar{r}_i^{(k)}$ is the average rank for the hyperparameters combination Θ_i on task $\mathcal{D}_j$ computed ignoring task $\mathcal{D}_k$ (or, alternatively, by ignoring the rank r_{ik} in Equation 5.9). For feature extraction, the training and validation set of $\mathcal{D}_k$ are used for 5-fold cross-validation. For fine-tuning, the training set is used for training the model and the validation set is used for model selection and hyperparameters tuning (see Section 5.3.6).

all left-out tasks:

$$\bar{r}_i = \frac{1}{T_{\text{out}}} \sum_{j=1}^{T_{\text{out}}} r_{ij} \tag{5.9}$$

where T_{out} is the number of left-out tasks.

5.3.5 Final performance evaluation

Our main objective is to compare transfer from multi-task and ImageNet pre-trained networks. In principle, two nested LOTO cross-validation loops should be adopted to carry out such comparison: for each left-out task in the external LOTO CV loop, an additional internal LOTO CV loop should be run to find the optimal hyperparameter combination for training the MTL model to be transferred to the (external) left-out task. Using two nested loops would be however too expensive computationally[2]. We have instead adopted the following simplified scheme. A single LOTO cross-validation is run as described in the previous section. Given a left-out task $\mathcal{D}_k$, we select the hyperparameter combination that minimizes the average rank but now excluding task $\mathcal{D}_k$ from the average computation (*i.e.* excluding r_{ik} from the calculation of $\bar{r}_i$ in Equation 5.9). All the models trained using this hyperparameters combination (*i.e.* one per seed) are transferred to the target task $\mathcal{D}_k$ using both transfer protocols (*i.e.* feature extraction or fine-tuning). The test set of $\mathcal{D}_k$ is used solely to evaluate the resulting transfer performance whereas the training and validation sets can be used by the transfer protocol, feature extraction or fine-tuning, for training and hyperparameter tuning (see Section 5.3.6). This process is described in Figure 5.4.

Unlike with a true double LOTO CV loop, the training and validation sets of the left-out task $\mathcal{D}_k$ are used, in our simplified scheme, to train the MTL models that are

[2]The simplified scheme we present hereafter has already yielded approximately 20k GPU hours of computation. This computation time would have been multiplied by 10 using two nested LOTO CV loops, which was impossible for us to carry out given our available computing resources.

Parameters		Values	Count
Learning rate (LR)	γ	$\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$	4
LR multiplier	τ_γ	$\{1, 5, 10\}$	3
Shared network	θ_s	$\{\text{ResNet50}, \text{DenseNet121}\}$	2
Warm up	w	$\{true, false\}$	2
Number of combinations $\|\mathcal{H}\|$			48
with random seeds			240

TABLE 5.2: The multi-task training parameters evaluated using the cross-validation procedure. Using the LOTO scheme, 240 trained models should be transferred to each left out dataset. $\mathcal{H}$ is the set of all hyperparameters combinations.

Task	**Train size**	**Full name**	**Metric**
BoMa	652	BoneMarrow	*Accuracy*
MoLb	2438	MouseLba	
HuLba	4397	HumanLba	
Lung	5443	Lung	
Nec	791	Necrosis	*ROC AUC*
PrPa	1346	ProliferativePattern	
CeIn	1816	CellInclusion	
Glom	14605	Glomeruli	
Br2	14953	Breast1	
Br1	18261	Breast2	

TABLE 5.3: Evaluation tasks, their evaluation metric, training set size and full name.

transferred to the other tasks for computing the rankings of the hyperparameters combinations for these tasks. However, this is not expected to introduce any bias since the data from task $\mathcal{D}_k$ is neither used to decide on the optimal hyperparameter setting of the MTL model transferred to $\mathcal{D}_k$ itself (since r_{ik} is excluded from the computation of $\bar{r}_i$), nor to train this MTL model.

5.3.6 Hyperparameters settings and experiments

The hyperparameters we have studied and their evaluated values are listed in Table 5.2. Parameters values and training choices were established based on early experiments which evaluated multi-task training stability and convergence. Regarding the selected range of learning rates, values higher than 10^{-3} resulted in very unstable or diverging trainings whereas values lower than 10^{-6} prevented convergence. As shared network θ_s, we have used two popular architectures ResNet50 [77] and DenseNet121 [81]. We have removed the fully connected layer of those networks and replaced it by a global average pooling. Moreover, we have loaded the networks with ImageNet pre-trained weights from PyTorch [149].

The multi-task network was trained for 50k iterations using batches of size 64 and SGD as optimizer using momentum set to its default value (*i.e.* 0.9), learning rate γ and heads learning rate multiplier τ_γ. Either the whole network was trained directly, or the heads were first warmed up for 5k iterations with learning rate $\gamma_w = 10^{-3}$ before the whole network was trained for 45k iterations.

Classical data augmentation and normalization have been applied to the input images. We have used ImageNet statistics for normalizing the images as early experiments have shown no significant improvement by normalizing with per-task statistics (*i.e.* mean and variance). As data augmentation, we have applied simple random vertical and horizontal flips as well as extraction of a random square crop (if the image is not square already).

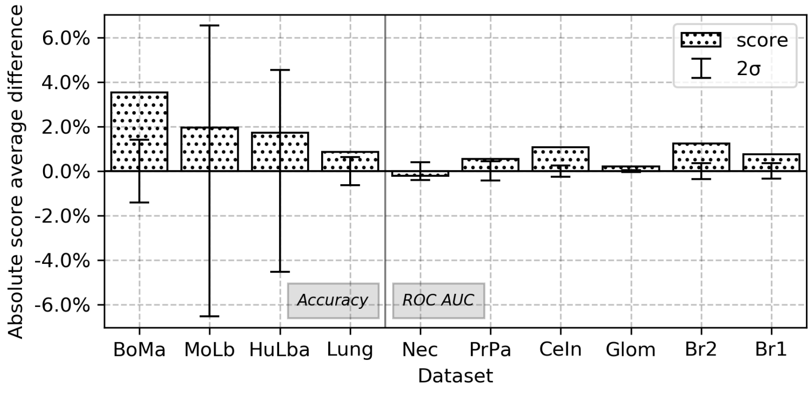
When feature extraction was applied either during the evaluation or selection, we have used linear SVM [55] as feature classifier. Whenever a SVM classifier was trained, we have tuned the C regularization parameter among the following values $\{10^{-10}, 10^{-9}, ..., 10^{-1}, 1\}$ by 5-fold cross-validation. The tasks were splitted into folds based on the most relevant information available for the task (patient, slide or image).

Regarding fine-tuning, we have trained the network for 100 epochs on the training set of the target task and have tuned the learning rate among $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ and have selected the best epoch on the validation set. When transferring from our multi-task pre-trained models, we have performed the selection of the best multi-task models as explained in Section 5.3.4. Then, for each model trained with the best hyperparameters (*i.e.* one per seed), we have trained one model per fine-tuning hyperparameters combination. For ImageNet, the approach was slightly different as we had only one model to start from. In this case, we have used 5 different seeds for fine-tuning.

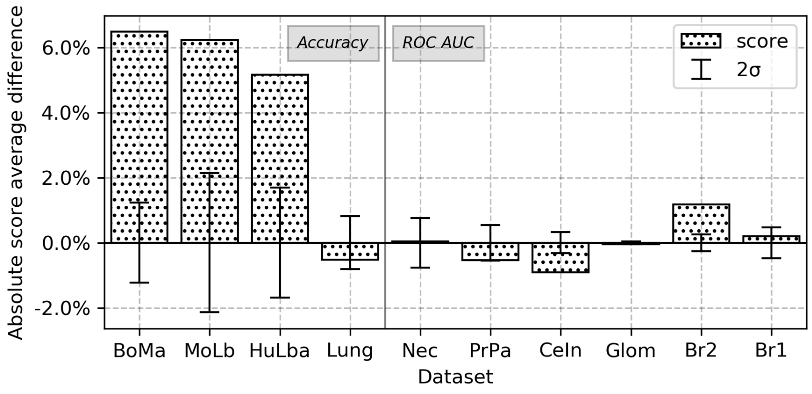
At this point, it is important to note that LOTO cross validation is quite demanding in terms of computing resources as, for all $\mathcal{T}$, all the combinations of hyperparameters and random seeds have to be evaluated. As indicated in Table 5.2, 240 models would have to be trained per set $\mathcal{T}$ of excluded tasks which, given 22 tasks, yields 4800 multi-task trainings. Due to limited availability of computing resources, we had to reduce this number and have done so by reducing the number of left out tasks used in our analysis. In particular, we have kept 10 tasks in 8 sets: $\{\text{Breast1}, \text{Breast2}\}$, $\{\text{CellInclusion}, \text{ProliferativePattern}\}$, $\{\text{MouseLba}\}$, $\{\text{HumanLba}\}$, $\{\text{Necrosis}\}$, $\{\text{Lung}\}$, $\{\text{Glomeruli}\}$ and $\{\text{BoneMarrow}\}$ (see Table 5.3). All other tasks were always incorporated however to train each multi-task model.

For the sake of completeness, we have also compared the transfer learning approaches with training from scratch and joint training. The former consists in training a network initialized with random weights. The latter consists in training a network in multi-task using the whole pool of tasks (including the target tasks).

For training from scratch, we have used the same settings as for the fine-tuning experiment except for the network weights initialization (using initialization strategy as defined in PyTorch): same evaluated learning rates, number of training epochs and same networks. Regarding joint training, we have used the same architecture and training algorithm as for our multi-task pre-training. For the evaluation tasks



(A) DenseNet121



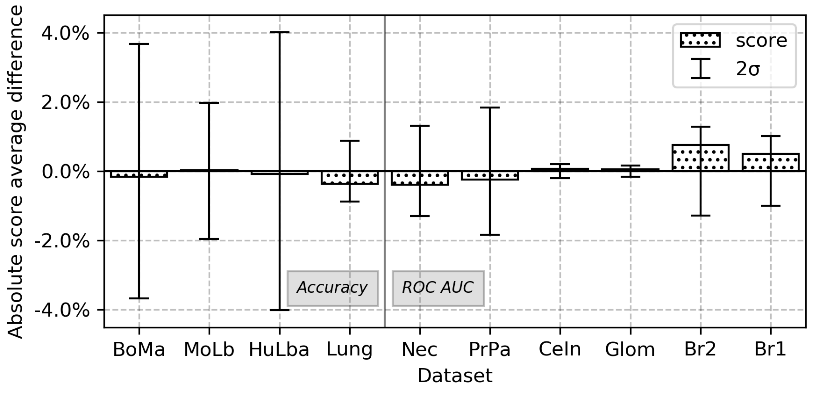
(B) ResNet50

FIGURE 5.5: Absolute score difference between multi-task versus ImageNet pre-training using **feature extraction** as transfer protocol on our ten evaluation tasks. Positive difference indicates that multi-task pre-training yield superior performance. Tasks are sorted by evaluation metric and increasing dataset size. The variability of the multi-task transfer is measured using ± 2 standard deviations given by the error bars.

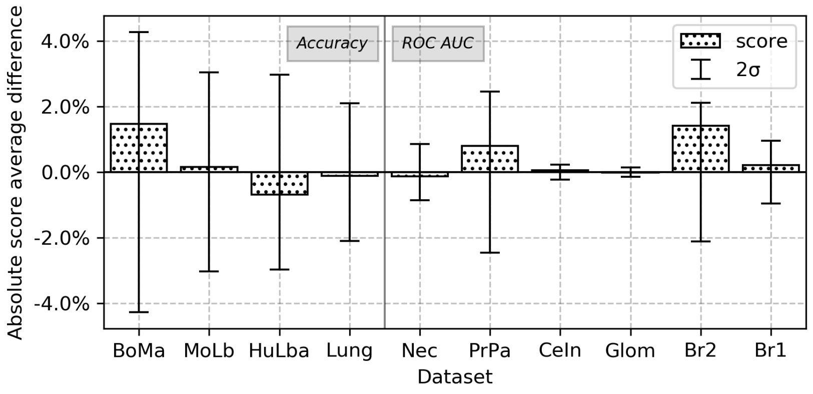
listed above, only their training set is used for the multi-task training, while their validation and test sets were respectively used for optimizing the hyperparameters and evaluating the selected model. The data for the other training tasks were kept the same as for multi-task pre-training.

5.4 Results and discussion

We report how our methods compare to transfer from ImageNet, training from scratch and joint training in Section 5.4.1. Then, we study the effect on transfer of the various evaluated multi-task training hyperparameters in Section 5.4.2. Finally, we discuss our results and future works in Section 5.4.4.



(A) DenseNet121



(B) ResNet50

FIGURE 5.6: Absolute score difference between multi-task versus ImageNet pre-training using **fine-tuning** as transfer protocol on our ten evaluation tasks. See Figure 5.5 for details. Error bars are computed using ±2 the largest standard deviation among the ones resulting from ImageNet and multi-task fine-tuning.

5.4.1 Transfer performance

As explained in Section 5.3.4, we have used both feature extraction and fine-tuning to evaluate transfer performance. We could not repeat our experiments with several datasets splits given the computational cost, which would have allowed us to perform formal statistical tests for method comparison. To ease the discussions below, we will nevertheless call significant any difference between two average errors that exceed, in absolute value, twice the maximum of their standard deviations. If the scores were Gaussian distributed, this would ensure that each average score is outside a 95%-confidence interval around the other score.

Absolute score differences between feature extraction from ImageNet and multi-task pre-trained networks can be found in Figures 5.5a and 5.5b for DenseNet121 and ResNet50 respectively. Our DenseNet121 features yield superior scores for nine datasets out of ten (all but *Necrose*) of which superiority is significant for all but two

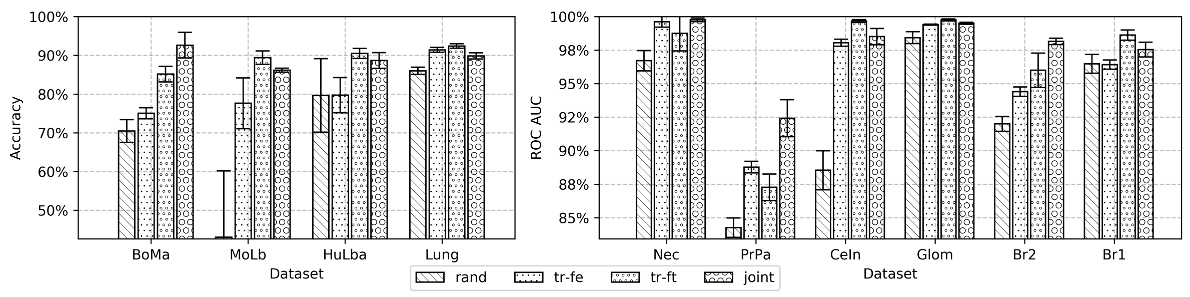


FIGURE 5.7: Performance comparison of different approaches using DenseNet121: training from scratch (*rand*), feature extraction (*tr-fe*) and fine-tuning (*tr-ft*) using our multi-task pre-trained networks and joint training (*joint*). Each bar is the average performance and its error bar gives ±2 standard deviation of a given method over five runs. For exact scores, see Table 5.4.

	Target task	Train size	Scratch	Feature extraction		Fine tuning		Joint training
				ImageNet	Multi-task	ImageNet	Multi-task	
Accuracy	BoMa	652	70.49 ± 1.47	71.52	75.05 ± 0.70	85.32 ± 1.84	85.16 ± 1.01	92.62 ± 1.66
	MoLb	2438	43.04 ± 8.56	75.68	77.63 ± 3.27	89.39 ± 0.98	89.41 ± 0.87	86.17 ± 0.26
	HuLba	4397	79.65 ± 4.75	78.01	79.73 ± 2.27	90.56 ± 2.01	90.48 ± 0.64	88.67 ± 1.01
	Lung	5443	86.01 ± 0.47	90.54	91.40 ± 0.32	92.77 ± 0.44	92.41 ± 0.29	89.86 ± 0.39
ROC AUC	Nec	791	96.71 ± 0.38	99.82	99.61 ± 0.20	99.14 ± 0.43	98.75 ± 0.65	99.76 ± 0.07
	PrPa	1346	84.26 ± 0.36	88.22	88.77 ± 0.21	87.51 ± 0.92	87.27 ± 0.49	92.42 ± 0.69
	CeIn	1816	88.54 ± 0.73	96.97	98.05 ± 0.13	99.60 ± 0.10	99.67 ± 0.05	98.51 ± 0.30
	Glom	14605	98.43 ± 0.22	99.20	99.40 ± 0.02	99.70 ± 0.08	99.75 ± 0.04	99.50 ± 0.04
	Br2	14953	92.00 ± 0.28	93.17	94.40 ± 0.18	95.24 ± 0.59	96.00 ± 0.64	98.15 ± 0.12
	Br1	18261	96.48 ± 0.35	95.66	96.42 ± 0.17	98.12 ± 0.50	98.62 ± 0.19	97.54 ± 0.27

TABLE 5.4: Performance of different evaluated approaches on DenseNet121. All reported scores are percentages. We compare training from scratch with feature extraction and fine-tuning from our multi-task pre-trained and the ImageNet models. We also provide results of the joint training experiment. Average performance and (±1) standard deviation are provided for all methods except for feature extraction from ImageNet as this procedure is deterministic.

(*HumanLba* and *MouseLba*). The score difference is in favor of ImageNet features on *Necrose* although it is not significant. ResNet50 features yield superior results for six out of ten datasets (all but *CellInclusion*, *Glomeruli*, *ProliferativePattern* and *Lung*) of which superiority is significant for all but two datasets (*Necrose* and *Breast1*). Out of the four datasets where ImageNet features yield superior scores, the difference is significant for only one of them (*CellInclusion*). It is interesting to note that the largest difference of scores in favor of ImageNet is only 0.21% (ROC AUC) on *Necrose* for DenseNet121 whereas the difference in favor of our features is at most 3.52% (ACC) on *BoneMarrow*. Similary with ResNet50, the largest differences are 0.91% (ROC AUC) on *CellInclusion* and 6.48% (ACC) on *BoneMarrow* respectively in favor of ImageNet features and ours. Therefore, it appears that the loss of performance when our features underperform compared to ImageNet is lower than the expected gain of performance when our features are better. This indicates that the performance gain or loss you would obtain with multi-task features are dataset dependent and is hard to predict

	Target task	Train size	Scratch	Feature extraction		Fine tuning		Joint training
				ImageNet	Multi-task	ImageNet	Multi-task	
Accuracy	BoMa	652	65.88 ± 1.16	71.52	78.00 ± 0.61	83.57 ± 2.14	85.04 ± 0.37	92.46 ± 1.23
	MoLb	2438	48.47 ± 4.19	73.08	79.31 ± 1.07	88.20 ± 1.52	88.36 ± 1.27	88.24 ± 0.54
	HuLba	4397	71.85 ± 8.33	77.81	82.97 ± 0.84	90.64 ± 1.49	89.95 ± 0.61	88.96 ± 0.85
	Lung	5443	84.75 ± 0.76	91.67	91.15 ± 0.41	92.73 ± 0.33	92.61 ± 1.05	90.89 ± 0.35
ROC AUC	Nec	791	97.21 ± 0.76	99.17	99.21 ± 0.38	99.21 ± 0.34	99.08 ± 0.43	99.81 ± 0.10
	PrPa	1346	83.00 ± 1.83	90.00	89.47 ± 0.27	87.80 ± 1.23	88.60 ± 0.72	93.92 ± 0.60
	CeIn	1816	84.44 ± 0.78	97.91	97.00 ± 0.16	99.59 ± 0.11	99.65 ± 0.12	98.69 ± 0.20
	Glom	14605	98.24 ± 0.08	99.41	99.36 ± 0.02	99.79 ± 0.03	99.78 ± 0.07	99.48 ± 0.09
	Br2	14953	92.48 ± 0.90	93.57	94.75 ± 0.13	94.26 ± 1.06	95.67 ± 1.03	98.29 ± 0.09
	Br1	18261	95.84 ± 0.48	96.49	96.68 ± 0.24	97.64 ± 0.48	97.85 ± 0.32	96.96 ± 0.19

TABLE 5.5: performance of different evaluated approaches on ResNet50. See Table 5.4 for explanation.

a priori, although the loss of performance is usually small compared to the possible improvement. Another interesting observation is the stability of transfer performance as only four evaluations (out of 20) exhibit standard deviations larger than 0.5% (ROC AUC or ACC).

Regarding fine-tuning, our features outperform ImageNet ones for five and six datasets with DenseNet121 and ResNet50 respectively (see Figures 5.6a and 5.6b). However, none of the differences are significant whether or not the advantage is in favor of our approach.

As a summary, feature extraction transfer approach seems to benefit from multi-task pre-training as 15 evaluations (out of 20) are in favor of our approach of which 11 are significant. Only two evaluations are significantly in favor of ImageNet features. However, fine-tuning from our features yield comparable performance with ImageNet features initialization as no score difference is significant (11 evaluations are in favor of our approach).

As an additional experiment, we compare transfer by feature extraction and fine-tuning with a similar model trained from scratch and the joint MTL approach described in Section 5.3.6 (see Figure 5.7 and Tables 5.4 and 5.5). These experiments show that fine-tuning improves over feature extraction significantly on most datasets which confirms our conclusions from Chapter 4. Moreover, they show that training from scratch is subpar compared to the transfer approaches, feature extraction included. This last observation confirms previously published results [156, 199, 182]. It appears that joint training significantly improves the performance on small datasets (*BoneMarrow* and *ProliferativePattern*) over all other approaches. For larger datasets, performance seem to lie between fine-tuning and feature extraction, or to be on par with fine-tuning on the datasets where the task is almost solved (ROC AUC or ACC close to 1).

5.4.2 Study of multi-task training hyperparameters

Our experiments have shown that the hyperparameter impacting transfer performance the most was the multi-task training learning rate. Figure 5.8 shows the distributions of scores per learning rate using DenseNet121 and *HumanLba* dataset. We have picked

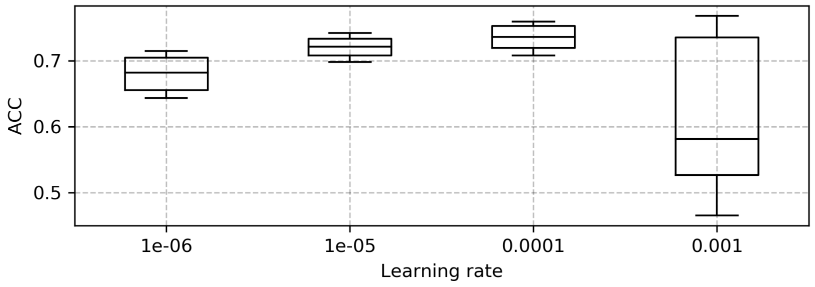


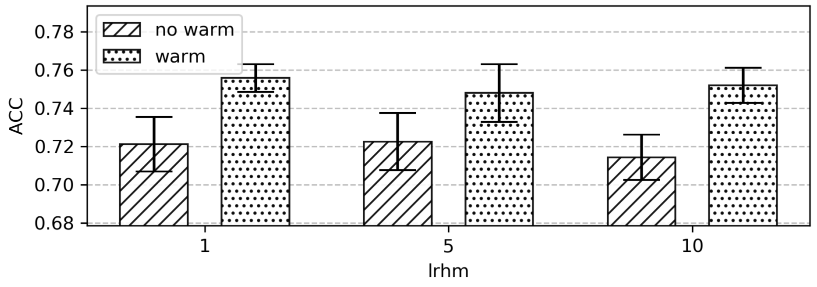
FIGURE 5.8: Distributions of scores per learning rate on DenseNet121 with HumanLba dataset. Each boxplot results from the aggregation of the transfer scores of all models using the same learning rate value on the given network and dataset.

this plot specifically because it exhibits the most frequent pattern regarding the effect of the learning rate. Similar plots for other datasets as well as ResNet50 can be found in Appendix Figure B.4. We have observed that the highest learning rate 10^{-3} yielded highly variable performance which were most of the time inferior compared to lower learning rates. It indicates that this specific value is too high to cope with the multi-task setting as it prevents the models from making use of the tasks information efficiently. This is likely due to training instabilities (convergence issues, noisy training loss) and overfitting. It appears that the lowest learning rate 10^{-6}, although yielding more stable performance, generally underperforms higher learning rates 10^{-5} and 10^{-4}. For both networks and most datasets, the latter learning rate 10^{-4} is the best performing on average.

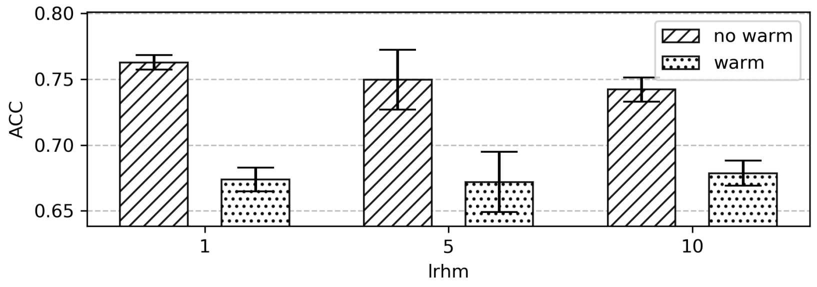
The impact of the two other hyperparameters τ_γ and w seems to be minor for most datasets as variation stays within two standard deviations. Moreover, there is no pattern emerging from our experiments regarding those hyperparameters in general. Two exceptions to this observation are the *HumanLba* and *MouseLba* datasets which exhibit significant performance variations on both networks. The variations are shown in Figure 5.9 (similar figures can be found for other networks and datasets in Appendix Figures B.1 and B.2). Those Figures show that *HumanLba* benefits from warming up whereas *MouseLba* performance are hurt. This indicates that the effect on transfer performance of warming up and multiplying heads learning rate is very dataset dependent and no general rule can be drawn from our experiments.

5.4.3 Evaluation of released multi-task pre-trained models

With the idea of releasing the best multi-task pre-trained models to the community, we have re-trained each network one time using the same procedure described in Section 5.3.2 and the best hyperparameters found using the ranks. Especially, for DenseNet121, we have picked $\gamma = 10^{-4}$ (learning rate), $\tau_\gamma = 5$ (learning rate heads multiplier) and warm up. For ResNet50, we have picked $\gamma = 10^{-3}$, $\tau_\gamma = 1$ and warm up. We have used all the data available for re-training (*i.e.* the training, validation and test sets of our 22 tasks).



(A) HumanLba



(B) MouseLba

FIGURE 5.9: Transfer performance for combinations of the hyperparameters γ_τ (learning rate heads multiplier, *lrhm*) and w (warm up) on two different datasets with learning rate $\gamma = 10^{-4}$ on DenseNet121. Error bars report twice the standard deviation.

In order to evaluate the to-be released models, we kept the BreakHis dataset [188] out of our pool. We have transferred the two resulting models using both feature extraction and fine-tuning as presented in the article. The transfer was repeated on the five original per-patient folds of BreakHis and performance was averaged over those folds. The resulting transfer scores are given in Table 5.6. We have not compared our approach to the full BreakHis benchmark but only to similar methods of transfer learning of which we have found two [189, 187] that perform feature extraction.

Our conclusions do not change. For feature extraction, our approach either improves over ImageNet transfer or provides similar performance. We have observed that our method yields better performance on higher magnification in particular. Our transfer learning approach is also on par with similar ones from the literature.

For fine-tuning, there does not seem to be a significant difference between our approach and ImageNet. Surprisingly, we have also observed that fine-tuning (from ImageNet or our models) yielded inferior performance compared to feature extraction in some cases which could be explained by the fact that our protocol requires a validation set for tuning the fine-tuning hyperparameters. To avoid reducing too much the size of the training set, we have extracted a small validation set (approximately 10% of the whole data) which might be too small for robust selection, especially when coupled with a reduced training set.

Tr.	Network	Source	40x	100x	200x	400x
FE	DenseNet121	ImageNet	85.83 ± 2.58	85.38 ± 4.14	84.50 ± 1.73	84.81 ± 1.26
		Multi-task	86.05 ± 2.51	84.74 ± 3.83	85.75 ± 2.64	87.22 ± 1.65
	ResNet50	ImageNet	85.42 ± 3.56	84.20 ± 3.18	86.40 ± 3.45	82.64 ± 1.16
		Multi-task	84.77 ± 3.76	83.95 ± 2.25	89.15 ± 4.40	86.86 ± 2.76
	[189] (Decaf)		84.00 ± 6.90	83.90 ± 5.90	86.30 ± 3.50	82.10 ± 2.40
	[187] (VGG-VD)		86.90 ± 5.20	85.40 ± 5.70	85.20 ± 4.40	85.70 ± 8.80
FT	DenseNet121	ImageNet	83.64 ± 5.78	85.99 ± 4.22	89.07 ± 3.45	85.38 ± 3.89
		Multi-task	82.69 ± 6.22	86.32 ± 1.08	90.91 ± 3.07	85.74 ± 3.44
	ResNet50	ImageNet	84.43 ± 5.48	83.13 ± 2.76	88.96 ± 3.27	84.08 ± 2.39
		Multi-task	85.83 ± 4.05	84.51 ± 3.46	87.99 ± 3.34	84.10 ± 4.00

TABLE 5.6: Transfer performance of our best multi-task pre-trained model on the BreakHis datasets using **feature extraction** (FE) and **fine-tuning** (FT). Average per-patient accuracies and standard deviations are given per magnification (x40, x100, x200 and x400).

5.4.4 Discussion and future works

Features extracted from our models are in general superior to ImageNet ones which shows that multi-task pre-training is effective at creating task-specific features. This important observation confirms the conclusions of previous works that domain-specific pre-training is a good idea and also validates the multi-task approach when a large source dataset is not available.

The second important observation is that fine-tuning does not benefit from using our models as we obtain similar transfer performance whatever the source. This indicates that fine-tuning is able to recover the lack of specificity of ImageNet features compared to ours. It contradicts our initial hypothesis that transfer should work better when source and target domains are similar. There might be several reasons why we observe such phenomenon. First, Yosinski et al. [228], who concluded about the importance of task similarity for efficient transfer, performed their experiments on simple architectures (*e.g.* simple stack of convolutional layers). In our case, we have used more recent architectures (*i.e.* residual and densely connected networks) which are easier to train and might be less impacted by their initial weights. Second, most previous works have shown that fine-tuning was beneficial in a single source task transfer scenario exclusively. Our multi-task pre-training is able to learn specific features but we cannot exclude that a more advanced approach could result in even stronger features. This might change our conclusion that fine-tuning does not benefit from MTL transfer. For instance, we have applied a minimal augmentation procedure with random flipping and cropping. Therefore, it would be interesting to consider other augmentation techniques to hopefully reinforce our model: color perturbation, blurring, *etc*. Some contributions have highlighted training difficulties associated with multi-task (*e.g.* gradients interference [230]) and transfer learning (*e.g.* batch normalization [35]) that could be investigated in our context. On a similar note, it is likely that, given a target task, not all tasks in the pool contribute equally to transfer performance. A catastrophic interference phenomenon might even be at play and some tasks might have a destructive effect during the pre-training phase (*i.e.* if they were removed, transfer performance would increase). This could be caused, for instance, by significant differences in the input distributions of our tasks. As stated in Section 5.3.6, we have applied the same normalization strategy for all tasks. A per-task normalization (other than standard) might help with catastrophic forgetting with, for instance, advanced techniques specific to pathology like stain normalization [91, 175, 240]. Similary, incorporating a per-task training mechanism that could dynamically increase (*resp.* decrease) the contribution of constructive (*resp.* destructive) tasks would certainly help improving the resulting features. Alternatively, instead of using all the tasks, one could find a mechanism for selecting a subset of (the most relevant) source tasks for a given target task. Such solution would entail however a significant additional computational cost, since a new MTL model would have to be trained for each new target task.

There are also several interesting research directions regarding the architecture. On the one hand, it would be interesting to study the effect of increasing the capacity of the task-specific parts. On the other hand, we have only worked with classification tasks so far but it is possible to incorporate directly segmentation or detection tasks to the pre-training by appending an ad-hoc network as a head. Hopefully, enriching the training signal with a dense prediction task such as segmentation could improve the transferability of the resulting models. However, this approach also raises pratical questions and issues such as model memory usage or loss aggregation.

5.5 Conclusion

We have investigated the use of multi-task learning for pre-training neural networks in digital pathology. We have first created a pool of classification tasks from existing sources containing almost 900k digital pathology images. Using this pool, we have pre-trained a neural network in a multi-task setting in order to transfer the resulting model to unseen digital pathology tasks. Using a robust evaluation protocol, we have shown that transferring a model pre-trained in multi-task can be beneficial for the performance on the target task. When compared to transfer from ImageNet, our pre-training approach coupled with feature extraction yields comparable or better performance depending on the target dataset. We have observed that fine-tuning multi-task or ImageNet pre-trained models yields comparable performance. It suggests that fine-tuning is able to recover from the lack of feature specificity whatever the pre-training source. However, pre-training remains crucial, as models trained from scratch are clearly inferior.

Part III

Self-training

Chapter

6 Self-training for segmentation from sparsely-labeled data

Overview

In this chapter, we propose a self-training based approach that can exploit both (few) exhaustively annotated images and (very) sparsely-annotated images to improve the training of deep learning models for image segmentation tasks. The approach is evaluated on three public and a sparsely-labeled dataset annotated by collaborators on Cytomine, representing a diverse set of segmentation tasks in digital pathology. The experimental results show that self-training allows to bring significant model improvement by incorporating sparsely annotated images and proves to be a good strategy to relieve labeling effort in the digital pathology domain.

References: this chapter is an adapted version of the following article: Romain Mormont, Mehdi Testouri, Raphaël Marée, and Pierre Geurts. "Relieving pixel-wise labeling effort for pathology image segmentation with self-training". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. Accepted for publication. 2022 .

Supplementary materials for this chapter can be found in Appendix C.

6.1 Introduction

In this chapter, we investigate a method for training a binary image segmentation model in a context where the segmentation ground truth is sparse. More precisely, we focus on a setup where the training set is composed of an exhaustively-labeled subset $\mathcal{D}_l$ and a sparsely-labeled subset $\mathcal{D}_s$. In particular, the images in $\mathcal{D}_l$ come with exhaustively-labeled segmentation masks (*i.e.* pixels of all objects of interest have been labeled as positive) whereas in $\mathcal{D}_s$, some (unknown) pixels belonging to objects of interest have not been labeled, hence the sparsity. This setup can occur for instance when a dataset is first annotated with polygon annotations for classification then used for segmentation (see Section 6.3.1). Indeed, classification does not require exhaustive pixel-wise labeling, unlike segmentation. This setup can also describe a dataset in the process of being annotated: as annotations are added, some others are still missing. A method supporting our setup could actually be used in the context of AI-assisted interactive labeling.

Typically, image segmentation methods require that the training images come with exhaustive pixel-wise labels. In our setup, they would allow us to only use $\mathcal{D}_l$ and force us to ignore $\mathcal{D}_s$. We believe that it is possible to include the sparsely-labeled images as well, and hopefully improve the performance over using only $\mathcal{D}_l$. One way of achieving this would be to somewhat "complete" the sparse segmentation masks and make them exhaustively-labeled. Generating pseudo-labels is precisely what self-training approaches do and, in this chapter, we devise a self-training workflow to exploit both our sets.

Our self-training workflow consists of two separate phases. During the "*warm-up*" phase, we train a U-Net [167] model in a classical supervised manner on $\mathcal{D}_l$ for a few epochs. Then, during the "*self-training*" phase (sketched in Figure 6.1), we repeat the following process for an arbitrary number of epochs : pseudo labels are generated for the unlabeled pixels from images in $\mathcal{D}_s$ using the currently trained model and the pseudo-labeled images are included in the training set for the next epoch. To control the impact of model uncertainty on pseudo-labeled pixels, we furthermore study different weighting schemes to tune their contributions to the training loss. We use binary ("hard") pseudo-labels and propose an auto-calibration approach for generating them. Experiments are carried out on three exhaustively-labeled public datasets, namely MoNuSeg [109], GlaS [184] and SegPC-2021 [71], on which sparsity is artificially enforced for evaluation purpose. Through these experiments, we investigate the interest of our method and the impact of its hyperparameters in different scarcity conditions and in comparison with different baselines. In a second stage, we apply our method on an actual sparsely-labeled dataset for cytology diagnosis.

Our main contributions and conclusions are as follows: **(1)** We design a self-training pipeline for binary segmentation to handle datasets composed on both exhaustively and sparsely annotated images (Section 6.2). **(2)** We show on three public datasets that this self-training approach is beneficial as, even in significant scarcity conditions, it improves over using only exhaustively-labeled data (see Section 6.5.2). **(3)** We confirm the interest of the approach in a real-world scenario, where a significant improvement of performance is achieved by exploiting sparse annotations (see Section 6.5.5). **(4)** We show that, at fixed annotation budget, it is not necessarily better to focus the annotation effort on exhaustive labels rather than sparse labels (see Section 6.5.3).

Related self-training works can be found in Sections 2.6.6 and 3.4.6. Beyond self-training, we would also like to point out that there exist methods, implemented in software such as QuPath [15], Ilastik [21] or Cytomine [132] and based on traditional computer vision or machine learning, that allow users to interactively complete a partial hand-drawn annotation of a given image. Among these methods are, for instance, Graph Cut [111] and GrabCut [169], or more recently DEXTRE [127] and NuClick [103]. While the self-training approach explored in this chapter can certainly be exploited in an interactive mode, this question will be left as future work.

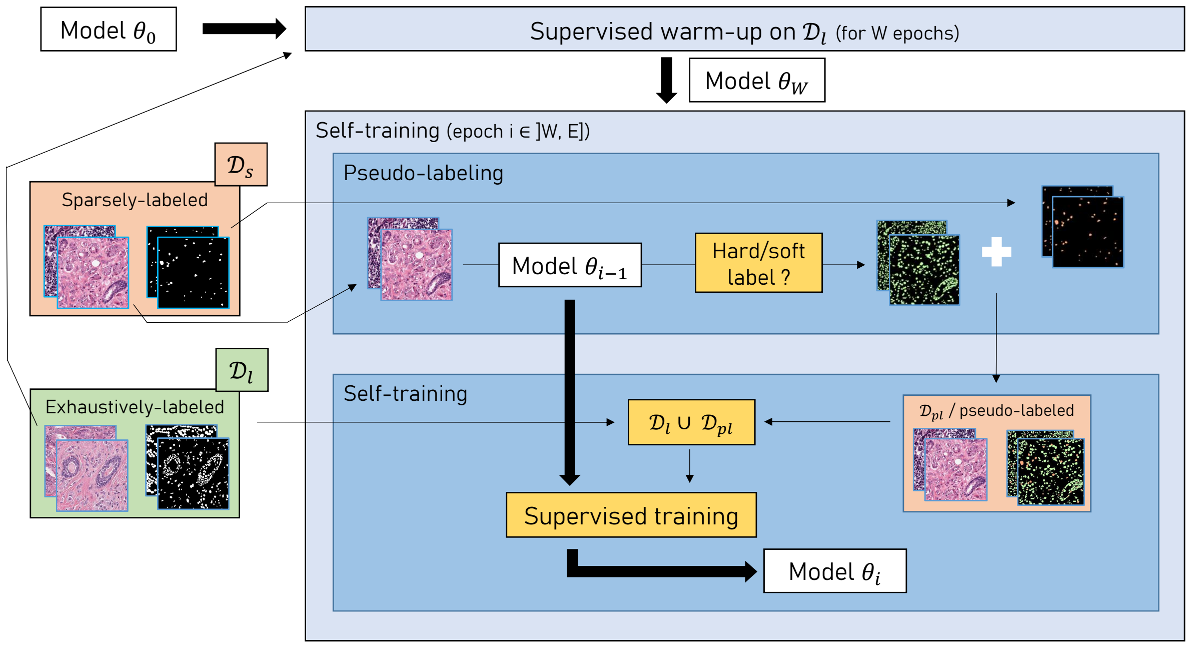


FIGURE 6.1: Our approach illustrated. The model is first warmed-up on the exhaustively-labeled data then further traind by self-training on the combined sparsely- and exhaustively-labeled sets. A more formal definition is provided in Algorithm 3.

6.2 Methods

In the following section, we present our method, a self-training image segmentation algorithm. The self-training aspects and training implementation details are discussed separately in Sections 6.2.1 and 6.2.2 respectively.

We will denote by $\mathcal{D} = (X, Y) \subset \mathcal{X} \times \mathcal{Y}$ a segmentation dataset, where X and Y respectively represent a set of input images and their corresponding binary segmentation masks. We will further consider a training dataset composed of two sets: $\mathcal{D}_l = (X_l, Y_l) \subset \mathcal{X} \times \mathcal{Y}$, the exhaustively-labeled set, and $\mathcal{D}_s = (X_s, Y_s) \subset \mathcal{X} \times \mathcal{Y}$, the sparsely-labeled set. In $\mathcal{D}_l$, the masks Y_l are entirely determined, since the ground truth is known for all pixels (hence the exhaustiveness). In $\mathcal{D}_s$, ground truth is only partially known: given an image $\mathbf{x} \in X_s$, either a pixel x_{ij} belongs to a structure of interest in which case the mask pixel $y_{ij} = 1$, or it is not labeled in which case $y_{ij} = 0$ and no assumption can be made a priori about the fact that the pixel belongs to a structure of interest or not. We will denote $n_l = |\mathcal{D}_l|$ and $n_s = |\mathcal{D}_s|$ the sizes of the sets. The total number of training images for a dataset will be denoted $n = n_l + n_s$.

6.2.1 Self-training

Our self-training algorithm is described in Algorithm 3 (and depicted in Figure 6.1). It features a warm-up phase during which the model is classically trained on the set

Algorithm 3: Our self-training approach. The `Train` operation trains the given model on the provided dataset according to the protocol explained in Section 6.2.2. The `Predict` operation produces segmentation masks for a set of input images using the model. The `Combine` operation combines ground truth masks and pseudo labels from the given sets as explained in Section 6.2.1.

```
Data: The exhaustively- and sparsely-labeled sets $\mathcal{D}_l$ and $\mathcal{D}_s$, a segmentation
      model $\theta_0$, $W$ and $E$ respectively the number of warm up epochs and the
      total number of epochs.
Result: A self-trained segmentation model $\theta_E$.
// Warm up
for $e \leftarrow 1$ to $W$ do
    $\theta_e$ = Train($\theta_{e-1}, \mathcal{D}_l$)
end
for $e \leftarrow W + 1$ to $E$ do
    // Pseudo labeling
    $\hat{Y}_s$ = Predict($\theta_{e-1}, X_s$)
    $Y_{pl}$ = Combine($\hat{Y}_s, Y_s$)
    $\mathcal{D}_{pl} = (X_s, Y_{pl})$
    // Self-training
    $\theta_e$ = Train($\theta_{e-1}, \mathcal{D}_l \cup \mathcal{D}_{pl}$)
end
return $\theta_E$
```

$\mathcal{D}_l$ (training implementation details are given in Section 6.2.2). The number of warm-up epochs $W > 0$ is fixed so that the model is able to converge on the labeled data. The warmed-up model is used as starting point for the self-training phase. Each self-training round e starts by pseudo-labeling $\mathcal{D}_s$ with the model θ_{e-1}. For an image $\mathbf{x} \in X_s$, the pseudo-label assigned to pixel x_{ij} is given by:

$$y_{ij}^{(pl)} = \begin{cases} 1, \text{ if } y_{ij} = 1 \\ g(\hat{y}_{ij}), \text{ otherwise} \end{cases} \tag{6.1}$$

where $\hat{y}_{ij}$ is the sigmoid output of model θ_{e-1} for pixel (i, j) given $\mathbf{x}$ as input and g is a function for generating the pseudo-label from $\hat{y}_{ij}$ (see below). In other words, we preserve the expert ground truth as pseudo-labels when available and use the model predictions for unlabeled pixels (this is the `Combine` step from Algorithm 3). With this strategy, entirely unlabeled images can also be included in $\mathcal{D}_s$. Our algorithm uses a single model (*i.e.* teacher = student) which is not reset between self-training rounds.

6.2.1.1 Soft and hard pseudo-labels

We considered two different pseudo-labeling strategies, or two different g functions (see Equation 6.1). Initially, we decided to simply take g to be the identity function $g(x) = x$ in which case the sigmoid output of the model was used as pseudo-label. This strategy is commonly called "*soft*" labeling. During the next self-training round, this soft pseudo-label will be compared to the network prediction which can be seen as a form of consistency constraint similar to those in [112, 205, 186]. However, early experiments have shown that this approach causes training instabilities. Therefore, we investigated a second strategy where the sigmoid output is binarized using a threshold $T_e \in [0, 1]$:

$$g(x) = \begin{cases} 1, \text{ if } x > T_e \\ 0, \text{ otherwise} \end{cases} \tag{6.2}$$

where e is a self-training round. We call this strategy "*hard*" labeling as pseudo-labels are either 0 or 1. In addition to ensuring some sort of consistency between the pseudo-labels and the predictions, as in the "*soft*" approach, this thresholding also encourages the model to produce confident predictions (closer to 0 or 1). Because we want to avoid T_e to be an additional hyperparameter to tune, we propose an auto-calibration strategy based on the Dice score:

$$Dice_T(\mathbf{y}, \hat{\mathbf{y}}) = \frac{2 \times \sum_{i,j} \left[\mathbb{1}_{\hat{y}_{ij} \geq T} \times y_{ij} \right]}{\sum_{i,j} \mathbb{1}_{\hat{y}_{ij} \geq T} + \sum_{i,j} y_{ij}} \tag{6.3}$$

where T is the threshold applied to the model output to generate a binary prediction. The auto-calibration procedure selects T_e such that the Dice score in (6.3) is maximized for the images from an exhaustively-labeled set $\mathcal{D}_a$:

$$T_e = \arg\max_T \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}_a} \text{Dice}_T \left(\mathbf{y}, h(\mathbf{x}; \theta_e) \right). \tag{6.4}$$

The ideal choice for $\mathcal{D}_a$ would be to use an external validation set but, in extreme data scarcity conditions, extracting such a validation set would penalize the performance of the algorithm by removing a significant amount of training data. Therefore, in this context, we consider using the training subset $\mathcal{D}_l$ as $\mathcal{D}_a$. This approach has the advantage of not requiring additional training data but also induces a risk of overfitting which might hurt generalization performance. We hope that the overfitting problem would be compensated by the improvement brought by hard pseudo-labeling. A performance comparison of these two pseudo-labeling strategies is given in Section 6.5.1 and motivates the use of the second strategy.

6.2.2 Training

In this section, we will provide more information about the `Train` procedure from Algorithm 3 which trains a model θ with a dataset $\mathcal{D}$. We use U-Net [167] as a segmentation architecture. We set the initial number of feature maps to 8 instead of 64 in the original article, with the rest of the network scaled accordingly. The main goal of this reduction of model capacity is to limit overfitting given the highly scarce data conditions explored in our experiments, but it would be worth exploring more complex architectures as future work.

The number of rounds W and E and the number of training iteration per round are chosen independently per dataset (see Appendix Section C.2). Every training iteration, we build a minibatch by sampling $B = 8$ images uniformly at random with replacement from $\mathcal{D}_l \cup \mathcal{D}_{pl}$ and by extracting one randomly located 512x512 patch and its corresponding mask from each of these images. The batch size was selected based on hardware memory constraints. We apply random data augmentation following best practices for machine learning in general and for self-training in particular [222, 186]. We apply horizontal and vertical flips, color perturbation in the HED space [206] (bias and coefficient factors up to 2.5%), Gaussian noise (standard deviation up to 10%) and Gaussian blur (standard deviation up to 5%).

As a training loss, we average the per-pixel binary cross-entropy ℓ over all pixels of the batch, as defined in:

$$\ell(\hat{y}; y) = y \log \hat{y} + (1 - y) \log(1 - \hat{y}), \tag{6.5}$$

$$\mathcal{L} = -\frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathbf{y}_b|} \sum_i \sum_j w_{ij,b} \ell(\hat{y}_{ij,b}; y_{ij,b}). \tag{6.6}$$

We multiply the per-pixel loss by a weight $w_{ij,b}$ for pixel (i, j) of the b^{th} image of the batch in order to tune the contribution of this pixel to the loss (see Section 6.2.2.1). We use Adam [100] as an optimizer with initial learning rate $\gamma = 0.001$ and default hyperparameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, no weight decay).

6.2.2.1 Weighting schemes

Different strategies are evaluated for generating the per-pixel weight $w_{ij,b}$ in Equation 6.6. For the sake of simplicity, we will drop the batch sample identifier b in the subsequent equations and denote this weight by w_{ij}. We introduced this weight term to have the possibility to tune the contribution of pseudo-labeled pixels when computing the loss. It is important to note that this weight only applies to pseudo-labeled pixels and, therefore, the ground truth pixels will always be attributed a weight of $w_{ij} = 1$. It is also important to note that the weight is inserted as a constant in our loss and the weight function is not differentiated during back-propagation.

We study five different weighting strategies each producing an intermediate weight w_{ij}^{ξ} where ξ is the strategy identifier. Because we want to avoid the amplitude of the loss and gradients to be impacted by the weighting strategy, we normalize it to obtain the final weight $w_{ij} = w^{\xi}_{ij} / \overline{w}^{\xi}$, where $\overline{w}^{\xi}$ is the average weight over all pixels of a patch. Our weighting strategies are as follows:

Constant. This strategy consists in setting $w^{\text{cst}}_{ij} = C$ where $C \in \mathbb{R}^+$ is an hyperparameter. Because $w_{ij} = 1$ for ground truth pixels, this allows to manually balance the relative contributions of ground truth and pseudo-labeled pixels. The special case $C = 1$ assigns the same weight to ground truth and pseudo-labels and therefore corresponds to removing $w_{ij,b}$ from Equation 6.6.

Balance This strategy automatically assigns a value to the C hyperparameter presented in the "*constant*" strategy. As a basis for this value, we use the ratio g of ground truth pixels in $\mathcal{D}_l \cup \mathcal{D}_s$. The final weight is given by:

$$w^{\text{bal}}_{ij} = \frac{g}{1 - g}. \tag{6.7}$$

This choice is motivated by the belief that our algorithm will provide more reliable pseudo-labels in a low data scarcity regime ($g \nearrow$) in which case it makes sense to tune up the contributions of those pseudo-labels to the loss. In the opposite situation of extreme data scarcity ($g \searrow$), we expect the algorithm to produce less reliable pseudo-labels.

Entropy Unlike the previous, this strategy is not concerned with balancing the contributions but instead penalizes pseudo-labels for which the model was uncertain. It considers the prediction $\hat{y}_{ij}$ as a probability and tune the contribution down using the Shannon entropy. The use of entropy is motivated by its use in several self-training methods [69, 116]. First, an intermediate weight ω_{ij} is computed as:

$$\omega_{ij} = 1 + \hat{y}_{ij} \log_2(\hat{y}_{ij}) + (1 - \hat{y}_{ij}) \log_2(1 - \hat{y}_{ij}). \tag{6.8}$$

Early experiments have shown that directly using ω_{ij} as a weight resulted in unstable training. Indeed, during early self-training rounds, the model typically produces $\hat{y}_{ij} \sim 0.5$ which results in $\omega_{ij} \sim 0$ for most pixels in a patch, leaving only foreground ground truth pixels to be evaluated in the loss. In order to avoid this behavior, we introduce a new hyperparameter $w_{min} \in]0, 1]$ which allows rescaling linearly the weights ω_{ij} to $w^{ent}_{ij} \in [w_{min}, 1]$ as defined in:

$$w^{ent}_{ij} = (1 - w_{\min})\, \omega_{ij} + w_{\min}. \tag{6.9}$$

Consistency. Self-training algorithms often enforce consistency between the teacher and student models predictions. Inspired from this, we exploit another form of consistency for this strategy. In structured output tasks like segmentation, there is a correlation between predictions that are spatially close, as, for most pixels, it is unlikely that the true label should differ between a pixel and its neighbors (except at the edges of objects of interest). Therefore, we use the pseudo-label consistency between a pixel and its neighbors as a proxy to evaluate reliability of this pseudo-label:

$$w_{ij}^{\text{cty}} = 1 - \frac{\sum_{k=-\eta}^{\eta} \sum_{l=-\eta}^{\eta} (\hat{y}_{ij} - \hat{y}_{(i+k)(j+l)})^2}{\eta^2 - 1} \tag{6.10}$$

where η is the size of the neighborhood and an hyperparameter of the method. We consider a square neighborhood around the central pixel and ignore pixels outside of the image at the image borders. We have arbitrarily chosen $\eta = 2$ as default value for all our experiments involving the "*consistency*" weighting strategy. This means that each pixel (except at the image borders) is compared to 24 neighboring pixels when computing consistency.

Merged. This strategy assigns a high weight to pixels for which the model is both certain and consistent (spatially). It achieves this by multiplying together the consistency weight w_{ij}^{cty} and the entropy raw weight ω_{ij}. Because ω_{ij} suffers from the issue described earlier, we apply the same re-scaling operation after multiplication:

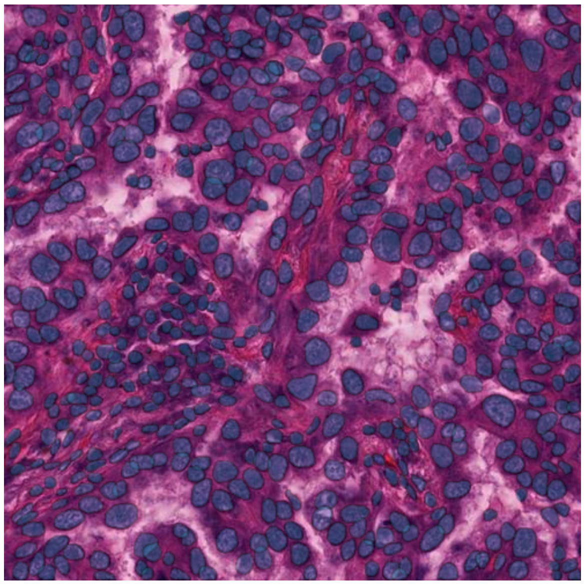
$$w_{ij}^{\text{mgd}} = (1 - w_{min}) \left(w_{ij}^{\text{cty}} \times \omega_{ij} \right) + w_{min}. \tag{6.11}$$

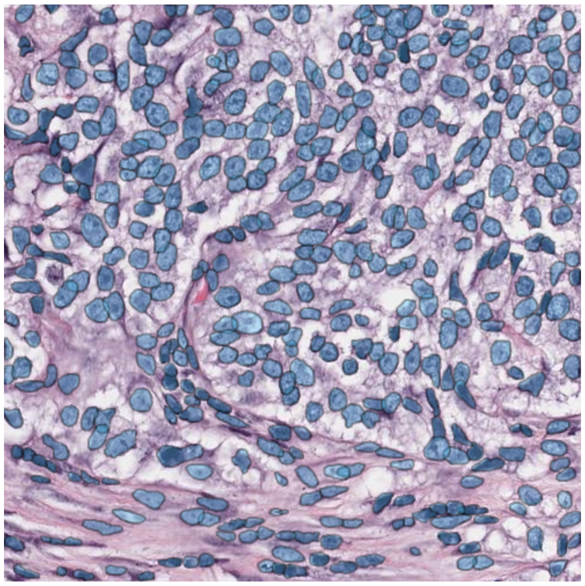
6.3 Data

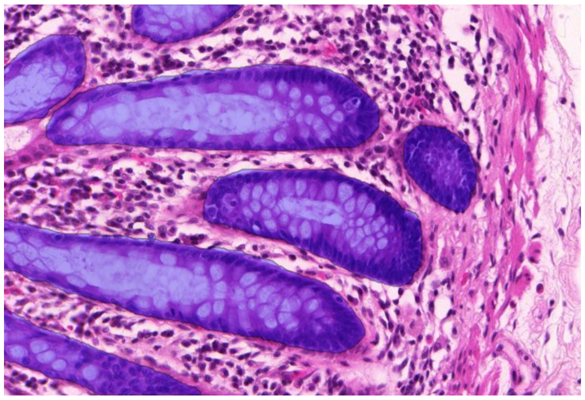
In this section, we describe the datasets we use to evaluate our method. It includes three public exhaustively-labeled segmentation datasets: MoNuSeg [109], GlaS [184] and SegPC-2021 [71]. The datasets are described in Table 6.1 and illustrated in Figure 6.2. MoNuSeg contains images of tissues coming from different organs, patients and hospitals where epithelial and stromal nuclei are annotated. Given the variety of sources, the images exhibit significant variations of staining and morphology. The density of annotations is also quite high compared to the other datasets. GlaS features images containing both benign tissues and malignant tissues with colonic carcinomas. The gland annotations vary greatly in shape and size. Originally, SegPC-2021 contains 3 classes: background, cytoplasm and nucleus. In our experiments, we merge the two latter classes as we focus on binary segmentation. One of the challenges of this dataset is the presence of non-plasma cells which should be ignored by the algorithm although they are very similar to plasma cells. Moreover, artefacts are present on the images (*e.g.* cracked scanner glass in foreground, scale reference or magnification written on the image).

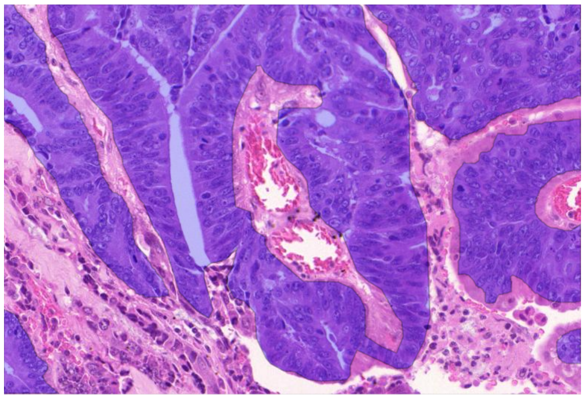
6.3.1 Thyroid FNAB

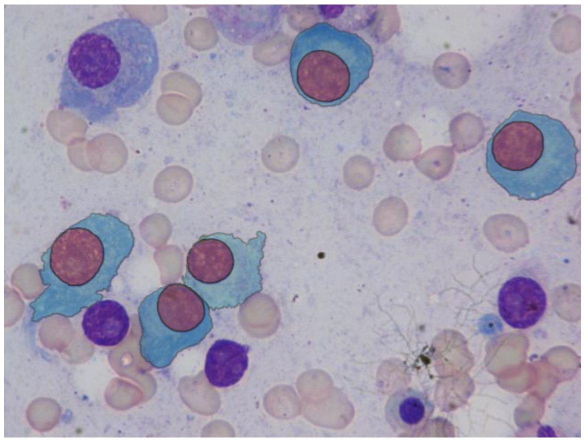
In addition to the three public datasets, we use a dataset that actually motivated the development of our method, a sparsely-labeled dataset for thyroid nodule malignancy


(A) MoNuSeg


(B) MoNuSeg


(C) GlaS


(D) GlaS


(E) SegPC-2021

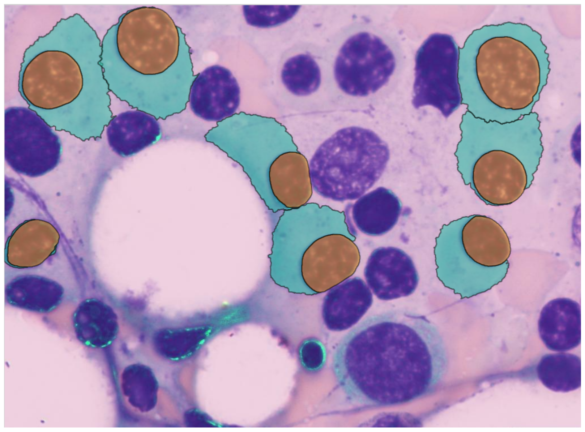

(F) SegPC-2021

FIGURE 6.2: Samples from MoNuSeg, GlaS and SegPC-2021 used in this chapter.

Dataset	Training set		Test set	
	Images	Annots	Images	Annots
MoNuSeg	30	17362	14	11484
GlaS	85	763	80	781
SegPC-2021	298	1643	300	900

TABLE 6.1: Summary statistics for the public datasets used in this chapter. An image is a region of interest extracted from a whole-slide image.

assessment. Pathologists[1] sparsely annotated nuclear features (see Figure 6.4) and architectural patterns (see Figure 6.3) with polygon annotations in 85 whole-slide images using Cytomine [132]. Each polygon was associated with a term from the ontology given in Appendix Section C.4. The training set consists of 4742 crops, one for each polygon annotation. The test set is a set of 45 regions of interest (2000x2000 pixels each) with annotations highlighting structures of interest (binary annotations, background *vs.* nuclei features and architectural cell patterns) made by a computer science student.

Given how the labeling process was carried out, we hypothesize that crops of architectural patterns are less likely to contain unlabeled cells than crops of nuclear features. Indeed, the former usually consist in large polygons delineating areas containing cell aggregates. Nuclear features, unlike architectural patterns, were usually labeled more sparsely and it is frequent to find annotations of a single cell within an unlabeled cell aggregate. From these observations, we have decided to assign the architectural patterns to $\mathcal{D}_l$ and nuclear features to $\mathcal{D}_s$.

6.4 Experimental setup

In this section, we present context information for our experiments: what are our baselines, how we have simulated sparse datasets from the public datasets and what evaluation protocol we have applied.

6.4.1 Transforming the datasets

In order to fit the sparsely-labeled settings described in Section 6.2, we generate new datasets from SegPC-2021, GlaS and MoNuSeg. This generation is controlled by two parameters: n_l and ρ. The former is the number of images to keep in the exhaustively-labeled set $\mathcal{D}_l$. These images are chosen at random without replacement in the original training set. The latter parameter ρ is the percentage of annotations to remove from the images to make the remaining images sparsely-labeled.

For MoNuSeg, we remove ρ% of the instances in each image selected for the sparse set. For SegPC-2021 and GlaS, because the number of instances per image is small (up to two or three dozen), we remove ρ% of the instances from the complete list of

[1]ULB Erasme hospital, Belgium, team of Pr. Isabelle Salmon.

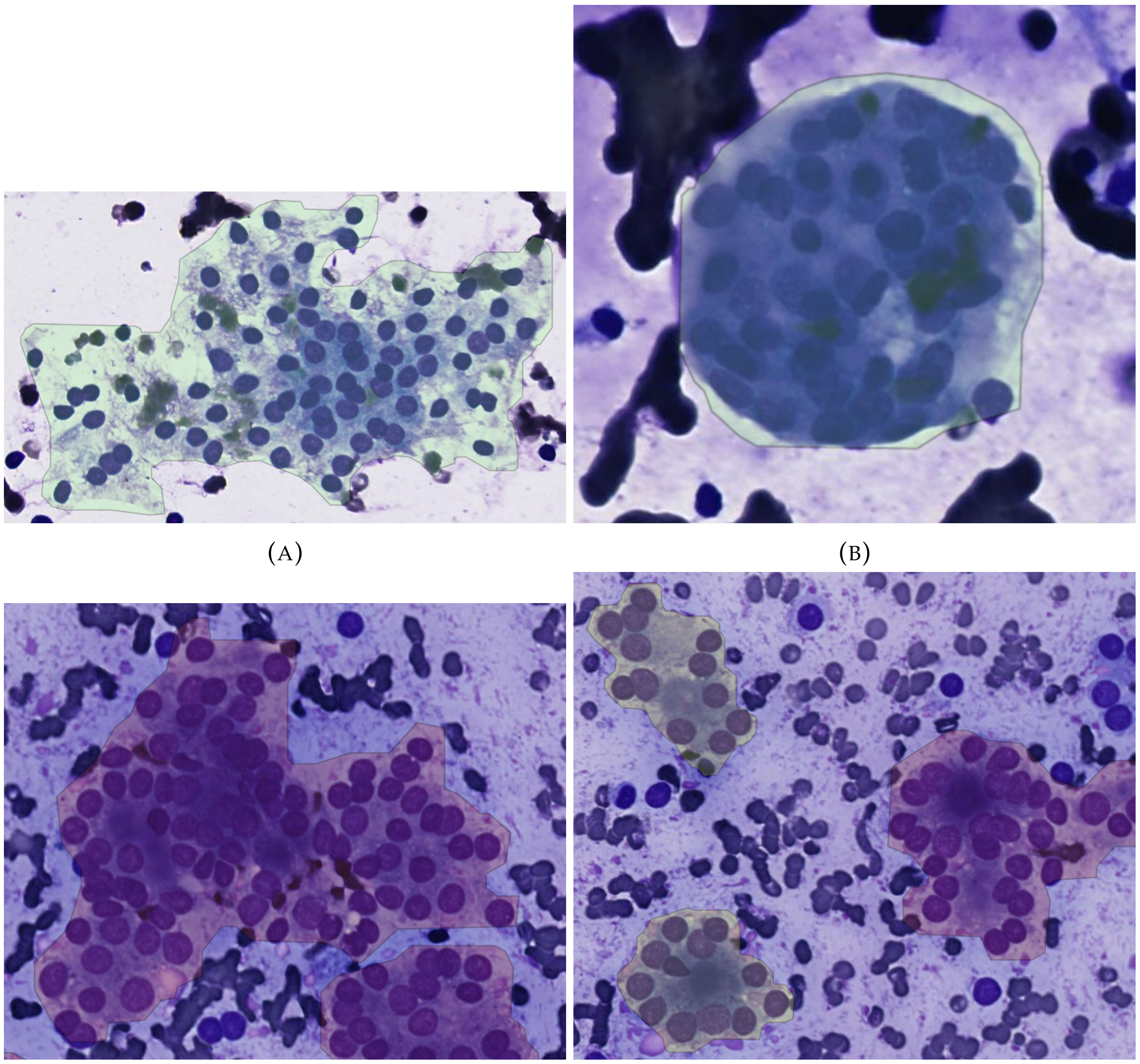

(A) (B) (C) (D)

FIGURE 6.3: Examples of architectural patterns annotations made by pathologists for Thyroid FNAB.

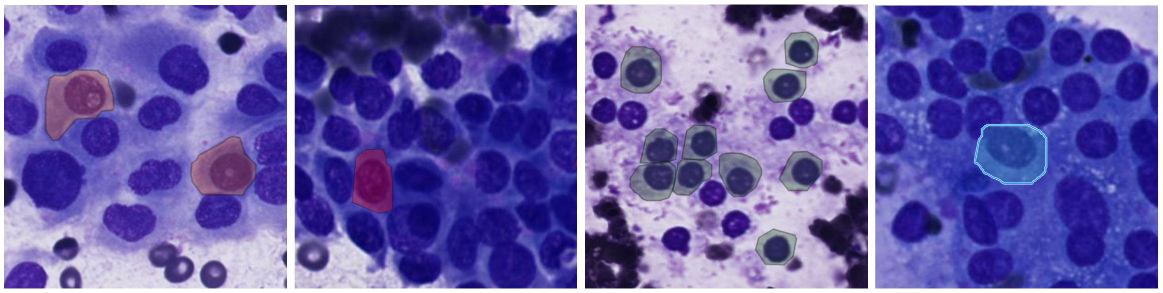

(A) (B) (C) (D)

FIGURE 6.4: Examples of nuclear features annotations made by pathologists for Thyroid FNAB.

instances. As a result, some sparse training images from these datasets can be completely void of ground truth. However, this is not a problem as our pseudo-labeling process is designed to support such images.

6.4.2 Baselines

We compare our self-training approach to three baselines, all exploiting a dataset in a fully supervised way. The first one, referred to as "*upper*", consists in using the full dataset, without removing any ground truth (*i.e.* $|\mathcal{D}_l| = n$ and $|\mathcal{D}_s| = 0$). Since it has access to all the annotations, this baseline is expected to represent an upper bound for all other strategies.

The second baseline consists in using the sparsely-annotated set $\mathcal{D}_s$ as if it was exhaustively annotated ($\mathcal{D}_{train} = \mathcal{D}_l \cup \mathcal{D}_s$). This strategy makes sense especially for moderately sparse datasets. Indeed, convolutional layers (as in U-Net) are able to cope with a bit of label noise given that gradients are averaged over feature maps to update the model parameters. Therefore, a bit of noise in certain parts of the images can be compensated by the feedback of ground truth labels in other locations. This baseline will be referred to as "$\mathcal{D}_l \cup \mathcal{D}_s$".

The third and last baseline, referred to as "$\mathcal{D}_l$ *only*", consists in not using at all the sparsely-annotated images during the training process (*i.e.* $\mathcal{D}_{train} = \mathcal{D}_l$).

Our self-training approach is of practical interest if it outperforms the two latter baselines. Moreover, the closer to the "*upper*" baseline the better.

6.4.3 Evaluation

We have built an evaluation protocol in order to ensure a fair comparison between the different strategies and the baselines. Ultimately, all approaches produce a segmentation model θ that will be the one evaluated. As an evaluation metric, we use the Dice score introduced in Equation 6.3 which requires a threshold T to turn the probability map produced by the model into a binary segmentation. In what follows, we will call T the *segmentation threshold*. To assess the performance of a model independently of the thresholding strategy, we will pick T so that the Dice score is maximized on the test set. In other words, in order to determine the segmentation threshold, we apply the optimization procedure described in Equation 6.4 where $\mathcal{D}_a$ is the test set $\mathcal{D}_{\text{test}}$. The Dice score resulting from this optimization will be referred to as "Dice*". It represents the optimal performance one could get, in terms of Dice score, on the test set with a given model. Dice* will be used in the next section to compare different algorithmic variants independently of the threshold tuning strategy. Obviously, in a context of extreme data scarcity, it will not be possible to tune the threshold this way because of the lack of a sufficiently large validation or test set. We will discuss and evaluate different ways to tune the segmentation threshold in Section 6.5.4.

Every experiment and hyperparameters combination we evaluate is run with ten random seeds to evaluate the variability. The seed affects the dataset sparsity (n_l and

ρ), model initialization, mini-batch sampling and data augmentation. We report Dice average and standard deviation over these random seeds.

6.5 Experiments and results

In this section, we present our experiments and results. In Section 6.5.1, we present our analysis of hard and soft pseudo-labels. In Section 6.5.2, we present our main self-training experiment in varying data scarcity conditions. In Section 6.5.3, we explore whether it is better to annotate a new segmentation dataset exhaustively or sparsely. We discuss the choice of a segmentation threshold in Section 6.5.4. Finally, we apply our method to the Thyroid FNAB dataset in Section 6.5.5.

6.5.1 Hard *vs.* soft pseudo-labeling

In this section, we explore how the two choices of pseudo-label generation presented in Section 6.2.1.1 impact the performance of self-training. In order to answer this question, we have run our self-training approach with different hyperparameter settings either with hard or soft pseudo-labels. We have performed 10 runs per hyperparameter combination and considered 24 combinations for SegPC-2021, 20 for MoNuSeg and 22 for GlaS (see Appendix Section C.1 for a detailed list of hyperparameters combinations). This experiment was carried out during the exploratory phase of the work where we studied a significantly scarce regime with $\rho = 90\%$ for all datasets and n_l set to 2 for MoNuSeg, 30 for SegPC-2021 and 8 for GlaS (*i.e.* between 90% and 95% of sparsely-labeled images with 90% of missing annotations).

The resulting performance are reported in Figure 6.5. We observe that, for all datasets except GlaS, the range of performance obtained with soft pseudo-labels fits within the range of performance of hard pseudo-labels. This indicates that soft pseudo-labels yield more stable performance and are less impacted by the choice of a weighting strategy and its hyperparameters, but also that when appropriately selecting the hyperparameters, hard pseudo-labels are able to produce better performance. However, it also means that a bad choice of self-training hyperparameters when using hard pseudo-labels would have a more significant negative impact on the performance. For GlaS, the performance obtained using soft pseudo-labels are inferior to those of hard pseudo-labels.

As a disclaimer, this analysis holds for the studied scarcity regime as the impact of hard and soft pseudo-labels could change in different conditions which we have not evaluated due to time and computational resources constraints.

6.5.2 Self-training performance at fixed n_l

In order to study how our self-training approach performs under different data scarcity conditions, we have generated several versions of our datasets by varying ρ with n_l fixed and have run the baselines and different hyperparameters combinations on the generated datasets. As discussed in Sections 6.2.1, we have used hard pseudo-labels

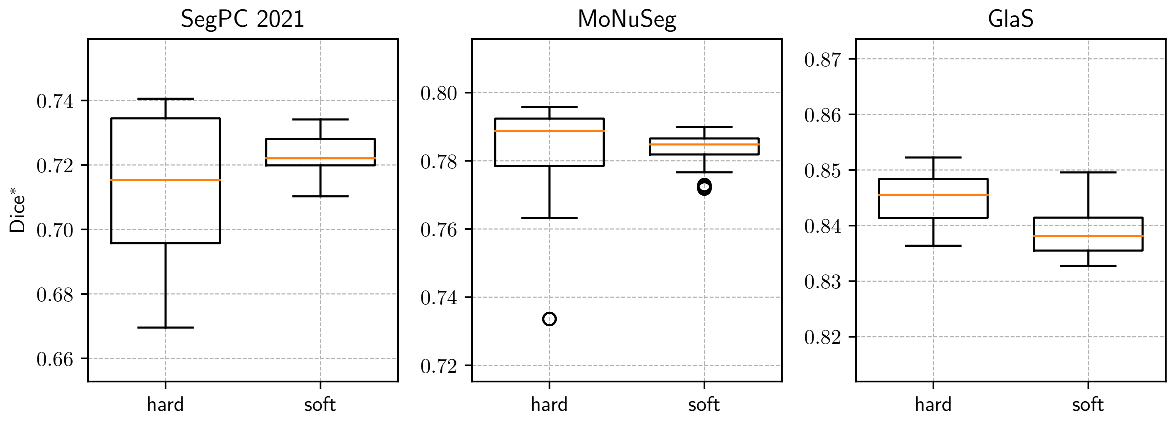


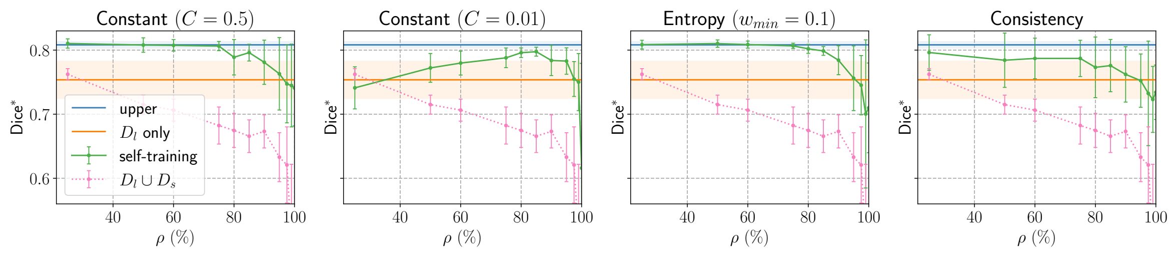
FIGURE 6.5: Comparing performance between using self-training with hard and soft pseudo-labels.

exclusively. The detailed hyperparameter combinations used in this section are provided in Appendix Section C.2.

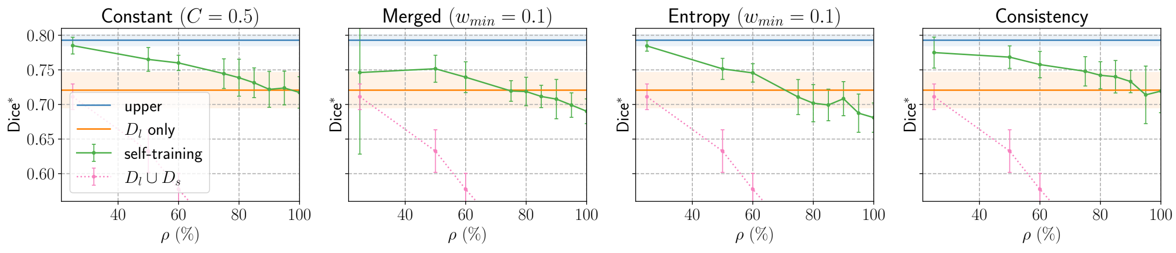
Results are shown for all three datasets in Figure 6.6. In general, self-training is always able to outperform significantly the "$\mathcal{D}_l$ *only*" and "$\mathcal{D}_l \cup \mathcal{D}_s$" baselines with a significantly reduced amount of sparse annotations (the exact value is dataset dependant, see below). Regarding the baselines, "*upper*" outperforms the two others. Moreover, using sparsely-labeled images as if they were exhaustively-labeled (*i.e.* $\mathcal{D}_l \cup \mathcal{D}_s$) appears not to be a good idea as it is outperformed by all self-training approaches and baselines in almost all scarcity conditions. The performance of this baseline increases as one adds more sparse annotations however and is able to catch up with the "$\mathcal{D}_l$ *only*" baseline in the lowest scarcity conditions validating the hypothesis presented in Section 6.4.2.

MoNuSeg. On this dataset, we can divide the analysis by differentiating three scarcity regimes: extreme ($\rho \in [95\%, 100\%]$), significant ($\rho \in [80\%, 90\%]$) and medium ($\rho \in [25\%, 75\%]$). Overall, most self-training approaches benefit from additional sparse annotations as their score increase when ρ decreases. This statement is true for all weighting strategies but the "*constant*" ($C = 0.1$) of which the performance plateau near $\rho = 85\%$, before decreasing as ρ decreases.

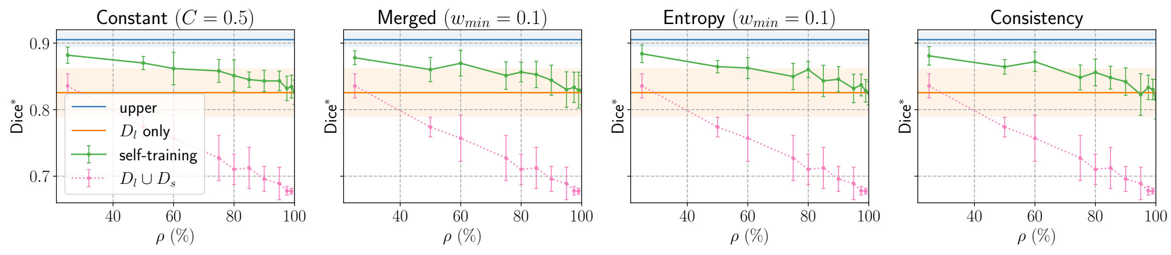
In the extreme regime, all self-training approaches exhibit high variance and are outperformed by the "$\mathcal{D}_l$ *only*" baseline, or yield comparable performance. In this situation, it appears to be better to work in a fully supervised fashion using only images from $\mathcal{D}_l$ rather then using our self-training approach. Indeed, it seems that the noise brought by the extreme annotation sparsity (or complete lack of annotation when $\rho = 100\%$) degrades the model significantly which cannot even make efficient use of the exhaustively-labeled images anymore. For $\rho = 95\%$, two self-training approaches ("*constant*" ($C = 0.1$) and "*entropy*") are on average better then the baseline



(A) MoNuSeg



(B) SegPC-2021



(C) GlaS

FIGURE 6.6: Performance of our baselines and self-training approaches with different hyperparameters combinations with a varying ρ and a fixed labeled set size n_l. We only report the weighting schemes that show representative behavior. Other weighting strategies are evaluated and reported in Appendix Section C.3.

but variance is still high making it difficult to really conclude that they are more efficient.

The situation is reversed in the significant regime where most self-training approaches (except the "*consistency*" weighting strategy) outperform the "$\mathcal{D}_l$ *only*" baseline and variance decreases significantly as well. As for the "*upper*" baseline, it remains more efficient than self-training. For $\rho = 90\%$, the most efficient weighting strategy on average is "*constant*" ($C = 0.1$) which also exhibits the smallest variance of all the self-training approaches. We believe that such a low constant is particularly helpful to combat the noise brought by the high sparsity as pseudo-labeled pixels contribute way less during training. For $\rho = 85\%$ and 90%, the "*constant*" ($C = 0.1$) strategy plateaus whereas others catch up in terms of performance and variance decrease with the "*entropy*" and "*merged*" (plot for this strategy can be found in Appendix Figure C.1) approaches taking up the lead.

In the medium regime, three self-training approaches reach, and even slightly surpass, the "*upper*" baseline: "*constant*" ($C = 0.5$), "*entropy*" and "*merged*". This result is interesting because it means that our self-training approach is able to reach the performance of a fully supervised approach but using only $\sim 30\%$ of the original annotations (*i.e.* $\rho = 75\%$, approximately 5k annotations instead of 17k) which is a significant annotation budget saving. The approach "*constant*" ($C = 0.1$) decreases with ρ indicating that such a low C prevents the model to learn efficiently from the additional annotations (compared to the significant regime). This strategy even finished below the $\mathcal{D}_l \cup \mathcal{D}_s$ baseline at $\rho = 25\%$.

Overall, results on MoNuSeg are quite satisfactory. Although our approach is struggling in an extreme scarcity regime, it quickly catches up with the "*upper*" baseline as one adds more annotations to $\mathcal{D}_s$. In this case, the choice of weighting strategy matters and depends on the sparsity of the dataset.

SegPC-2021. Regarding the trend, our self-training approach behaves similarly on SegPC-2021 (see Figure 6.6b) compared to MoNuSeg: all self-training approaches without exception seem to benefit from additional annotations in $\mathcal{D}_s$. Moreover, the $\mathcal{D}_l \cup \mathcal{D}_s$ baseline is particularly inefficient and finishes just below the "$\mathcal{D}_l$ *only*" baseline at $\rho = 25\%$. However, in the extreme regime, the gap between self-training and the "$\mathcal{D}_l$ *only*" baseline is less than on MoNuSeg. The rate at which our approach improves over the "$\mathcal{D}_l$ *only*" is also slower as it takes a larger ρ (around 75%) for the performance of self-training to become significantly better than this baseline. The best-performing weighting strategies also differ. The best strategies overall are "*constant*" (with $C = 0.5$ or 1) and "*consistency*". The "*merged*" and "*entropy*" are worse than the others, although the latter catches up at $\rho = 25\%$. On this dataset, only the "*constant*" and "*entropy*" strategies come close to catching up with the upper baseline but it takes proportionally more annotations compared to MoNuSeg as it happens around $\rho = 25\%$.

GlaS. On this dataset, all self-training approaches benefit from additional sparse annotations in $\mathcal{D}_s$. Compared to the "$\mathcal{D}_l$ *only*" baseline, the self-training approaches are never worse, even in the extreme scarcity regime, and it takes a ρ between 60% and 75% for self-training to become significantly better. Self-training is not able to catch up the "*upper*" baseline in this case.

6.5.3 Labeling a new dataset: sparsely or exhaustively?

The fact that self-training is able to equal or outperform the "$\mathcal{D}_l$ *only*" and "*upper*" baselines suggests that it might be more interesting to consider an alternative annotation strategy to exhaustive labeling when annotating a new dataset. At fixed annotation budget, it might indeed be more interesting to combine sparse labeling and self-training rather than performing fully supervised training on an exhaustively-labeled dataset (*i.e.* $\mathcal{D}_l$ only). To answer this question, we have conducted a set of experiments where we compare a self-training approach (entropy weighting strategy,

$w_{min} = 0.1$) and the baselines all run against different sparsity regimes, varying both $\rho \in \{90\%, 50\%, 25\%\}$ and n_l (values are dataset specific). The results of these experiments are given in Figure 6.7 where the values of n_l we have used are also specified. In these plots, the performance of all methods are reported over a common metric, the percentage of annotations used, which can be equated with the annotation budget for creating the dataset.

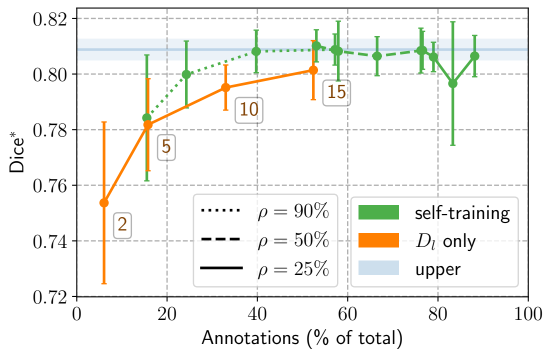
Our experiments show very dataset-dependent results. On MoNuSeg, we observe that self-training outperforms supervised training for all tested budgets. This indicates that it would have been more interesting to sparsely annotate this dataset. However, this conclusion does not hold for the other datasets as, within the same annotation budget, using "$\mathcal{D}_l$ *only*" outperforms self-training.

This experiment also allows to compare which labeling scheme is better for self-training: for a given annotation budget, is it better to favor a larger set $\mathcal{D}_l$ or to add more sparse annotations in $\mathcal{D}_s$? For MoNuSeg and SegPC-2021, it appears that, for a similar annotation budget, self-training performance are comparable whatever the values of n_l and ρ. Therefore, for those datasets, it does not really matter if the annotation budget is spent for exhaustive or sparse labeling. For GlaS, however, there is a performance loss when switching from a lower ρ value to a higher (*e.g.* going from $(\rho, n_l) = (90\%, 40)$ to $(50\%, 8)$ in Figure 6.7c). It indicates that, it is more interesting to label images exhaustively rather than sparsely for this dataset.

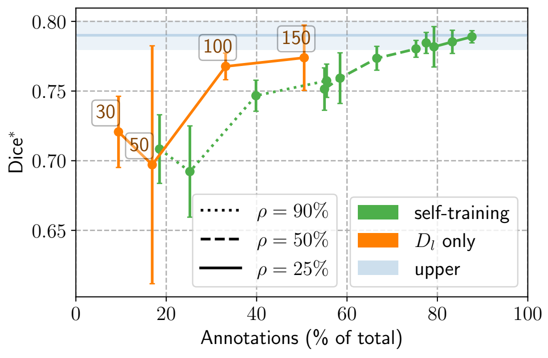
At this point, the experiments in this section do not allow us to provide definitive guidelines on how to focus annotation efforts to achieve optimal performance on a new dataset, but at least, they show that it can be beneficial to sparsely annotate more images than to exhaustively annotate fewer images.

6.5.4 Tuning the segmentation threshold

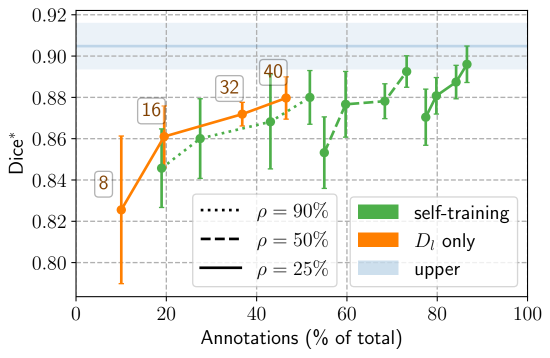
In previous experiments, we have compared algorithms in terms of Dice* scores, i.e, assuming that the segmentation threshold T^* has been optimally tuned on the test set. As discussed earlier, this tuning strategy is not realistic in a scarce data regime, because of the lack of a large validation set. In this section, we will assess the impact of two more realistic strategies to tune the segmentation threshold. The first one simply exploits the auto-calibration procedure used during self-training to determine the hard pseudo-labels, i.e., the segmentation threshold is tuned on the fully labeled training set $\mathcal{D}_l$. We will denote this threshold as T_E (as it corresponds to T_e as defined in Equation 6.4 at the last epoch E). This is likely to produce a biased threshold as the model has been specifically trained to produce good performance on $\mathcal{D}_l$. The second strategy we propose consists is tuning the segmentation threshold on only a few images from the test set. If the model has to be used to interactively assist the labeling of new images by a human user, it seems indeed realistic to require this user to calibrate the threshold by visual inspection of a couple of images. We will mimic this process by tuning the segmentation threshold on two images picked randomly from



(A) MoNuSeg, $n_l = \{2, 5, 10, 15\}$



(B) SegPC-2021, $n_l = \{30, 50, 100, 150\}$



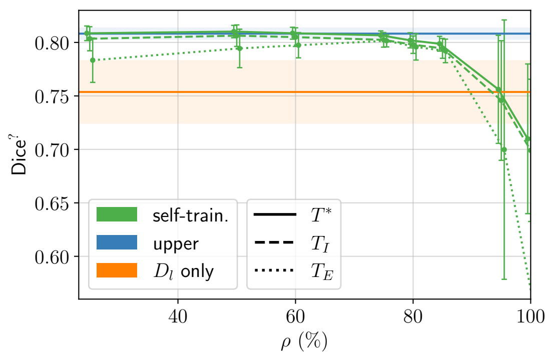
(C) GlaS, $n_l = \{8, 16, 32, 40\}$

FIGURE 6.7: Self-training *vs.* the baselines for varying ρ and n_l. A point corresponds to 10 runs of a given approach (see line color) and some sparsity parameters, ρ (see line style) and n_l (see subcaptions, increasing from left to right on a constant ρ curve, see "$\mathcal{D}_l$ *only*" for example). A new dataset is generated for each run, with the corresponding n_l and ρ values. The x value of a point corresponds to the average percentage of annotations used by the 10 datasets, or, alternatively, to the annotation budget dedicated for labeling the dataset.

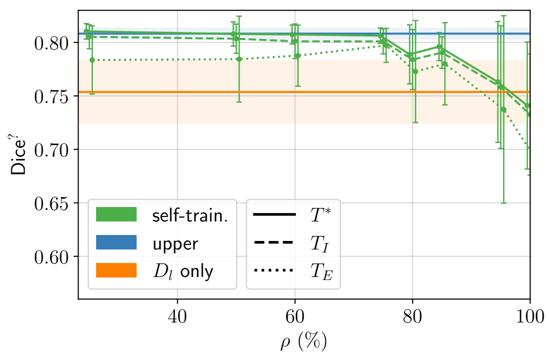
the test set. The threshold obtained this way will be denoted T_I below (for "Interactive"). This experiment is carried out on the same datasets as in Section 6.5.2 (same ρ and n_l values).

These two strategies are compared against the Dice* score with the models produced by the experiments in Section 6.5.2. Representative plots for each dataset can be found in Figure 6.8. As previously, results are averaged over 10 runs, with the two images used to obtained T_I randomized in each run.

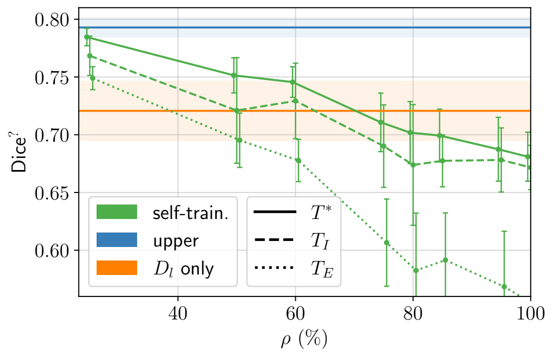
As expected, it appears that tuning the threshold T on the training set (*i.e.* using T_E) results in overfitting as we observe a performance drop compared to using T^* (tuned on $\mathcal{D}_{test}$). However, overfitting does not seem major for MoNuSeg and GlaS (in general, at most 4% Dice score difference compared to using T^* but often less) as opposed to SegPC-2021 where the Dice score difference in favor of T^* can reach more than 10% for some ρ values and does not go below 4%. Using T_I mostly reduces the performance gap with respect to T^* compared to using T_E. This is true for all datasets



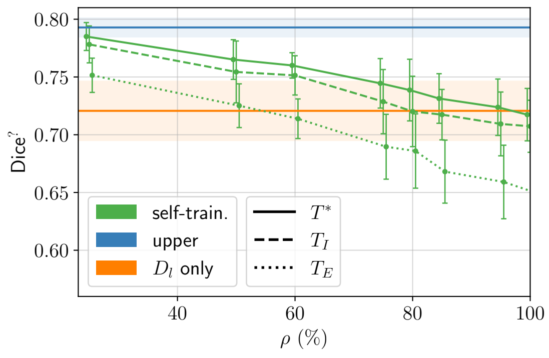
(A) MoNuSeg, entropy



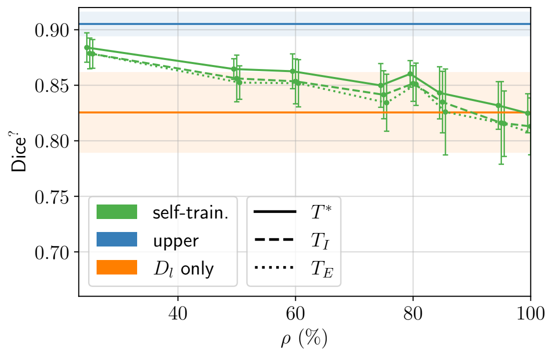
(B) MoNuSeg, constant ($C = 0.5$)



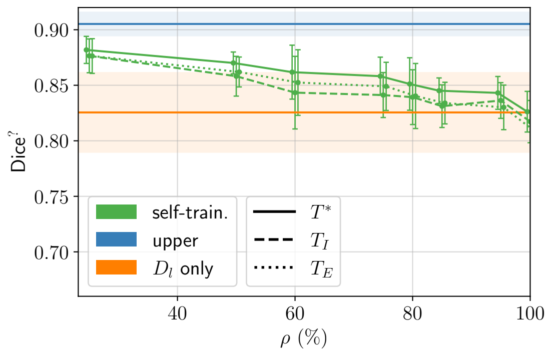
(C) SegPC-2021, entropy



(D) SegPC-2021, constant ($C = 0.5$)



(E) GlaS, entropy



(F) GlaS, constant ($C = 0.5$)

FIGURE 6.8: Performance resulting from using different threshold tuning techniques on our three datasets with two self-training approaches: "*entropy*" and "*constant*" ($C = 0.5$). The segmentation threshold for the "*upper*" and "$\mathcal{D}_l$ *only*" baselines is T^*.

Method	Dice*
Self-training	89.05 ± 0.85
$\mathcal{D}_l$ only	80.30 ± 5.39
$\mathcal{D}_l \cup \mathcal{D}_s$	83.62 ± 3.52

TABLE 6.2: Experiment on the Thyroid FNAB dataset. The self-training approach uses the "*entropy*" weighting strategy and $w_{min} = 0.1$.

and evaluated self-training approaches except for GlaS and "*constant*" ($C = 0.5$) where using T_E is better. The inversion of performance between T_E and T_I in this case occurs likely because overfitting is more significant with T_I than T_E as the number of images used for tuning T_I is limited compared to T_E (2 test images over 8 images from $\mathcal{D}_l$).

As a conclusion, depending on the dataset, one could already obtain good performance with T_E meaning that no test data is needed at all to tune the threshold. This is interesting for applications of our algorithm to interactive annotation for instance. Otherwise, using T_I is a strong alternative that requires only few test images. In practice, it is however not possible to determine the best approach a priori without a test set.

6.5.5 Experiments on Thyroid FNAB

The Thyroid FNAB dataset introduced in Section 6.3.1 offers a great opportunity to test our method on a real case of sparsely-labeled data. It is interesting to note that this dataset is larger than the three public datasets used in this study (almost 5k images in total).

Based on the results of Section 6.5.2, we have chosen to use the "*entropy*" weighting strategy with $w_{min} = 0.1$ for our self-training approach, as it provides consistently good results across datasets. We compare this approach with two of our three baselines: "*$\mathcal{D}_l$ only*" and "$\mathcal{D}_l \cup \mathcal{D}_s$". The "*upper*" baseline obviously cannot be evaluated because we do not have access to the complete ground truth. The resulting performances are given in Table 6.2.

We observe that our self-training approach significantly outperforms the two baselines and remains quite stable as its standard deviation is below 1%. This confirms the interest of self-training when working with a sparse dataset.

6.6 Conclusion and future works

In this chapter, we have introduced a method based on self-training for training a deep binary segmentation model with sparsely-labeled data. Using four datasets, we have shown that the method could indeed make use of sparse annotations to improve model performance over using only exhaustively-labeled data. For one of our datasets, our self-training approach using only 30% of the original training annotations is even able to reach performance comparable to using all of them in a supervised way. We have also shown that it can be beneficial to label a new dataset sparsely instead of exhaustively and confirmed the interest of our method on an actual sparsely-labeled dataset where self-training improved performance by a 5% margin compared to the baselines.

In the future, we want to further study the impact of various training choices and hyperparameters (model complexity, weighting strategies, soft pseudo-labeling, *etc.*) that we could not explore due to time and computing resources constraints. We also want to further study how the type of dataset (variability in images, density of ground truth, large or small annotations, *etc.*) impacts the performance margins of self-training. Moreover, we have removed annotation randomly from the datasets. In practice, it is unlikely that the existing annotations are really randomly chosen and it would be interesting to study the effect of the labeling process. To reinforce the generality of our conclusions, it would be interesting to include more segmentation datasets in a future study.

Currently, the method only supports binary segmentation but it is possibile to extend it to multiclass segmentation. This extension would start by changing the sigmoid output layer of UNet by a softmax function which implies that a label (and by extension a pseudo-label) is not a probability anymore but rather a probability distribution. Many aspects of our approach can be trivially adapted to this change (cross-entropy loss, soft labeling, entropy weighting strategy, *etc.*). There are two elements in particular that require a little more thoughts. The first is hard pseudo-labeling as thresholding is not relevant anymore and one could, for instance, take the most probable class. This criterion would however ignore the possible uncertainty of the model when entropy is high. A smarter strategy might be useful in this case (if the "*entropy*' 'weighting strategy is not enough). The second is the consistency weighting strategy as a new function to compare probability distributions must be chosen.

It has been shown for other applications that self-training is also useful when data is plentiful (*e.g.* [244]). So it would be interesting to study more extensively how our approach would perform in lower data scarcity condiditions (*e.g.* larger n and n_l).

7 Chapter
Conclusion and future works

7.1 Wrapping up

In this thesis, we investigated how image classification and segmentation machine learning techniques could be applied in the context of digital pathology. In particular, we studied how different methods could be used to alleviate the consequences of data scarcity, a prevalent issue in the domain. Causes and consequences of data scarcity are numerous and are discussed Chapter 3. Some of the consequences include small dataset size, lack of variety and oversimplification of the learning tasks, *etc*. These can have a significant negative impact on the efficiency of machine and especially deep learning methods which are known to be particularly data-hungry.

In Part II, we first investigated transfer learning. In Chapter 4, we compared different ways seven popular deep learning architectures pre-trained on ImageNet could be transferred to pathology classification tasks. We studied both feature extraction and fine-tuning. Regarding feature extraction, we considered features not only from the last layer of the networks but also from inner layers. Moreover, we devised an experiment to evaluate the redundancy of these features using recursive feature elimination. We showed that, although source (*i.e.* natural images) and target (*i.e.* pathology images) domains are quite dissimilar, both feature extraction and fine-tuning from ImageNet were effective transfer techniques as they were able to improve significantly over our baseline. Moreover, fine-tuning proved to be more efficient than feature extraction. Among the architectures we have tested, ResNet50 and DenseNet121 often yielded the best performance.

Moving on to Chapter 5, guided by the fact that transfer learning is known to perform better when source and target tasks are close, we searched for a way to create a transferrable model pre-trained on pathology data. However, because of data scarcity, it was (and still is to this date) not possible to find a sufficently large and versatile pathology dataset to pre-train a deep learning model. Therefore, we decided to exploit the fact that many small and medium pathology datasets had been released over the years and decided to pre-train our model in a multi-task fashion using as many of these datasets as possible. We collected 22 classification tasks featuring 81 classes and almost 900k images and designed a multi-task pre-training protocol to exploit them. We showed that our models used as feature extractors either outperformed or provided comparable performance compared to models pre-trained on ImageNet. We also observed that fine-tuning multi-task or ImageNet pre-trained models yielded comparable performance, indicating that fine-tuning was able to recover the lack of specificity of ImageNet features. We also confirmed the conclusions from Chapter 4 as we showed that fine-tuning outperformed feature extraction. Moreover, we also confirmed the consensus that transfer was more efficient than training from scratch.

In Part III and Chapter 6, we focused on semi-supervised learning and self-training in particular. We proposed a self-training algorithm for image segmentation able to make use of both exhaustively and sparsely-labeled data. Every training epoch, our approach generates pseudo-labels for unlabeled pixels in the training set using the currently trained model and combines them with the sparse ground truth. The model can then be trained on the combination of the pseudo-labeled and exhaustively-labeled set. Using three exhaustively-labeled datasets that we artificially made sparse, we showed that our self-training approach was able to improve performance over a fully supervised approach learning from the exhaustively-labeled data only. For one of our datasets, self-training was even able to reach performance of a model trained on a completely annotated dataset with only $\sim 30\%$ of the training annotations. We confirmed our findings on a sparsely-labeled dataset annotated by collaborators using Cytomine. On this dataset, we showed that self-training improved performance by a 5% margin compared to our baselines. We finally showed that when annotating a new dataset, it was not necessarily better to annotate it exhaustively.

We hope that, with these three contributions, we have confirmed that data scarcity in digital pathology was not an inextricable issue and that there exists algorithmic solutions to at least alleviate it.

7.2 Future works

In this section, we discuss future perspectives for our works.

Transfer with new architectures. In 2018, when we experimented with deep transfer learning, we investigated the transfer learning potential of many neural network architectures that were quite recent at that time. Since we published our first article (which is the subject of Chapter 4), many new architectures have been proposed and it would be interesting to evaluate how good these architectures would be in terms of transfer performance to pathology tasks. This includes new convolutional neural network architectures (*e.g.* NASNets [243], EfficientNets [202], EfficientNetsV2 [203], *etc.*) but more prominently transformer architectures like ViT which are currently state-of-the-art on ImageNet [229].

Consider more tasks and types of task. We have used 22 classification tasks for pre-training our models in multi-task. It would be interesting to add more tasks to the pool to reinforce even more the versatility of the pre-trained model. Since the publication or the related article in 2020, new pathology datasets have been released and would be good candidate to be integrated (*e.g.* CoNiC and NuCLS introduced in Table 3.1). Moreover, only supporting classification tasks is restrictive as it prevents from exploiting other kind of datasets like detection or segmentation (those that cannot be easily converted to classification at least). One way of including these types of task would be to use new head architectures. For instance, the classification head could be replaced by a decoder from a UNet model to support a segmentation task. This kind of change would require careful consideration because adding a large head for a bunch of tasks would increase the memory and time requirements for training the model. In the same spirit, it would be interesting to study the effects of using more complex heads for classification as well.

Study self-training with larger datasets. We have mostly studied how our method performed in high data scarcity conditions. However, self-training methods have shown to improve performance in context where data is plentiful (*e.g.* [244]). This can be done for instance by pseudo-labeling entirely unlabeled datasets. When working with whole-slide images, it is common to only have regions of interest annotated instead of entire slides. Therefore, this leaves out many unlabeled area that could be included as training data after pseudo-labeling. It would be interesting to study how including these would help improve performance. There exists pathology datasets that could directly be used for this purpose like, for instance, Camelyon [123] which features 209 whole-slide images with pixel-wise segmentation labels and 1190 unannotated slides. Another example is the Thyroid FNAB dataset used in Chapter 6. Implementing such an experiment raises interesting technical questions regarding processing time as it is probably not necessary nor efficient to predict pseudo-labels for billions of unlabeled pixels every training round as only a subset of them will likely be used for the next training round.

Self-training as an interactive annotation algorithm. As discussed in Chapter 6, one possible application of our self-training algorithm would be to use it as an interactive assistant for annotating new datasets. Because a dataset being annotated is sparse by nature, we believe that a self-training approach could be leveraged to train a model using the data being annotated. The model, while being trained, could also be used to generate segmentation masks to be refined and corrected by human annotators. The resulting corrected annotations could be incuded in the training set. Investigating this idea would require not only to re-implement the self-training in an interactive way but also raises practical questions about graphical user interface and user experience. The system should for instance minimize latency when a user requests annotations for a selected area meaning that the model should be able to receive queries while being trained. The training process should handle addition of new training data on-the-fly as provided by the annotator. The interface should provide an WSI viewer and efficient annotation and correction tools (*e.g.* Cytomine).

Systematic benchmarking of representation learning methods in pathology. Representation learning is interested in methods that train a model able to generate a rich representation for a data sample (*e.g.* in our case, feature vectors for pathology images). In Part II, we have investigated few representation learning approaches focused on transfer from ImageNet, fine-tuning and multi-task pre-training. These are only a few examples of representation learning as there exist many more ways of learning a representation which have been explored by the community. A particularly popular approach is self-supervised learning (see Section 2.2.2) which has seen a surge of interest by the computational pathology community in the past few years (*e.g.* [223, 217, 27, 104, 38]). Although each contribution individually provides a comparison against popular techniques, they are not easily comparable between each other because they do not use the same reference datasets as benchmarks. In the same spirit as Biaflows [172] (see Appendix E), it would be interesting to provide a way to systematically benchmark the performance of a representation learning methods on an open platform.

Mixing it all up. There is nothing that prevents the different methods explored in this thesis to be used together. Indeed, transfer learning can be used to initialize the encoder part of a Unet, multi-task learning can be used to train a segmentation model. More interestingly, we could use self-training and multi-task learning together to train a classification or a segmentation model. For instance, we could train a multi-task UNet on the datasets used in Chapter 6 (MoNuSeg, SegPC-2021 and GlaS and Thyroid FNAB) simultaneously. Every epoch we would pseudo-label unlabeled regions of the Thyroid whole-slide images that could be included as training data for the next training epoch.

Part IV

Appendices

A Appendix

Appendices for Chapter 4

A.1 Selected hyperparameters for the classifiers and the baseline

When we train a classifier on extracted features, we tune its hyperparameters by cross-validation. For SVM, we tune the penalty parameter C with values taken in $\{10^{-10}, 10^{-9}, ..., 10\}$. For extremely randomized trees, we grow fully expanded trees and tune the number of features evaluated at each split k among $\left\{1, \sqrt{n_f}, \frac{n_f}{2}, n_f\right\}$ where n_f is the total number of features. For the single layer perceptron, we tune the number of iterations among $\{1000, 2500, 5000, 10000\}$ and the learning rate among $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. All the other parameters values are the ones provided by default by the scikit-learn package [151]. More precisely, we use: the Adam [100] optimizer with default parameters, an L2 penalty on the weights with α set to 10^{-4} and a batch size of 200 samples.

For the baseline ET-FL, we tune the size of the windows (w_{min}, w_{max}) among all valid size ranges with `min_size` in $\{0.0, 0.25, 0.5, 0.75\}$ and `max_size` in $\{0.25, 0.5, 0.75, 1.0\}$ and the colorspace L among $\{$`TRGB`, `HSV`$\}$ where `TRGB` and `HSV` are respectively the normalized RGB and hue-saturation-value colorspaces. The number of extracted subwindow per image w is taken such that the total number of subwindows for the dataset w_t is approximately 1 million. The other fixed parameters are T, the number of trees, l_{min}, the minimum number of samples in a leave of a tree. The value for those parameters as well as the selected values for tuned parameters are given in Table A.1.

A.2 Best features

In order to investigate the best features (according to feature importances) for a given network, we compute the minimum size of the best features subset so that at least one feature is in this subset for all datasets. The resulting sizes for all networks are given in Table A.2. We also compute the percentages of overlap between subsets of selected features by RFE for all pairs of datasets (see Tablez A.8 and A.9)

Datasets	Fixed						Tuned (best)		
	T	w	w_t	l_{min}	k	C	w_{min}	w_{max}	L
C	20	551	1000616	1000	384	0.01	0.25	0.50	TRGB
G	20	69	1007745	1000	384	0.01	0.00	0.75	HSV
P	20	743	1000078	1000	384	0.01	0.25	0.50	HSV
N	20	1265	1000615	1000	384	0.01	0.00	0.25	HSV
B	20	55	1004355	1000	384	0.01	0.00	0.75	HSV
M	20	261	1001457	1000	384	0.01	0.25	0.75	TRGB
L	20	184	1001512	1000	384	0.01	0.25	0.50	HSV
H	20	228	1002516	1000	384	0.01	0.25	0.50	TRGB

TABLE A.1: Hyperparameters for ET-FL.

$\mathcal{N}$	**# feat.**	**% feat.**
Mobile	753	73.54
IncResV2	1277	83.14
IncV3	1537	75.05
ResNet	1803	88.04
VGG16	409	79.88
VGG19	392	76.56
DenseNet	1477	76.93

TABLE A.2: Given a network, this table gives the best features subset minimum size so that there is at least one feature that is in this subset for all datasets.

Layer l (name)	**Feat. maps dim.**		
	h_a	w_a	d
activation_1	112	112	64
activation_4	55	55	256
activation_7	55	55	256
activation_10	55	55	256
activation_13	28	28	512
activation_16	28	28	512
activation_19	28	28	512
activation_22	28	28	512
activation_25	14	14	1024
activation_28	14	14	1024
activation_31	14	14	1024
activation_34	14	14	1024
activation_37	14	14	1024
activation_40	14	14	1024
activation_43	7	7	2048
activation_46	7	7	2048
activation_49 (last)	7	7	2048
Total	/	/	**15168**

TABLE A.3: Name and dimensions of the layers extracted from inside ResNet for the "*Merging features across layers*" and "*Inner layers*" experiments.

A.3 Features and cut points information

This section provides more details about the inner layers we have considered in our experiments. The inner layers information for ResNet, IncResV2 and DenseNet can be respectively found in Tables A.3, A.4 and A.5.

A.4 Features selected with RFE

A summary of the number of selected features and the cross-validation curves for all datasets and networks are respectively given in Table A.6 and FigureS A.1 and A.2.

A.5 Detailed scores for transfer learning experiments

Detailed scores for all datasets, experiments and networks are given in Tables A.10 and A.11.

Layer l (name)	**Feat. maps dim.**		
	h_a	w_a	d
max_pooling2d_2	25	25	192
mixed_5b	25	25	320
block35_1_ac	25	25	320
block35_4_ac	25	25	320
block35_7_ac	25	25	320
block35_10_ac	25	25	320
mixed_6a	12	12	1088
block17_5_ac	12	12	1088
block17_10_ac	12	12	1088
block17_15_ac	12	12	1088
block17_20_ac	12	12	1088
mixed_7a	5	5	2080
block8_3_ac	5	5	2080
block8_6_ac	5	5	2080
block8_9_ac	5	5	2080
conv_7b_ac (last)	5	5	1536
Total	/	/	**17088**

TABLE A.4: Name and dimensions of the layers extracted from inside IncResV2 for the *"Merging features across layers"* and *"Inner layers"* experiments.

Layer l (name)	**Feat. maps dim.**		
	h_a	w_a	d
pool1	56	56	64
conv2_block6_concat	56	56	256
pool2_pool	28	28	128
conv3_block12_concat	28	28	512
pool3_pool	14	14	256
conv4_block48_concat	14	14	1792
pool4_pool	7	7	896
conv5_block32_concat	7	7	1920
bn (last)	7	7	1920
Total	/	/	**7744**

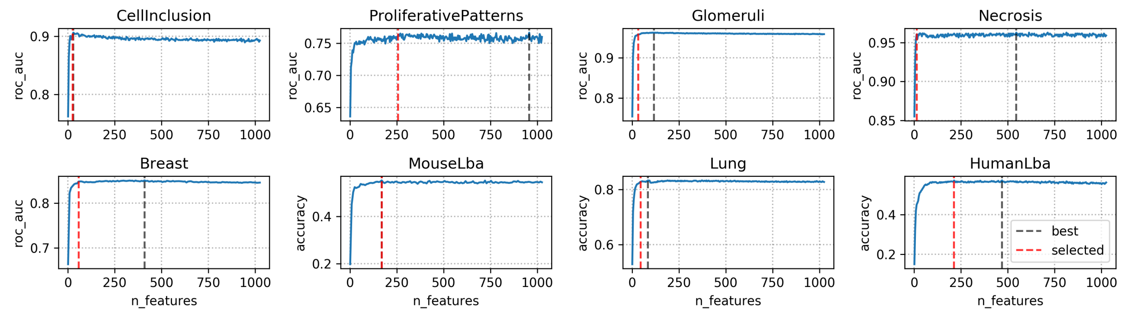
TABLE A.5: Name and dimensions of the layers extracted from inside DenseNet for the *"Merging features across layers"* and *"Inner layers"* experiments.

$\mathcal{N}$	**Number of features**							
	C	P	G	N	B	M	L	H
Mobile	25	257	33	13	57	169	45	213
DenseNet	25	37	17	13	49	241	57	245
IncResV2	29	53	29	13	185	249	69	173
ResNet	13	69	33	97	57	89	77	109
IncV3	33	13	37	13	97	121	105	137
VGG19	25	41	41	21	25	161	61	73
VGG16	25	29	33	29	49	225	93	113

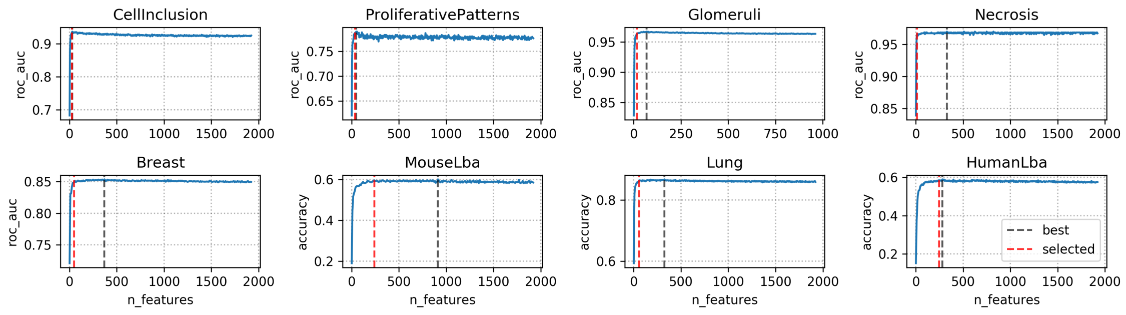
TABLE A.6: Number of features selected by RFE for all datasets and networks.

$\mathcal{N}$	**Proportion of features (%)**								Average
	C	P	G	N	B	M	L	H	
Mobile	2.44	25.10	3.22	1.27	5.57	16.50	4.39	20.80	9.91
DenseNet	1.30	1.93	0.89	0.68	2.55	12.55	2.97	12.76	4.45
IncResV2	1.89	3.45	1.89	0.85	12.04	16.21	4.49	11.26	6.51
ResNet	0.63	3.37	1.61	4.74	2.78	4.35	3.76	5.32	3.32
IncV3	1.61	0.63	1.81	0.63	4.74	5.91	5.13	6.69	3.39
VGG19	4.88	8.01	8.01	4.10	4.88	31.45	11.91	14.26	10.94
VGG16	4.88	5.66	6.45	5.66	9.57	43.95	18.16	22.07	14.55
Average	2.52	6.88	3.41	2.56	6.02	18.70	7.26	13.31	**7.58**

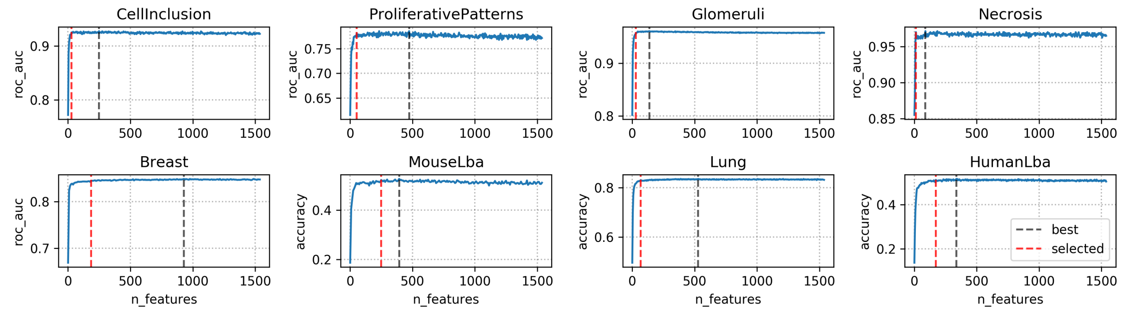
TABLE A.7: Proportion of features selected by RFE for all datasets and networks.



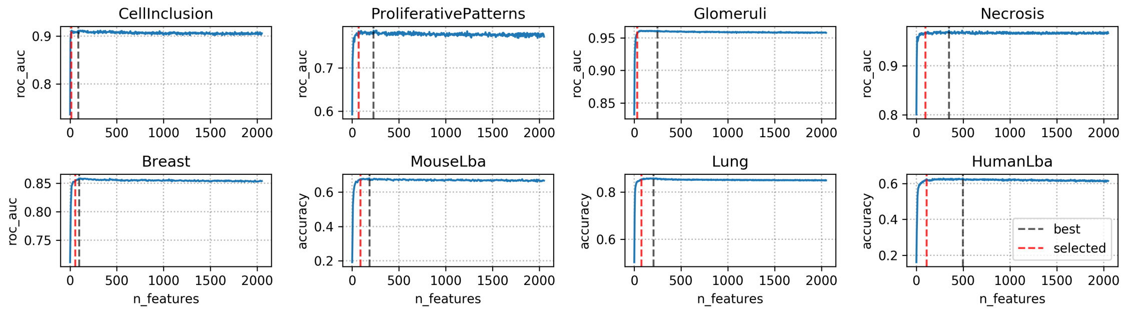
(A) Mobile



(B) DenseNet

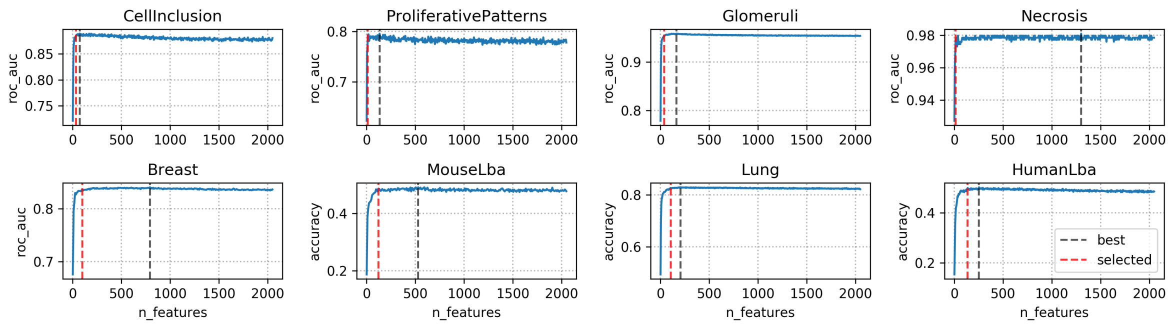


(C) IncResV2

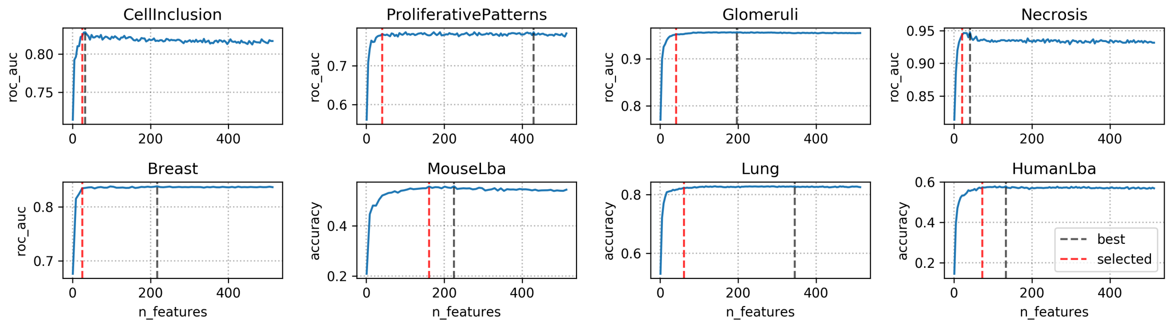


(D) ResNet

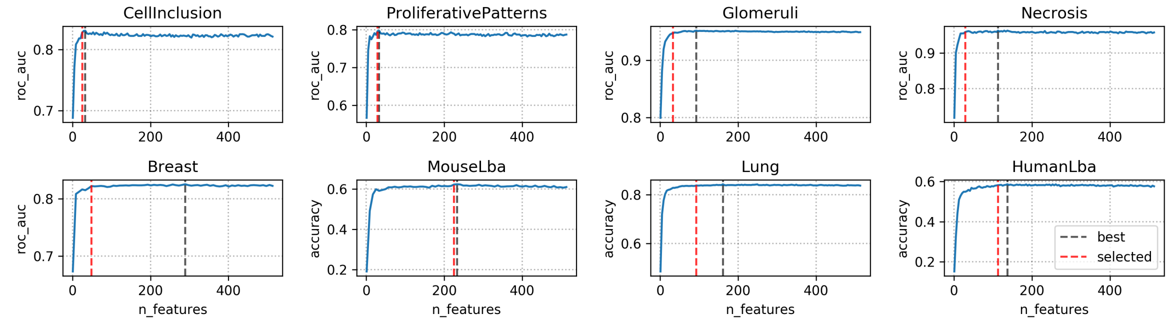
FIGURE A.1: RFE curves for last layers features from InceptionV3, VGG19 and VGG16.



(A) IncV3



(B) VGG19



(C) VGG16

FIGURE A.2: Cross validation curves from recursive feature elimination for last layers features from InceptionV3, VGG19 and VGG16.

Dataset	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
C		1	3	0	0	3	4	2
P	20		27	53	40	33	33	34
G	4	3		7	14	4	15	6
N	0	2	3		5	1	4	0
B	0	8	24	23		5	17	8
M	24	22	21	15	17		26	31
L	8	5	21	15	14	7		6
H	20	28	39	15	33	40	31	

(A) Mobile

Dataset	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
C		1	0	0	3	2	7	3
P	3		3	0	6	4	2	5
G	0	1		0	1	2	0	1
N	0	0	0		0	0	0	0
B	20	22	10	7		14	20	13
M	17	22	17	15	18		20	21
L	17	3	0	0	7	5		7
H	20	17	10	0	12	14	18	

(B) IncResV2

Dataset	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
C		0	2	0	4	2	3	3
P	0		0	7	0	1	2	1
G	3	0		0	8	1	6	0
N	0	7	0		2	2	1	1
B	12	0	21	15		10	13	10
M	9	15	5	23	13		21	20
L	12	23	18	15	14	19		10
H	15	15	2	15	15	23	14	

(C) IncV3

Dataset	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
C		2	0	2	3	3	0	2
P	15		0	7	7	7	14	11
G	0	0		2	5	2	5	1
N	15	10	6		19	11	11	9
B	15	5	9	11		6	16	11
M	23	10	6	10	10		10	23
L	0	15	12	9	22	9		10
H	23	18	6	10	21	29	14	

(D) ResNet

TABLE A.8: Percentages of overlap between features selected by RFE on the studied datasets (part 1, see Figure A.9 for part 2). The tables can be read as follows: the number at row *i* and column *j* is the percentage of features among the ones selected for dataset *j* that were also selected for the dataset *i*.

Dataset	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
C		10	3	10	8	6	5	7
P	12		9	13	14	9	14	9
G	4	10		13	12	8	17	6
N	12	13	12		16	7	8	11
B	16	24	18	27		16	18	19
M	60	72	60	58	73		62	78
L	20	44	48	27	34	25		26
H	32	37	21	44	44	39	32	

(A) VGG16

Dataset	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
C		4	2	4	12	6	8	8
P	8		14	19	12	11	14	16
G	4	14		0	24	10	21	6
N	4	9	0		20	6	9	8
B	12	7	14	23		7	13	9
M	44	46	41	52	48		50	45
L	20	22	31	28	32	19		16
H	24	29	12	28	28	20	19	

(B) VGG19

Dataset	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
C		2	0	0	0	1	3	2
P	4		11	0	2	3	3	3
G	0	5		0	6	0	1	0
N	0	0	0		0	1	0	0
B	0	2	17	0		3	5	2
M	16	21	11	23	16		21	24
L	8	5	5	0	6	5		7
H	28	21	11	15	12	24	31	

(C) DenseNet

TABLE A.9: Percentages of overlap between features selected by RFE on the studied datasets (part 2, see Figure A.8 for part 1).

Experiment	$\mathcal{C}$	$\mathcal{N}$	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
Baseline	ET-FL		0.9250	0.8268	0.9551	0.9805	0.9345	0.7568	0.8547	0.6960
Last layer	SVM	Mobile	0.9749	0.8844	0.9935	0.9953	0.9427	0.7611	0.9043	0.6882
		DenseNet	0.9794	0.8852	0.9938	0.9864	0.9257	0.7010	0.9133	0.7820
		IncResV2	0.9795	0.8698	0.9928	0.9982	0.9485	0.6566	0.9077	0.7351
		ResNet	0.9748	0.8893	0.9924	0.9882	0.9372	0.7633	0.9122	0.7791
		IncV3	0.9722	0.8670	0.9910	0.9964	0.8951	0.6371	0.9088	0.7175
		VGG19	0.8853	0.8654	0.9860	0.9905	0.9241	0.7237	0.8885	0.7302
		VGG16	0.8824	0.8808	0.9859	0.9893	0.9413	0.7438	0.9020	0.7028
Last layer	ET	Mobile	0.9608	0.8848	0.9854	0.9840	0.9487	0.5872	0.8648	0.7126
		DenseNet	0.9726	0.8889	0.9891	0.9870	0.9556	0.6381	0.8874	0.7410
		IncResV2	0.9618	0.8699	0.9824	0.9953	0.9408	0.4789	0.8570	0.6676
		ResNet	0.9634	0.8832	0.9758	0.9929	0.9507	0.6186	0.8851	0.7752
		IncV3	0.9481	0.8795	0.9793	0.9929	0.9428	0.4583	0.8300	0.6305
		VGG19	0.8551	0.8430	0.9795	0.9861	0.9231	0.5379	0.8536	0.6989
		VGG16	0.8412	0.8791	0.9690	0.9888	0.9254	0.5791	0.8659	0.6833
Last layer	FC	Mobile	0.9796	0.8661	0.9794	0.9935	0.9603	0.7941	0.8986	0.6823
		DenseNet	0.9822	0.8668	0.8316	0.9852	0.9482	0.7291	0.9054	0.7664
		IncResV2	0.9756	0.8676	0.9729	0.9976	0.9597	0.6598	0.9043	0.7038
		ResNet	0.9726	0.8670	0.9771	0.9899	0.9583	0.7996	0.9133	0.7674
		IncV3	0.9714	0.8417	0.9796	0.9893	0.9377	0.6538	0.8998	0.7038
		VGG19	0.8447	0.8553	0.9661	0.9899	0.9237	0.6636	0.8863	0.7410
		VGG16	0.8298	0.8718	0.9573	0.9852	0.9421	0.6956	0.9088	0.7185
Feature selection	SVM	Mobile	0.9610	0.7876	0.9794	0.9870	0.9597	0.7421	0.8682	0.6618
		DenseNet	0.9347	0.8212	0.8316	0.9888	0.9436	0.6614	0.7984	0.6931
		IncResV2	0.9665	0.8476	0.9729	0.9976	0.9443	0.6403	0.8682	0.7214
		ResNet	0.9578	0.8337	0.9771	0.9722	0.9492	0.7438	0.8806	0.7644
		IncV3	0.9562	0.8308	0.9796	0.9964	0.9436	0.6430	0.8750	0.6843
		VGG19	0.8284	0.8488	0.9661	0.9888	0.8860	0.6750	0.8784	0.7038
		VGG16	0.8071	0.8810	0.9573	0.9899	0.9131	0.7362	0.8941	0.7038
Feature selection	ET	Mobile	0.9617	0.8798	0.9799	0.9888	0.9581	0.6582	0.8694	0.7400
		DenseNet	0.9676	0.8861	0.9843	0.9994	0.9489	0.6939	0.8919	0.6667
		IncResV2	0.9609	0.8646	0.9743	0.9944	0.9421	0.5330	0.8491	0.6500
		ResNet	0.9503	0.8786	0.9799	0.9852	0.9488	0.6961	0.8863	0.7038
		IncV3	0.9473	0.8410	0.9786	0.9959	0.9466	0.5531	0.8378	0.7703
		VGG19	0.8506	0.8492	0.9732	0.9947	0.9186	0.5850	0.8468	0.7019
		VGG16	0.8232	0.8774	0.9659	0.9953	0.9282	0.6349	0.8615	0.6745
Merging networks	ET	merged	0.9897	0.8573	0.9948	0.9858	0.8851	0.8169	0.9155	0.7928
	SVM	merged	0.9784	0.8984	0.9912	0.9864	0.9549	0.6896	0.8615	0.6063
Merging layers	SVM	DenseNet	0.9757	0.8090	0.9835	0.9870	0.9470	0.7042	0.8840	0.7761
		IncResV2	0.9808	0.8418	0.9920	0.9964	0.9559	0.7031	0.9155	0.7761
		ResNet	0.9789	0.8576	0.9927	0.9953	0.9234	0.7941	0.9268	0.7977
Merging layers	ET	DenseNet	0.9605	0.8892	0.9875	0.9911	0.9588	0.6993	0.8818	0.7370
		IncResV2	0.9799	0.8906	0.9944	0.9920	0.9639	0.6495	0.8897	0.7370
		ResNet	0.9424	0.8787	0.9847	0.9929	0.9619	0.7080	0.8885	0.7683
Fine-tuning	SVM	DenseNet	0.9883	0.8556	0.9944	0.9870	0.9777	0.8342	0.9119	0.8553
		IncResV2	0.9841	0.8377	0.9909	0.9941	0.9403	0.6847	0.9039	0.7390
		ResNet	0.9921	0.8705	0.9897	0.9941	0.9637	0.8147	0.9119	0.8456
Fine-tuning	ET	DenseNet	0.9828	0.8965	0.9950	0.9876	0.9827	0.7887	0.8982	0.8094
		IncResV2	0.9769	0.8776	0.9850	0.9929	0.9477	0.5406	0.8446	0.7048
		ResNet	0.9909	0.8806	0.9879	0.9870	0.9772	0.7763	0.8845	0.8289
Fine-tuning	Net	DenseNet	0.9892	0.8797	0.9977	0.9893	0.9835	0.8483	0.9405	0.8641
		IncResV2	0.9851	0.8795	0.9971	0.9929	0.9873	0.8727	0.9165	0.8182
		ResNet	0.9926	0.8778	0.9953	0.9970	0.9827	0.8288	0.8971	0.8416
Metric			ROC AUC					Accuracy (multi-class)		

TABLE A.10: Detailed scores for all datasets and for the *"Last layer"*, *"Feature selection"*, *"Merging features across networks"*, *"Merging features across layers"* and *"Fine-tuning"* experiments.

$\mathcal{N}$	**Layer** l	**C**	**P**	**G**	**N**	**B**	**M**	**L**	**H**
Baseline / ET-FL		0.9250	0.8268	0.9551	0.9805	0.9345	0.7568	0.8547	0.6960
ResNet	activation_1	0.7720	0.8415	0.9275	0.9811	0.9100	0.4946	0.8153	0.5758
	activation_4	0.8283	0.8275	0.9723	0.9964	0.9390	0.7308	0.8806	0.6285
	activation_7	0.8456	0.8276	0.9772	0.9888	0.9351	0.7275	0.8930	0.6755
	activation_10	0.8574	0.8292	0.9759	0.9882	0.9439	0.6717	0.8930	0.6491
	activation_13	0.8859	0.8608	0.9824	0.9888	0.9483	0.7313	0.9077	0.6188
	activation_16	0.8975	0.8418	0.9860	0.9876	0.9478	0.7356	0.9054	0.6598
	activation_19	0.8877	0.8503	0.9892	0.9888	0.9499	0.7270	0.9077	0.6510
	activation_22	0.9244	0.8763	0.9892	0.9882	0.9555	0.7010	0.9223	0.6940
	activation_25	0.9506	0.8785	0.9933	0.9858	0.9639	0.7736	0.9223	0.7253
	activation_28	0.9489	0.8884	0.9935	0.9876	0.9638	0.8131	0.9245	0.7634
	activation_31	0.9519	0.8724	0.9938	0.9876	0.9659	0.7996	0.9201	0.7351
	activation_34	0.9584	0.8947	0.9940	0.9876	0.9606	0.7514	0.9223	0.7977
	activation_37	0.9671	0.8959	0.9942	0.9876	0.9663	0.7600	0.9280	0.7996
	activation_40	0.9621	0.8894	0.9949	0.9864	0.9664	0.7914	0.9155	0.8113
	activation_43	0.9710	0.8950	0.9942	0.9852	0.9648	0.8017	0.9223	0.8074
	activation_46	0.9712	0.8848	0.9937	0.9870	0.9652	0.7860	0.9291	0.8094
	activation_49 (last)	0.9748	0.8893	0.9924	0.9882	0.9640	0.7860	0.9122	0.7791
DenseNet	pool1	0.7187	0.8276	0.8994	0.9533	0.9227	0.4821	0.7826	0.4653
	conv2_block6_concat	0.7982	0.8374	0.9609	0.9905	0.9374	0.6300	0.8536	0.5259
	pool2_pool	0.8185	0.8296	0.9570	0.9893	0.9510	0.6235	0.8570	0.5337
	conv3_block12_concat	0.9024	0.8361	0.9861	0.9882	0.9522	0.6696	0.9020	0.6823
	pool3_pool	0.9309	0.8900	0.9832	0.9893	0.9382	0.6300	0.9088	0.6686
	conv4_block48_concat	0.9803	0.8876	0.9962	0.9870	0.9699	0.8012	0.9223	0.7674
	pool4_pool	0.9843	0.8984	0.9954	0.9870	0.9613	0.7703	0.9268	0.7859
	conv5_block32_concat	0.9862	0.8981	0.9955	0.9917	0.9623	0.7806	0.9201	0.7879
	bn (last)	0.9784	0.8867	0.9931	0.9852	0.9538	0.7573	0.9043	0.7967
IncResV2	max_pooling2d_2	0.8403	0.8091	0.9716	0.9941	0.9340	0.6143	0.8851	0.6158
	mixed_5b	0.8265	0.8146	0.9771	0.9905	0.9424	0.6945	0.8897	0.6461
	block35_1_ac	0.8325	0.8412	0.9776	0.9941	0.9412	0.6576	0.8897	0.6373
	block35_4_ac	0.8673	0.8770	0.9834	0.9923	0.9556	0.6495	0.8998	0.6716
	block35_7_ac	0.8981	0.8709	0.9844	0.9935	0.9590	0.6354	0.9043	0.7048
	block35_10_ac	0.9219	0.8692	0.9900	0.9935	0.9616	0.6549	0.9110	0.7253
	mixed_6a	0.9445	0.8747	0.9920	0.9953	0.9706	0.7172	0.9088	0.7439
	block17_5_ac	0.9681	0.8665	0.9945	0.9917	0.9695	0.8066	0.9190	0.7713
	block17_10_ac	0.9711	0.8687	0.9958	0.9935	0.9720	0.8137	0.9234	0.7674
	block17_15_ac	0.9762	0.8939	0.9960	0.9923	0.9622	0.7985	0.9144	0.7419
	block17_20_ac	0.9860	0.8948	0.9957	0.9923	0.9649	0.7741	0.9155	0.7693
	block8_3_ac	0.9873	0.8905	0.9959	0.9953	0.9676	0.7790	0.9190	0.7273
	block8_6_ac	0.9868	0.8871	0.9953	0.9923	0.9686	0.7562	0.9212	0.7468
	block8_9_ac	0.9824	0.8773	0.9946	0.9964	0.9632	0.7427	0.9144	0.7468
	mixed_7a	0.9868	0.8934	0.9962	0.9959	0.9602	0.7893	0.9178	0.7214
	conv_7b_ac (last)	0.9773	0.8615	0.9926	0.9982	0.9619	0.6766	0.8998	0.7361
Metric		ROC AUC					Accuracy (multi-class)		

TABLE A.11: Detailed scores for all datasets and for the *"Inner layer"* experiment.

B

Appendix

Appendices for Chapter 5

B.1 Data preparation

The general goal of the pre-processing is to obtain images which dimensions are compatible with state-of-the-art neural networks (*e.g.* ResNet [77], DenseNet [81]) and that are properly labeled. Classification (CLF) datasets already had classes associated to each image. For these datasets, we have thus kept the original classes. When datasets had large input images, we have further splitted them into smaller patches (see Table B.1).

We have considered the detection (DET) datasets to be the ones that contained several objects per input image where each object was usually denoted by a point annotation, and sometimes a label (*e.g.* Warwick CRC [185]). For those datasets, the transformations were more involved (see Table B.2). When the concentration of annotations was high (*i.e.* a typical patch in the image contains tens of annotations) and/or the input image size was small (*i.e.* $<$ 1k pixels square), overlapping patches were extracted. Each patch was associated a binary class indicating whether the entity to detect was present or absent in this patch. When a label was available, the associated class was chosen to indicate the presence of one type of object versus the other(s). For datasets where the objects to detect were fewer and more scattered over the input images (*e.g.* mitosis detection), the previous approach was inappropriate as it would have yielded highly imbalanced datasets. Therefore, in this case, negative patches were still sampled exhaustively with an overlap but positive patches were sampled around the objects of interest with random shifts, yielding several samples per object.

For the segmentation (SEG) datasets (see Table B.3), the patch sampling was the same as for the detection (*i.e.* exhaustive with overlap). The class was determined if the surface ratio of the positive entity (*e.g.* tumor) in the patch exceeded a threshold (*e.g.* 10% of the patch). The only exception is *Breast1* dataset for which the class of the patch is the class of its central pixel. Camelyon16 [19] dataset was applied an additionnal pre-processing to exclude most of the whole-slide image (WSI) background.

The last transformation step was to split each resulting dataset into some training, validation and test sets for future training. We have followed a rigorous splitting process: whenever possible we have made sure that images from a same patient, or a same slide were not in two different sets. Sometimes, none of those information were available in which case we have randomly split the data. Moreover, we have ensured that all classes were present in all sets.

Dataset	Patches
BACH1018 Micro	512×512
Stroma LBP	original
UMCM Colorectal	original
Janowczyk	original
Janowczyk	384×384

TABLE B.1: Details for classification datasets transforms. Patches indicate whether the final patches are the *original* images. Provided dimensions indicate that patches of those dimensions were extracted from the original images to make the final classification datasets.

Dataset	Positive	Negative	Other	Dim. / Sup.
Warwick CRC	{Inflammatory}	{Epithelial, Fibroblast, Others, ∅}	/	100×100 / none
TUPAC2016 Mitosis	{Mitosis}	{∅}	/	250×250 / 10
MITOS-ATYPIA	{Mitosis}	{∅}	{NonMitosis}	323×323 / 10
Janowczyk 5	{Mitosis}	{∅}	/	250×250 / 10

TABLE B.2: Details for detection datasets transforms. Columns *Positive* and *Negative* indicate which annotation information or label was used to set respectively the patch class as positive or negative. The ∅ means "*no annotation*". *Dim* and *Sup* stand for *Dimensions* and *Supersample*. The former indicates whether or not the positive patches was supersampled, and if so, how many patches were extracted per positive annotation.

Resulting classification datasets and their splits are listed in Table B.4. Selected samples for each of our final classification tasks are given in Figure 5.1.

B.2 Transfer performances

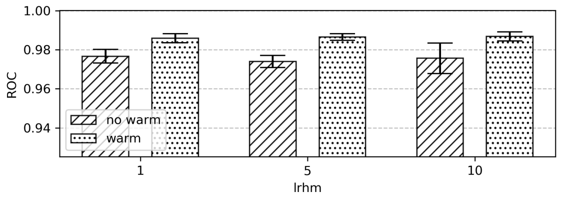
Figures B.1 and B.2 give the transfer performance of different combination of training hyperparameters.

Dataset	Classes			Dimensions			WSI	P/CW
	Positive	Negative	Area (%)	Extracted	Rescaled	Overlap		
Janowczyk 1	{Nuclei}	{∅}	5	250 × 250	/	125	no	/
Janowczyk 2	{Epithelium}	{∅}	10	200 × 200	/	100	no	/
Camelyon 16	{Tumor}	{∅}	10	768 × 768	384 × 384	0	yes	1000
Breast1	{InSitu, Infiltration}	{∅}	/	384 × 384	/	/	no	/
Breast2	{InSitu, Infiltration}	{∅}	10	250 × 250	/	125	no	/

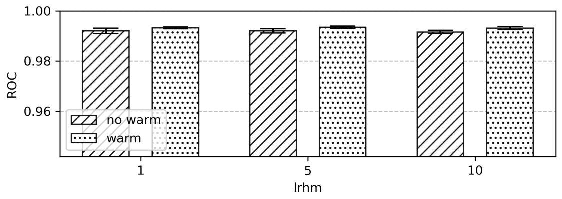
TABLE B.3: Details for segmentation datasets transforms. *Area* is the surface threshold we have used to separate positive from negative patches (if surface of positive annotation was larger than the given value, then the patch was considered positive). Column "*WSI*" indicates that the original images are whole-slide images and were applied additional preprocessing to remove background tiles. Column "*P/CW*" indicates whether or not the patches were subsampled. If a value is provided, this value is the maximum number of samples per class per WSI capped that was produced.

Name	**Cls**	**Train**		**Val**		**Test**		**Total**		**Split**
		Img.	**p/s**	**Img.**	**p/s**	**Img.**	**p/s**	**Img.**	**p/s**	
Necrosis	2	695	9	96	1	91	3	882	13	slide
ProliferativePattern	2	1179	19	167	4	511	13	1857	36	slide
CellInclusion	2	1643	21	173	2	1821	22	3637	45	slide
MouseLba	8	1722	9	716	4	1846	7	4284	20	slide
HumanLba	9	4051	50	346	5	1023	9	5420	64	slide
Lung	10	4881	669	562	73	888	139	6331	881	slide
Glomeruli	2	12157	10	2448	8	14608	102	29213	120	slide
Breast1	2	14055	22	4206	8	4771	4	23032	34	patient
Breast2	2	11483	22	3470	8	2570	4	17523	34	patient
BoneMarrow	8	522	522	130	130	639	639	1291	1291	slide
Janowczyk 1	2	17550	77	4500	19	9675	41	31725	137	patient
Janowczyk 2	2	1701	21	405	5	1296	16	3402	42	patient
Janowczyk 5	2	16560	7	4551	2	3759	3	24870	12	patient
Janowczyk 6	2	224822	230	31934	29	20768	20	277524	279	patient
Janowczyk 7	3	1350	225	456	76	438	73	2244	374	patient
MITOS-ATYPIA	3	40364	13	12799	4	11710	5	64873	22	slide
Warwick CRC	2	1500	60	500	20	500	20	2500	100	image
Camelyon 16	2	237753	221	27950	26	26523	24	292226	271	slide
TUPAC2016 Mitosis	2	62874	526	7827	74	7152	56	77853	656	patient
Stroma LBP	2	947	492	407	228	959	656	2313	1376	image
BACH2018 Micro	4	2760	143	720	52	1320	89	4800	284	patient
UMCM Colorectal	8	3349	6	/	/	1651	4	5000	10	patient
Total	81	663918	3374	104363	778	114519	1949	882800	6101	/

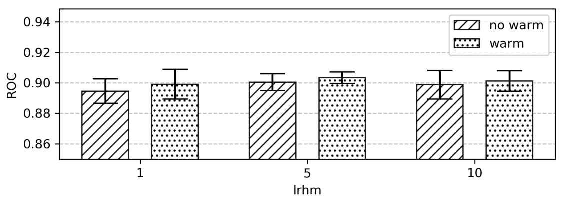
TABLE B.4: Classification datasets generated from the collected datasets. *p/s* indicate the number of distinct patients, or slides (if no patient information was available), or images (in case when none of the two information were available) in the set. The column *Split* indicates whether the dataset was split patient, slide or image-wise.



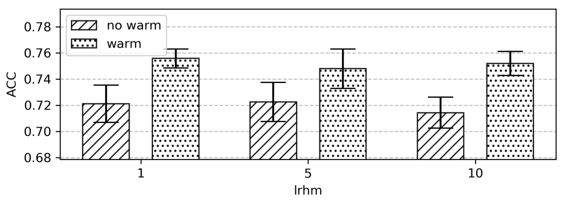
(A) CellInclusion



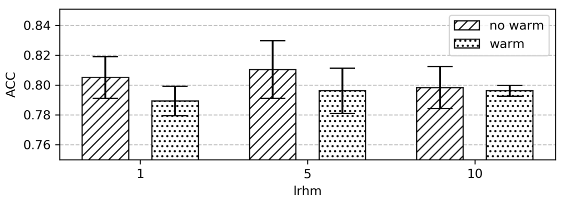
(B) Glomeruli



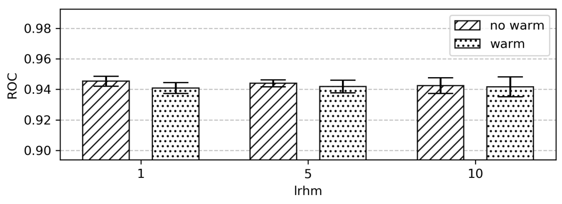
(C) ProliferativePattern



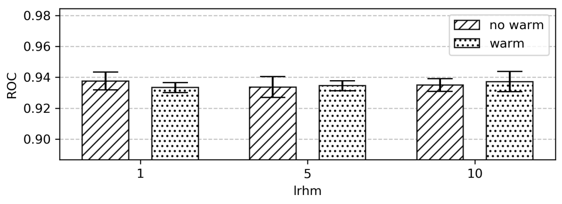
(D) HumanLba



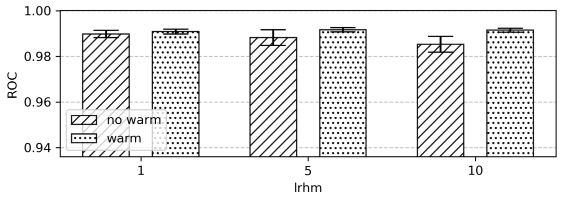
(E) BoneMarrow



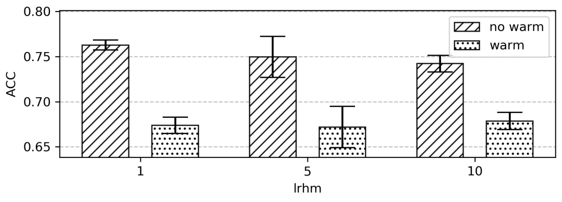
(F) Breast1



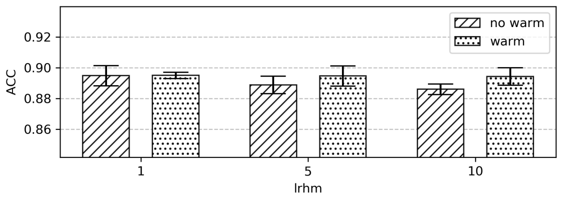
(G) Breast2



(H) Necrose

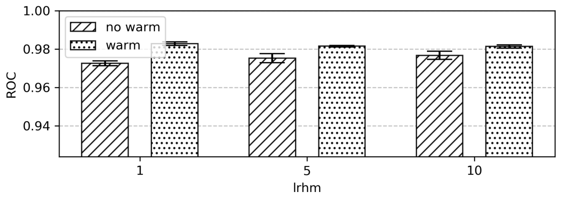


(I) MouseLba

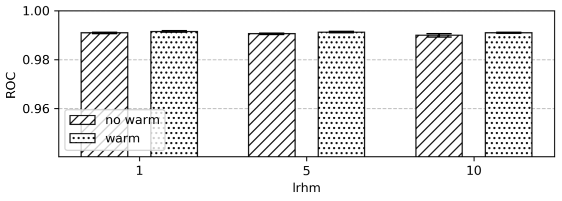


(J) Lung

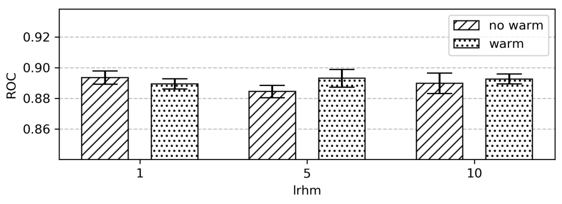
FIGURE B.1: Transfer performance for combinations of the hyperparameters γ_τ (learning rate heads multiplier) and w (warm up) with learning rate $\gamma = 10^{-4}$ on DenseNet121.



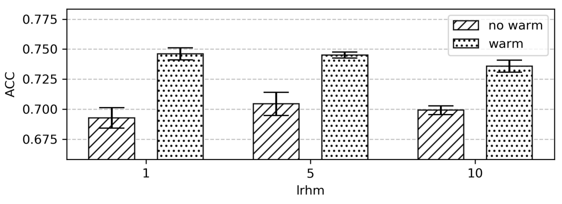
(A) CellInclusion



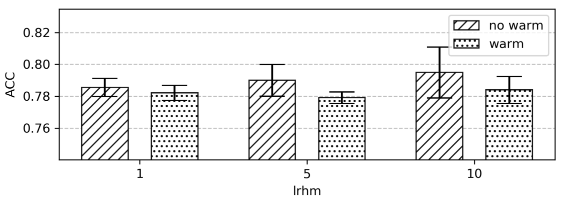
(B) Glomeruli



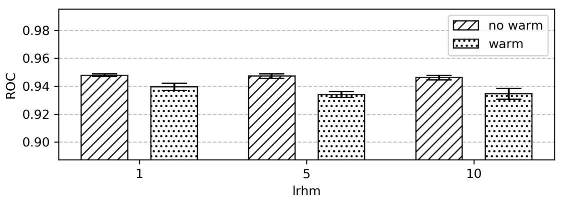
(C) ProliferativePattern



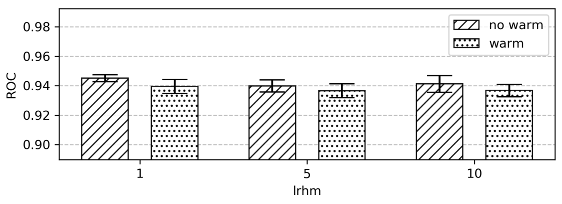
(D) HumanLba



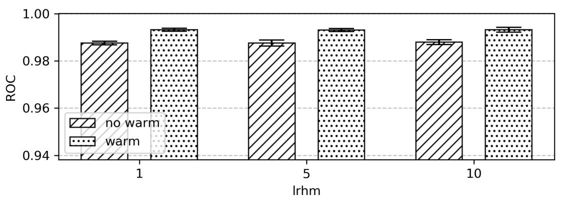
(E) BoneMarrow



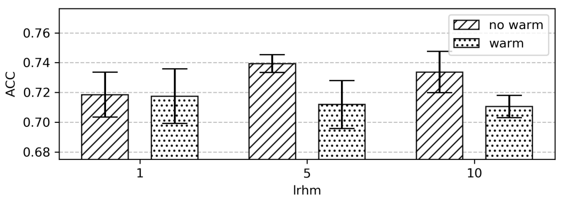
(F) Breast1



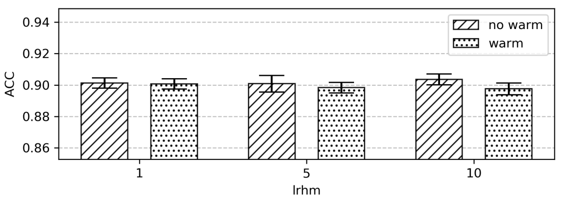
(G) Breast2



(H) Necrose

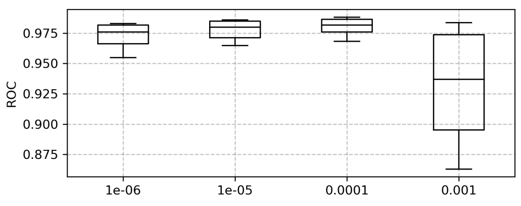


(I) MouseLba

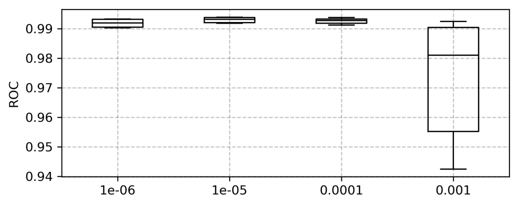


(J) Lung

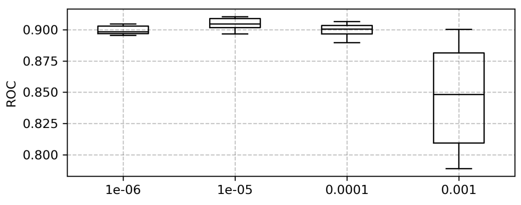
FIGURE B.2: Transfer performance for combinations of the hyperparameters γ_τ (learning rate heads multiplier) and w (warm up) with learning rate $\gamma = 10^{-4}$ on ResNet50.



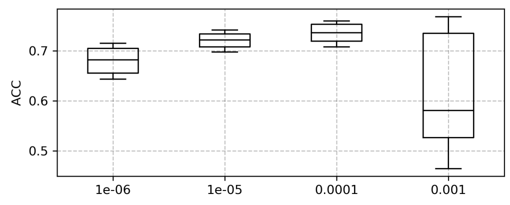
(A) CellInclusion



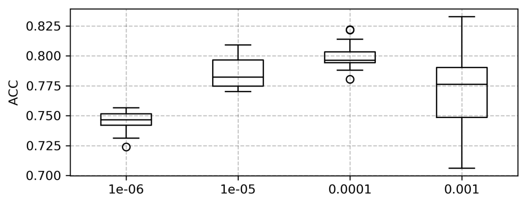
(B) Glomeruli



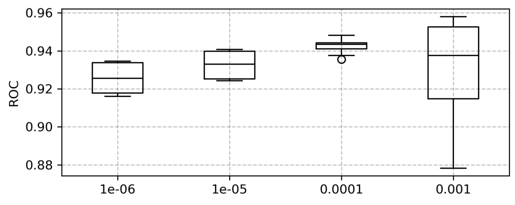
(C) ProliferativePattern



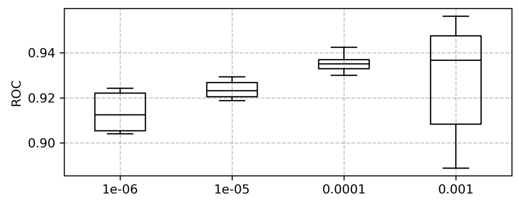
(D) HumanLba



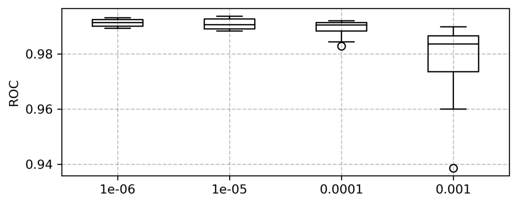
(E) BoneMarrow



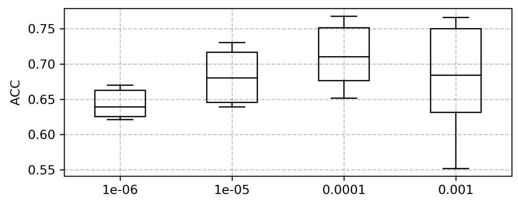
(F) Breast1



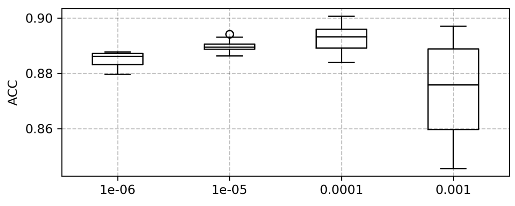
(G) Breast2



(H) Necrose

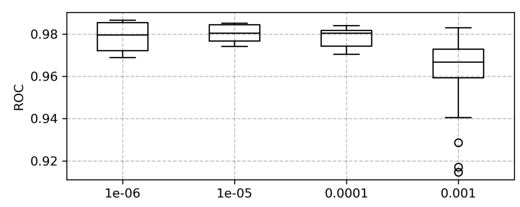


(I) MouseLba

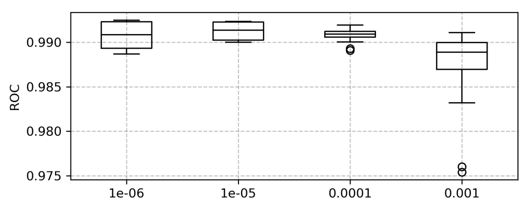


(J) Lung

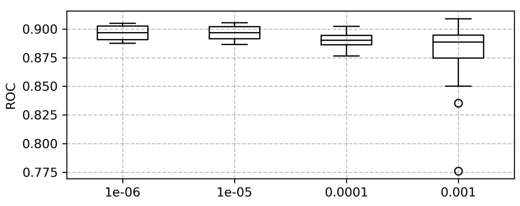
FIGURE B.3: Distributions of scores per learning rate on DenseNet121. Each boxplot results from the aggregation of the transfer scores of all models using the a learning rate value on the given network and dataset.



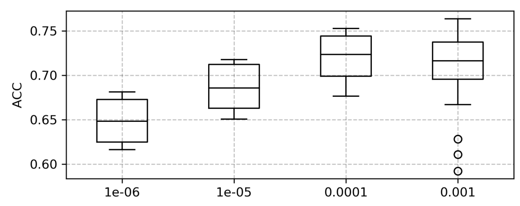
(A) CellInclusion



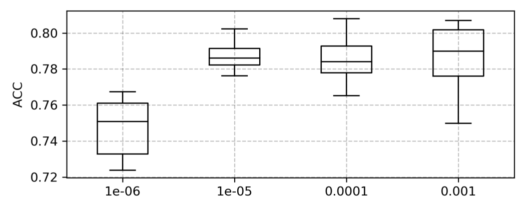
(B) Glomeruli



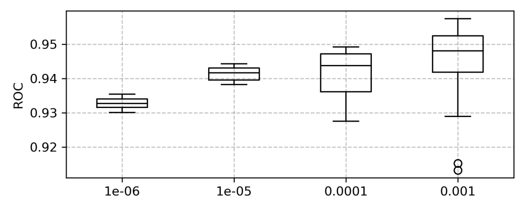
(C) ProliferativePattern



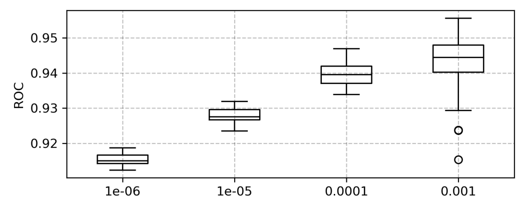
(D) HumanLba



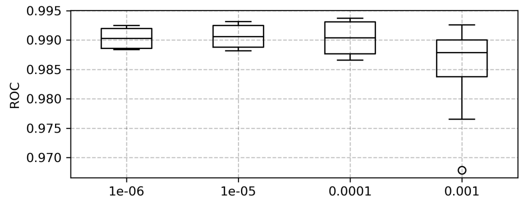
(E) BoneMarrow



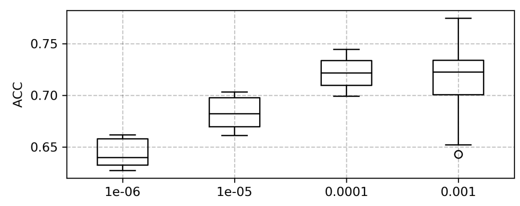
(F) Breast1



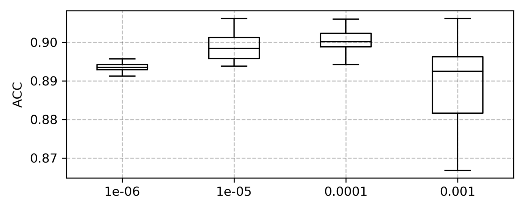
(G) Breast2



(H) Necrose



(I) MouseLba



(J) Lung

FIGURE B.4: Distributions of scores per learning rate on ResNet50. Each boxplot results from the aggregation of the transfer scores of all models using the a learning rate value on the given network and dataset.

Appendix

C Appendices for Chapter 6

C.1 Hyperparameters for soft *vs.* hard labels experiment

In this section, we list the hyperparameters we have used in our soft *vs.* hard labels experiment. They are reported in Tables C.1, C.2, C.3 for MoNuSeg, SegPC-2021 and GlaS respectively.

C.2 Hyperparameters for self-training performance at fixed n_l

This section provides a detailed list of the chosen hyperparameters foreach dataset. Self-training hyperparameters can be found in Table C.4. The other hyperparameters are listed in Table C.5.

C.3 Additional weighting strategies evaluated for the fixed n_l experiment

This section reports performance for all the weighting schemes actually evaluated (see Figures C.1, C.2 and C.3) for the fixed n_l experiment.

Weighting scheme	# comb.	Hyperparameters
Constant	6	$C \in \{0.01, 0.05, 0.1, 0.25, 0.5, 1.0\}$
Balance	1	/
Entropy	6	$w_{min} \in \{0.01, 0.05, 0.1, 0.25, 0.5, 0.75\}$
Consistency	1	$\eta = 2$
Merged	6	all combinations of $w_{min}, c(y_1, y_2)$ and η listed above
Total	20	

TABLE C.1: Hyperparameters for the soft *vs.* hard labels experiment on MoNuSeg.

Weighting scheme	# comb.	Hyperparameters
Constant	10	$C \in \{0.01, 0.05, 0.1, 0.25, 0.5, 1, 1.25, 1.75, 1.5, 2\}$
Balance	1	/
Entropy	6	$w_{min} \in \{0.01, 0.05, 0.1, 0.25, 0.5, 0.75\}$
Consistency	1	$\eta = 2$
Merged	6	all combinations of $w_{min}, c(y_1, y_2)$ and η listed above
Total	24	

TABLE C.2: Hyperparameters for the soft *vs.* hard labels experiment on SegPC-2021.

Weighting scheme	# comb.	Hyperparameters
Constant	8	$C \in \{0.01, 0.05, 0.1, 0.25, 0.5, 1, 2, 3\}$
Balance	1	/
Entropy	6	$w_{min} \in \{0.01, 0.05, 0.1, 0.25, 0.5, 0.75\}$
Consistency	1	$\eta = 2$
Merged	6	all combinations of $w_{min}, c(y_1, y_2)$ and η listed above
Total	22	

TABLE C.3: Hyperparameters for the soft *vs.* hard labels experiment on GlaS.

TABLE C.4: Training hyperparameters used for the experiments of Section 6.5.2.

Dataset	n_l	iter/epoch	tile size	W	E
MoNuSeg	2	100	512	10	50
SegPC-2021	30	300	512	10	50
GlaS	8	225	384	10	50

TABLE C.5: Self-training hyperparameters used for the experiments of Section 6.5.2 for MoNuSeg and SegPC-2021. The same hyperparameters have been used for GlaSexcept for the combination "*constant*" and $C = 0.2$.

Weight	C	w_{min}	η	Datasets		
				M	S	G
constant	0.01			✓	✓	
constant	0.5			✓	✓	✓
constant	1.0			✓	✓	✓
constant	2.0				✓	
entropy		0.1		✓	✓	✓
consistency			2	✓	✓	✓
merged		0.1	2	✓	✓	✓

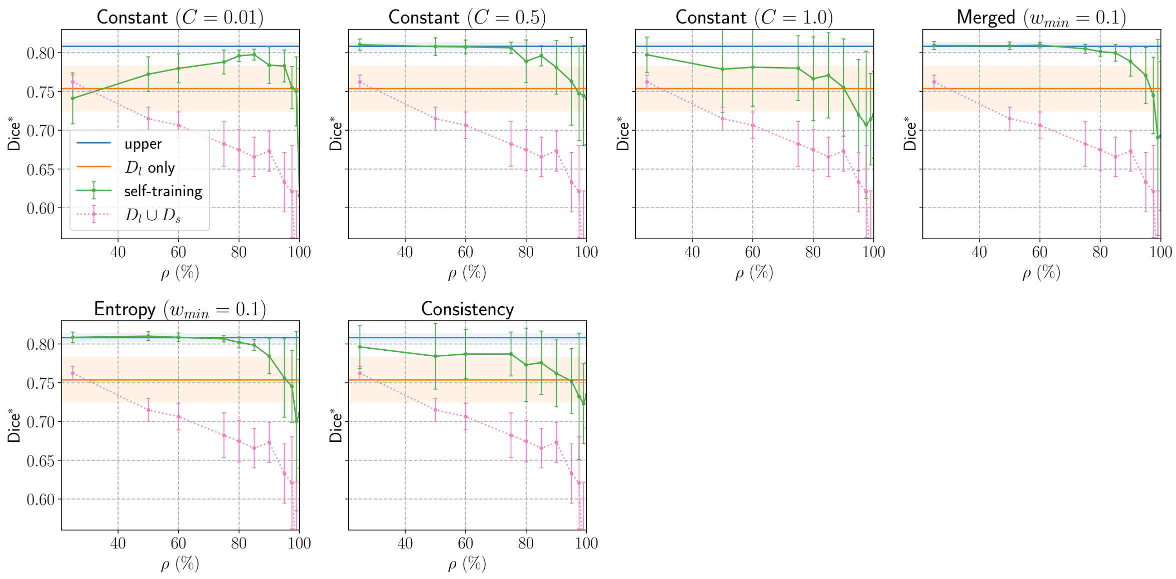


FIGURE C.1: MoNuSeg, see Figure 6.6 for explanation.

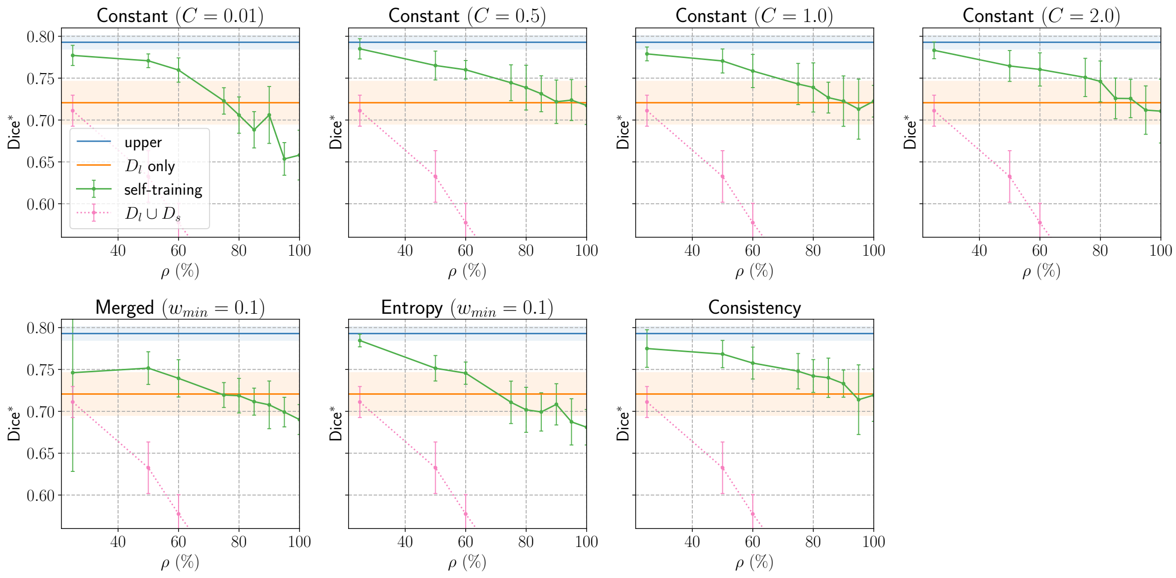


FIGURE C.2: SegPC-2021, see Figure 6.6 for explanation.

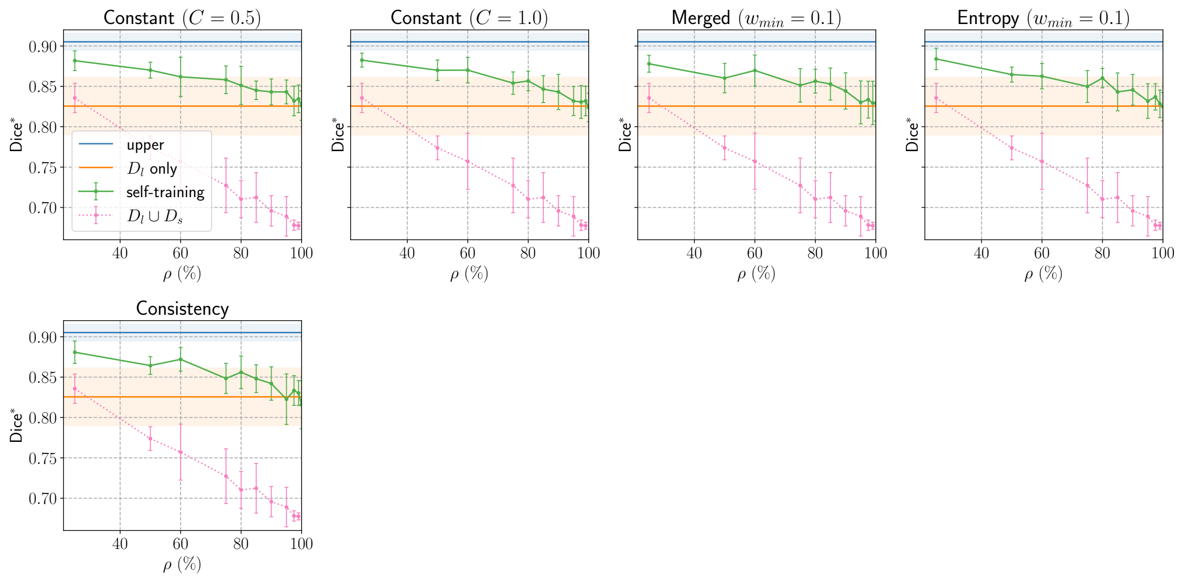


FIGURE C.3: GlaS, see Figure 6.6 for explanation.

C.4 The Thyroid FNAB ontology

The Thyroid FNAB dataset was labeled by experienced pathologists who followed a detailed ontology to categorize their annotations:

1. Architectural patterns (see examples in Figure 6.3):
   - Normal follicular architectural pattern
   - Proliferative follicular architectural pattern
   - Proliferative follicular architectural pattern (minor sign)
2. Nuclear features (see examples in Figure 6.4):
   - Papillary cell NOS
   - Normal follicular cells
   - Normal follicular cell with pseudo-inclusion (artefact)
   - Papillary cell with ground glass nuclei
   - Papillary cell with nuclear grooves
   - Papillary cell with inclusion
3. Others:
   - Macrophages
   - Red blood cells

- PN (polynuclear)
- Colloid
- Artefacts
- Background

D Appendix
Datasets

We thank our collaborators for bringing images and annotations on Cytomine:

- Thyroid FNAB: Caroline Degand and Isabelle Salmon (Erasme Hospital, Université Libre de Bruxelles, Belgium)
- Breast: Michel Reginster and Philippe Delvenne (University Hospital, ULiège, Belgium)
- Necrosis: Natacha Leroi and Philippe Martinive (GIGA-Cancer, ULiège, Belgium)
- HumanLba: Sandrine Rorive and Isabelle Salmon (Erasme Hospital, Université Libre de Bruxelles, Belgium)
- MouseLba: Natacha Rocks, Christine Fink, Fabienne Perin, and Didier Cataldo (GIGA-Cancer, ULiège, Belgium)
- Lung: Natacha Rocks, Christine Fink, Fabienne Perin, and Didier Cataldo (GIGA-Cancer, ULiège, Belgium)
- Glomeruli: Vannary Meas-Yedid and Jean-Christophe Olivo-Marin (Pasteur Institute, Paris, France), and Eric Thervet (Georges Pompidou European Hospital, Paris, France)
- BoneMarrow: Kainz et al. [89]

The actual diagnosis problem behind the Thyroid FNAB was discussed in Chapter 3 (Section 3.3.3). Thyroid FNAB was used in our three contributions as the classification datasets *CellInclusion* and *ProliferativePatterns* in Chapters 4 and 5 and as a segmentation dataset in Chapter 6. *Breast* was used to generate the classification dataset referred to as *Breast* and *Breast1* respectively in Chapters 4 and 5. It was also used to generate the classification dataset *Breast2* in Chapter 5. *BoneMarrow* was used in Chapter 5 only. The other datasets were used in both Chapters 4 and 5.

We also thank all the researchers who made their datasets publicly available:

- MITOS-ATYPIA 14: Roux et al. [170]
- Warwick CRC: Sirinukunwattana et al. [185]

- Andrew Janowczyk's tutorials: Janowczyk, Madabhushi, et al. [86]
- Stroma LBP: Linder et al. [122]
- TUPAC2016: Veta et al. [213]
- BACH18 Micro: Aresta et al. [8]
- Camelyon16: Bejnordi et al. [19]
- UMCM Colorectal: Kather et al. [94]
- GlaS: Sirinukunwattana et al. [184]
- MoNuSeg: Kumar et al. [109]
- SegPC-2021: Gupta et al. [71]

All these datasets were used throughout the thesis and greatly helped the development and evaluation of our algorithms.

E

Appendix

Biaflows

Abstract. Image analysis is key to extracting quantitative information from scientific microscopy images, but the methods involved are now often so refined that they can no longer be unambiguously described by written protocols. We introduce BIAFLOWS, an open-source web tool enabling to reproducibly deploy and benchmark bioimage analysis workflows coming from any software ecosystem. A curated instance of BIAFLOWS populated with 34 image analysis workflows and 15 microscopy image datasets recapitulating common bioimage analysis problems is available online. The workflows can be launched and assessed remotely by comparing their performance visually and according to standard benchmark metrics. We illustrated these features by comparing seven nuclei segmentation workflows, including deep-learning methods. BIAFLOWS enables to benchmark and share bioimage analysis workflows, hence safeguarding research results and promoting high-quality standards in image analysis. The platform is thoroughly documented and ready to gather annotated microscopy datasets and workflows contributed by the bioimaging community.

Publication. Ulysse Rubens*, Romain Mormont*, et al. "BIAFLOWS: A collaborative framework to reproducibly deploy and benchmark bioimage analysis workflows". In: *Patterns* 1.3 (2020), p. 100040 (*These authors contributed equally).

Article. https://www.cell.com/patterns/fulltext/S2666-3899%2820%2930045-3

Code. https://github.com/Neubias-WG5

Documentation. http://biaflows-doc.neubias.org/

List of Figures

List of Tables

Bibliography

[1] Saad Abbasi et al. “All-optical reflection-mode microscopic histology of unstained human tissues”. In: *Scientific reports* 9.1 (2019), pp. 1–11.

[2] Esther Abels et al. “Computational pathology definitions, best practices, and recommendations for regulatory guidance: a white paper from the Digital Pathology Association”. In: *The Journal of pathology* 249.3 (2019), pp. 286–294.

[3] Chris Allan et al. “OMERO: flexible, model-driven data management for experimental biology”. In: *Nature methods* 9.3 (2012), pp. 245–253.

[4] Mohamed Amgad et al. “Nucls: A scalable crowdsourcing, deep learning approach and dataset for nucleus classification, localization and segmentation”. In: *arXiv preprint arXiv:2102.09099* (2021).

[5] Mohamed Amgad et al. “Structured crowdsourcing enables convolutional segmentation of histology images”. In: *Bioinformatics* 35.18 (2019), pp. 3461–3467.

[6] Joseph Antony, Kevin McGuinness, Noel E O’Connor, and Kieran Moran. “Quantifying radiographic knee osteoarthritis severity using deep convolutional neural networks”. In: *Pattern Recognition (ICPR), 2016 23rd International Conference on*. IEEE. 2016, pp. 1195–1200.

[7] Salvatore Arena et al. “Intranuclear cytoplasmic inclusions in cytologically suspicious or malignant thyroid nodules: identification and correlation with echogenicity and size of the nodules”. In: *Endocrine* 46.1 (2014), pp. 114–122.

[8] Guilherme Aresta et al. “Bach: Grand challenge on breast cancer histology images”. In: *Medical image analysis* 56 (2019), pp. 122–139.

[9] Eirini Arvaniti and Manfred Claassen. “Coupling weak and strong supervision for classification of prostate cancer histopathology images”. In: *arXiv preprint arXiv:1811.07013* (2018).

[10] Marc Aubreville et al. “Mitosis domain generalization challenge”. In: *Zenodo, doi* 10 (2021), p. 5281.

[11] Morteza Babaie et al. “Classification and retrieval of digital pathology scans: A new dataset”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 8–16.

[12] Wenjia Bai et al. “Semi-supervised learning for network-based cardiac MR image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2017, pp. 253–260.

[13] Dana H Ballard. “Modular learning in neural networks.” In: *Aaai*. Vol. 647. 1987, pp. 279–284.

[14] Peter Bandi et al. “From detection of individual metastases to classification of lymph node status at the patient level: the CAMELYON17 challenge”. In: *IEEE transactions on medical imaging* 38.2 (2018), pp. 550–560.

[15] Peter Bankhead et al. “QuPath: Open source software for digital pathology image analysis”. In: *Scientific reports* 7.1 (2017), pp. 1–7.

[16] Yaniv Bar et al. “Chest pathology detection using deep learning with non-medical training”. In: *Biomedical Imaging (ISBI), 2015 IEEE 12th International Symposium on*. IEEE. 2015, pp. 294–297.

[17] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “Surf: Speeded up robust features”. In: *European conference on computer vision*. Springer. 2006, pp. 404–417.

[18] Neslihan Bayramoglu and Janne Heikkilä. “Transfer learning for cell nuclei classification in histopathology images”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 532–539.

[19] Babak Ehteshami Bejnordi et al. “Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer”. In: *Jama* 318.22 (2017), pp. 2199–2210.

[20] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. “Reconciling modern machine-learning practice and the classical bias–variance trade-off”. In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.

[21] Stuart Berg et al. “ilastik: interactive machine learning for (bio)image analysis”. In: *Nature Methods* (Sept. 2019). ISSN: 1548-7105. DOI: 10.1038/s41592-019-0582-9. URL: https://doi.org/10.1038/s41592-019-0582-9.

[22] biochain.com. *FFPE Tissue Samples Stored on Cassettes*. [Online; accessed Dec 29, 2021]. 2021. URL: https://www.biochain.com/wp-content/uploads/2017/08/ffpe-tissue-sample-500x333.jpg (visited on 12/29/2021).

[23] John-Melle Bokhorst et al. “Learning from sparsely annotated data for semantic segmentation in histopathology images”. In: *International Conference on Medical Imaging with Deep Learning–Full Paper Track*. 2018.

[24] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. “A training algorithm for optimal margin classifiers”. In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 144–152.

[25] Léon Bottou and Olivier Bousquet. “13 the tradeoffs of large-scale learning”. In: *Optimization for machine learning* (2011), p. 351.

[26] Hervé Bourlard and Yves Kamp. “Auto-association by multilayer perceptrons and singular value decomposition”. In: *Biological cybernetics* 59.4 (1988), pp. 291–294.

[27] Joseph Boyd et al. "Self-supervised representation learning using visual field expansion on digital pathology". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 639–647.

[28] Stevo Bozinovski. "Reminder of the First Paper on Transfer Learning in Neural Networks, 1976". In: *Informatica (Slovenia)* 44 (2020).

[29] L Breiman, JH Friedman, R Olshen, and CJ Stone. "Classification and Regression Trees". In: (1984).

[30] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[31] Nicole Bussola et al. "AI slipping on tiles: Data leakage in digital pathology". In: *International Conference on Pattern Recognition*. Springer. 2021, pp. 167–182.

[32] Andrey Bychkov and Ayana Suzuki. *Thyroid & parathyroid, cytology, bethesda system*. 2022. URL: https://www.pathologyoutlines.com/topic/thyroiddiagnostic.html (visited on 04/27/2022).

[33] Rich Caruana. "Multitask learning". In: *Machine learning* 28.1 (1997), pp. 41–75.

[34] Chengliang Chai and Guoliang Li. "Human-in-the-loop Techniques in Machine Learning". In: *Data Engineering* 37 (2020), p. 16.

[35] Woong-Gi Chang et al. "Domain-Specific Batch Normalization for Unsupervised Domain Adaptation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7354–7362.

[36] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, 2006.

[37] Dong Chen et al. "Joint cascade face detection and alignment". In: *European Conference on Computer Vision*. Springer. 2014, pp. 109–122.

[38] Richard J Chen and Rahul G Krishnan. "Self-supervised vision transformers learn visual concepts in histopathology". In: *arXiv preprint arXiv:2203.00585* (2022).

[39] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[40] Travers Ching et al. "Opportunities and obstacles for deep learning in biology and medicine". In: *Journal of The Royal Society Interface* 15.141 (2018), p. 20170387.

[41] François Chollet et al. *Keras*. https://github.com/keras-team/keras. 2015.

[42] Francesco Ciompi, Mitko Veta, Jeroen van der Laak, and Nasir Rajpoot. "Editorial Computational Pathology". In: *IEEE Journal of Biomedical and Health Informatics* 25.2 (2021), pp. 303–306.

[43] Francesco Ciompi et al. "Automatic classification of pulmonary peri-fissural nodules in computed tomography using an ensemble of 2D views and a convolutional neural network out-of-the-box". In: *Medical image analysis* 26.1 (2015), pp. 195–202.

[44] clinicallabmanager.com. *Picture of microscope slides*. [Online; accessed Dec 21, 2021]. 2021. URL: https://cdn.clinicallabmanager.com/assets/articleNo/24866/aImg/46099/a-photo-of-a-lab-technician-wearing-ppe-holding-glass-tissue-histology-slides-m.webp (visited on 12/21/2021).

[45] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 160–167.

[46] Radboud University Medical Center Computation Pathology Group. *Automated Slide Analysis Platform (ASAP)*. 2022. URL: https://computationalpathologygroup.github.io/ASAP/ (visited on 04/15/2022).

[47] datahacker.rs. *Schema of VGG16 and VGG19*. [Online; accessed Dec 14, 2021]. 2018. URL: https://media5.datahacker.rs/2018/11/vgg-ispravljeno-.png (visited on 12/14/2021).

[48] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[49] Jeff Donahue et al. "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International conference on machine learning*. 2014, pp. 647–655.

[50] Colin Doolan, Olga Colgan, and Sherri Heffner. *What is digital pathology?* Jan. 2019. URL: https://www.leicabiosystems.com/knowledge-pathway/what-is-digital-pathology/.

[51] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[52] Danielle D Elliott Range et al. "Application of a machine learning algorithm to predict malignancy in thyroid cytopathology". In: *Cancer cytopathology* 128.4 (2020), pp. 287–295.

[53] Catarina Eloy et al. "Digital Pathology Workflow Implementation at IPATIMUP". In: *Diagnostics* 11.11 (2021), p. 2111.

[54] Deng-Ping Fan et al. "Inf-net: Automatic covid-19 lung infection segmentation from ct images". In: *IEEE Transactions on Medical Imaging* 39.8 (2020), pp. 2626–2637.

[55] Rong-En Fan et al. "LIBLINEAR: A library for large linear classification". In: *Journal of machine learning research* 9.Aug (2008), pp. 1871–1874.

[56] Adrien Foucart, Olivier Debeir, and Christine Decaestecker. "SNOW: Semi-supervised, NOisy and/or Weak data for deep learning in digital pathology". In: *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*. IEEE. 2019, pp. 1869–1872.

[57] Robert M French. “Catastrophic forgetting in connectionist networks”. In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[58] Jevgenij Gamper et al. “PanNuke: An Open Pan-Cancer Histology Dataset for Nuclei Instance Segmentation and Classification”. In: *European Congress on Digital Pathology*. Springer. 2019, pp. 11–19.

[59] Stuart Geman, Elie Bienenstock, and René Doursat. “Neural networks and the bias/variance dilemma”. In: *Neural computation* 4.1 (1992), pp. 1–58.

[60] Pierre Geurts. “Bias vs variance decomposition for regression and classification”. In: *Data mining and knowledge discovery handbook*. Springer, 2009, pp. 733–746.

[61] Pierre Geurts, Damien Ernst, and Louis Wehenkel. “Extremely randomized trees”. In: *Machine learning* 63.1 (2006), pp. 3–42.

[62] Hossein Gholamalinezhad and Hossein Khosravi. “Pooling Methods in Deep Neural Networks, a Review”. In: *arXiv preprint arXiv:2009.07485* (2020).

[63] Bram van Ginneken, Arnaud AA Setio, Colin Jacobs, and Francesco Ciompi. “Off-the-shelf convolutional neural network features for pulmonary nodule detection in computed tomography scans”. In: *Biomedical Imaging (ISBI), 2015 IEEE 12th International Symposium on*. IEEE. 2015, pp. 286–289.

[64] Simon Graham et al. “CoNIC: Colon Nuclei Identification and Counting Challenge 2022”. In: *arXiv preprint arXiv:2111.14485* (2021).

[65] Simon Graham et al. “Hover-net: Simultaneous segmentation and classification of nuclei in multi-tissue histology images”. In: *Medical Image Analysis* 58 (2019), p. 101563.

[66] Simon Graham et al. “Lizard: A Large-Scale Dataset for Colonic Nuclear Instance Segmentation and Classification”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 684–693.

[67] Simon Graham et al. “MILD-Net: Minimal information loss dilated network for gland instance segmentation in colon histology images”. In: *Medical image analysis* 52 (2019), pp. 199–211.

[68] *Grand Challenge*. [Online; accessed Feb 09, 2022]. URL: https://grand-challenge.org/.

[69] Yves Grandvalet and Yoshua Bengio. “Semi-supervised learning by entropy minimization”. In: *Advances in neural information processing systems* 17 (2004).

[70] NP Group et al. “Histopathology is ripe for automation”. In: *Nature Biomedical Engineering* 1.12 (2017), p. 925.

[71] A Gupta, R Gupta, S Gehlot, and S Goswami. “Segpc-2021: Segmentation of multiple myeloma plasma cells in microscopic images”. In: *IEEE Dataport* 1.1 (2021), p. 1.

[72] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. "Gene selection for cancer classification using support vector machines". In: *Machine learning* 46.1-3 (2002), pp. 389–422.

[73] Maryam Haghighat et al. "PathProfiler: Automated Quality Assessment of Retrospective Histopathology Whole-Slide Image Cohorts by Artificial Intelligence, A Case Study for Prostate Cancer Research". In: *medRxiv* (2021). DOI: 10.1101/2021.09.24.21263762. eprint: https://www.medrxiv.org/content/early/2021/09/27/2021.09.24.21263762.full.pdf. URL: https://www.medrxiv.org/content/early/2021/09/27/2021.09.24.21263762.

[74] Steven I Hajdu. "The first use of the microscope in medicine". In: *Annals of Clinical & Laboratory Science* 32.3 (2002), pp. 309–310.

[75] Chu Han et al. "Multi-Layer Pseudo-Supervision for Histopathology Tissue Semantic Segmentation using Patch-level Classification Labels". In: 2021. arXiv: 2110.08048.

[76] Trevor Hastie, Jerome H Friedman, and Robert Tibshirani. *The elements of statistical learning: Data mining, inference, and prediction*. springer open, 2017.

[77] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[78] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[79] Le Hou et al. "Automatic histopathology image analysis with CNNs". In: *Scientific Data Summit (NYSDS), 2016 New York*. IEEE. 2016, pp. 1–6.

[80] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[81] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. "Densely connected convolutional networks". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR2017*. 2017, pp. 2261–2269.

[82] Matthew Humerick. "Taking AI personally: how the EU must learn to balance the interests of personal data privacy & artificial intelligence". In: *Santa Clara High Tech. LJ* 34 (2017), p. 393.

[83] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *International Conference on Machine Learning, ICML2015*. Vol. 37. 2015, pp. 448–456.

[84] Stephan W Jahn, Markus Plass, and Farid Moinfar. "Digital Pathology: Advantages, Limitations and Emerging Perspectives". In: *Journal of Clinical Medicine* 9.11 (2020), p. 3697.

[85] Amit Kumar Jaiswal et al. "Semi-supervised learning for cancer detection of lymph node metastases". In: *arXiv preprint arXiv:1906.09587* (2019).

[86] Andrew Janowczyk, Anant Madabhushi, et al. "Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases". In: *Journal of Pathology Informatics* 7.1 (2016), p. 29.

[87] Andrew Janowczyk et al. "HistoQC: an open-source quality control tool for digital pathology slides". In: *JCO clinical cancer informatics* 3 (2019), pp. 1–7.

[88] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

[89] Philipp Kainz, Harald Burgsteiner, Martin Asslaber, and Helmut Ahammer. "Training echo state networks for rotation-invariant bone marrow cell classification". In: *Neural Computing and Applications* 28.6 (2017), pp. 1277–1292.

[90] Dong Un Kang et al. *PAIP2021: Perineural Invasion in Multiple Organ Cancer*. [Online; accessed on May 31, 2022]. 2022. URL: `https://paip2021.grand-challenge.org/` (visited on 05/31/2022).

[91] Hongtao Kang et al. "Stainnet: a fast and robust stain normalization network". In: *Frontiers in Medicine* 8 (2021).

[92] Petros Karakitsos et al. "Learning vector quantizer in the investigation of thyroid lesions". In: *Analytical and Quantitative Cytology and Histology* 21.3 (1999), pp. 201–208.

[93] Andrej Karpathy. *ImageNet*. [Online; accessed July 14, 2022]. 2022. URL: `https://cs.stanford.edu/people/karpathy/cnnembed/cnn_embed_full_1k_icon.jpg` (visited on 07/14/2022).

[94] Jakob Nikolas Kather et al. "Multi-class texture analysis in colorectal cancer histology". In: *Scientific reports* 6 (2016), p. 27988.

[95] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. "Leakage in data mining: Formulation, detection, and avoidance". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6.4 (2012), pp. 1–21.

[96] Brie Kezlarian and Oscar Lin. "Artificial intelligence in thyroid fine needle aspiration biopsies". In: *Acta Cytologica* 65.4 (2021), pp. 324–329.

[97] Umair Akhtar Hasan Khan et al. "Improving Prostate Cancer Detection with Breast Histopathology Images". In: *European Congress on Digital Pathology*. Springer. 2019, pp. 91–99.

[98] Brady Kieffer, Morteza Babaie, Shivam Kalra, and HR Tizhoosh. "Convolutional Neural Networks for Histopathology Image Classification: Training vs. Using Pre-Trained Networks". In: *arXiv preprint arXiv:1710.05726* (2017).

[99] Edward Kim, Miguel Corte-Real, and Zubair Baloch. "A deep semantic mobile application for thyroid cytopathology". In: *Medical Imaging 2016: PACS and Imaging Informatics: Next Generation and Innovations*. Vol. 9789. International Society for Optics and Photonics. 2016, 97890A.

[100] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[101] Iasonas Kokkinos. "Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6129–6138.

[102] Daisuke Komura and Shumpei Ishikawa. "Machine learning methods for histopathological image analysis". In: *Computational and structural biotechnology journal* 16 (2018), pp. 34–42.

[103] Navid Alemi Koohbanani, Mostafa Jahanifar, Neda Zamani Tajadin, and Nasir Rajpoot. "NuClick: a deep learning framework for interactive segmentation of microscopic images". In: *Medical Image Analysis* 65 (2020), p. 101771.

[104] Navid Alemi Koohbanani et al. "Self-path: Self-supervision for classification of pathology images with limited annotations". In: *IEEE Transactions on Medical Imaging* 40.10 (2021), pp. 2845–2856.

[105] Simon Kornblith, Jonathon Shlens, and Quoc V Le. "Do better imagenet models transfer better?" In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 2661–2671.

[106] Oren Z Kraus et al. "Automated analysis of high-content microscopy data with deep learning". In: *Molecular systems biology* 13.4 (2017), p. 924.

[107] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[108] Meghana Dinesh Kumar et al. "A Comparative Study of CNN, BoVW and LBP for Classification of Histopathological Images". In: *arXiv preprint arXiv:1710.01249* (2017).

[109] Neeraj Kumar et al. "A multi-organ nucleus segmentation challenge". In: *IEEE transactions on medical imaging* 39.5 (2019), pp. 1380–1391.

[110] Alina Kuznetsova et al. "The open images dataset v4". In: *International Journal of Computer Vision* 128.7 (2020), pp. 1956–1981.

[111] Vivek Kwatra et al. "Graphcut textures: Image and video synthesis using graph cuts". In: *Acm transactions on graphics (tog)* 22.3 (2003), pp. 277–286.

[112] Samuli Laine and Timo Aila. "Temporal ensembling for semi-supervised learning". In: *arXiv preprint arXiv:1610.02242* (2016).

[113] Yann Le Cun and Françoise Fogelman-Soulié. "Modèles connexionnistes de l'apprentissage". In: *Intellectica* 2.1 (1987), pp. 114–143.

[114] Y LeCun and I Misra. *Self-supervised Learning: The Dark Matter of Intelligence*. 2021.

[115] Yann LeCun et al. "Handwritten digit recognition with a back-propagation network". In: *Advances in neural information processing systems* 2 (1989).

[116] Dong-Hyun Lee et al. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks". In: *Workshop on challenges in representation learning, ICML*. Vol. 3. 2013, p. 896.

[117] Aleš Leonardis and Sanja Fidler. "Learning hierarchical representations of object categories for robot vision". In: *Robotics Research*. Springer, 2010, pp. 99–110.

[118] Jiahui Li et al. "Signet ring cell detection with a semi-supervised learning framework". In: *International conference on information processing in medical imaging*. Springer. 2019, pp. 842–854.

[119] Jiayun Li et al. "An EM-based semi-supervised deep learning approach for semantic segmentation of histopathological images from radical prostatectomies". In: *Computerized Medical Imaging and Graphics* 69 (2018), pp. 125–133.

[120] Yanghao Li et al. "Adaptive batch normalization for practical domain adaptation". In: *Pattern Recognition* 80 (2018), pp. 109–117.

[121] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[122] Nina Linder et al. "Identification of tumor epithelium and stroma in tissue microarrays using texture analysis". In: *Diagnostic pathology* 7.1 (2012), p. 22.

[123] Geert Litjens et al. "1399 H&E-stained sentinel lymph node sections of breast cancer patients: the CAMELYON dataset". In: *GigaScience* 7.6 (2018), giy065.

[124] Geert Litjens et al. "A survey on deep learning in medical image analysis". In: *Medical image analysis* 42 (2017), pp. 60–88.

[125] Xiaodong Liu et al. "Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational Linguistics, June 2015, pp. 912–921.

[126] Yehe Liu, Richard M Levenson, and Michael W Jenkins. "Slide over: advances in slide-free optical microscopy as drivers of diagnostic pathology". In: *The American journal of pathology* (2021).

[127] Kevis-Kokitsi Maninis, Sergi Caelles, Jordi Pont-Tuset, and Luc Van Gool. "Deep extreme cut: From extreme points to object segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 616–625.

[128] Raphaël Marée. "Open practices and resources for collaborative digital pathology". In: *Frontiers in Medicine* (2019), p. 255.

[129] Raphaël Marée. "The need for careful data collection for pattern recognition in digital pathology". In: *Journal of pathology informatics* 8 (2017).

[130] Raphaël Marée, Stephane Dallongeville, J-C Olivo-Marin, and Vannary Meas-Yedid. "An approach for detection of glomeruli in multisite digital pathology". In: *Biomedical Imaging (ISBI), 2016 IEEE 13th International Symposium on*. IEEE. 2016, pp. 1033–1036.

[131] Raphaël Marée, Pierre Geurts, and Louis Wehenkel. "Towards generic image classification using tree-based learning: An extensive empirical study". In: *Pattern Recognition Letters* 74 (2016), pp. 17–23.

[132] Raphaël Marée et al. "Collaborative analysis of multi-gigapixel imaging data using Cytomine". In: *Bioinformatics* 32.9 (2016), pp. 1395–1401.

[133] Niccolo Marini et al. "Multi_Scale_Tools: a python library to exploit multi-scale whole slide images". In: *Data-Enabled Intelligence for Medical Technology Innovation, Volume I* (2022).

[134] Michael T McCann et al. "Automated histology analysis: Opportunities for signal processing". In: *IEEE Signal Processing Magazine* 32.1 (2014), pp. 78–87.

[135] Robert K McConnell. *Method of and apparatus for pattern recognition*. US Patent 4,567,610. Jan. 1986.

[136] Alfonso Medela et al. "Few Shot Learning in Histopathological Images: Reducing the Need of Labeled Data on Biological Datasets". In: *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*. IEEE. 2019, pp. 1860–1864.

[137] Thomas Mensink et al. "Factors of influence for transfer learning across diverse appearance domains and task types". In: *arXiv preprint arXiv:2103.13318* (2021).

[138] Kyle Millar, Adriel Cheng, Hong Gunn Chew, and Cheng-Chew Lim. "Using convolutional neural networks for classifying malicious network traffic". In: *Deep Learning Applications for Cyber Security*. Springer, 2019, pp. 103–126.

[139] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. "Cross-stitch networks for multi-task learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3994–4003.

[140] Tom M. Mitchell. *Machine learning*. New York, NY [u.a.: McGraw-Hill, 1997. URL: http://www.amazon.com/Machine-Learning-Tom-M-Mitchell/dp/0070428077.

[141] Romain Mormont, Jean-Michel Begon, Renaud Hoyoux, and Raphaël Marée. "SLDC: an open-source workflow for object detection in multi-gigapixel images". In: *The 25th Belgian-Dutch Conference on Machine Learning (Benelearn)*. 2016.

[142] Romain Mormont, Pierre Geurts, and Raphaël Marée. "Comparison of deep transfer learning strategies for digital pathology". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 2262–2271.

[143] Romain Mormont, Pierre Geurts, and Raphaël Marée. "Multi-task pre-training of deep neural networks for digital pathology". In: *IEEE journal of biomedical and health informatics* 25.2 (2020), pp. 412–421.

[144] Romain Mormont, Mehdi Testouri, Raphaël Marée, and Pierre Geurts. "Relieving pixel-wise labeling effort for pathology image segmentation with self-training". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. Accepted for publication. 2022.

[145] Pierre Moulin, Katrien Grünberg, Erio Barale-Thomas, and Jeroen van der Laak. *IMI—Bigpicture: A Central Repository for Digital Pathology*. 2021.

[146] Myura Nagendran et al. "Artificial intelligence versus clinicians: systematic review of design, reporting standards, and claims of deep learning studies". In: *Bmj* 368 (2020).

[147] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. "A review on the attention mechanism of deep learning". In: *Neurocomputing* 452 (2021), pp. 48–62.

[148] Xipeng Pan et al. "Multi-task Deep Learning for Fine-Grained Classification/Grading in Breast Cancer Histopathological Images". In: *International Symposium on Artificial Intelligence and Robotics*. Springer. 2018, pp. 85–95.

[149] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[150] Ankush Patel et al. "Contemporary whole slide imaging devices and their applications within the modern pathology department: A selected hardware review". In: *Journal of Pathology Informatics* 12 (2021).

[151] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[152] Mohammad Peikari, Sherine Salama, Sharon Nofech-Mozes, and Anne L Martel. "A cluster-then-label semi-supervised learning approach for pathology image classification". In: *Scientific reports* 8.1 (2018), pp. 1–13.

[153] Jialin Peng and Ye Wang. "Medical image segmentation with limited supervision: A review of deep network models". In: *IEEE Access* (2021).

[154] Mark Peplow. "Citizen science lures gamers into Sweden's Human Protein Atlas". In: *Nature Biotechnology* 34.5 (2016), pp. 452–454.

[155] Hieu Pham, Zihang Dai, Qizhe Xie, and Quoc V Le. "Meta pseudo labels". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 11557–11568.

[156] Francesco Ponzio, Gianvito Urgese, Elisa Ficarra, and Santa Di Cataldo. "Dealing with Lack of Training Data for Convolutional Neural Networks: The Case of Digital Pathology". In: *Electronics* 8.3 (Feb. 2019), p. 256. ISSN: 2079-9292. DOI: 10.3390/electronics8030256.

[157] Pijush Kanti Dutta Pramanik, Saurabh Pal, and Moutan Mukhopadhyay. "Healthcare big data: A comprehensive overview". In: *Intelligent systems for healthcare management and delivery* (2019), pp. 72–100.

[158] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1.1 (1986), pp. 81–106.

[159] Alvin Rajkomar, Jeffrey Dean, and Isaac Kohane. "Machine learning in medicine". In: *New England Journal of Medicine* 380.14 (2019), pp. 1347–1358.

[160] Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. "Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.1 (2017), pp. 121–135.

[161] Hariharan Ravishankar et al. "Understanding the mechanisms of deep transfer learning for medical images". In: *Deep Learning and Data Labeling for Medical Applications*. Springer, 2016, pp. 188–196.

[162] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN features off-the-shelf: an astounding baseline for recognition". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. IEEE. 2014, pp. 512–519.

[163] Emily S Reisenbichler and Debra L Zynger. *Breast, general, staging*. [Online; accessed on Apr 25, 2022]. 2022. URL: https://www.pathologyoutlines.com/topic/breastmalignantstaging.html (visited on 04/25/2022).

[164] Stephanie Robertson, Hossein Azizpour, Kevin Smith, and Johan Hartman. "Digital image analysis in breast pathology-from image processing techniques to artificial intelligence". In: *Translational Research* 194 (2018), pp. 19–35.

[165] Marcial Garcia Rojo and Gloria Bueno. "Analysis of the impact of high-resolution monitors in digital pathology". In: *Journal of Pathology Informatics* 6 (2015).

[166] Geoffrey Rolls. "Process of Fixation and the Nature of Fixatives". In: *Fixation and Fixatives Retrieved* 20.06 (2012), p. 2014.

[167] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[168] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[169] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. “" GrabCut" interactive foreground extraction using iterated graph cuts”. In: *ACM transactions on graphics (TOG)* 23.3 (2004), pp. 309–314.

[170] Ludovic Roux et al. “Mitos & atypia”. In: *Image Pervasive Access Lab (IPAL), Agency Sci., Technol. & Res. Inst. Infocom Res., Singapore, Tech. Rep* 1 (2014), pp. 1–8.

[171] Monika Roychowdhury. *Breast cancer histologic grading*. [Online; accessed on Apr 22, 2022]. 2022. URL: https://www.pathologyoutlines.com/topic/breastmalignanthistologic.html (visited on 04/22/2022).

[172] Ulysse Rubens et al. “BIAFLOWS: A collaborative framework to reproducibly deploy and benchmark bioimage analysis workflows”. In: *Patterns* 1.3 (2020), p. 100040.

[173] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International conference on computer vision*. Ieee. 2011, pp. 2564–2571.

[174] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[175] Marlen Runz et al. ”Normalization of HE-stained histological images using cycle consistent generative adversarial networks”. In: *Diagnostic Pathology* 16.1 (2021), pp. 1–10.

[176] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[177] Ravi K Samala et al. “Multi-task transfer learning deep convolutional neural network: application to computer-aided diagnosis of breast cancer on mammograms”. In: *Physics in Medicine & Biology* 62.23 (2017), pp. 8894–8908.

[178] Henry Scudder. “Probability of error of some adaptive pattern-recognition machines”. In: *IEEE Transactions on Information Theory* 11.3 (1965), pp. 363–371.

[179] Pierre Sermanet et al. “OverFeat: Integrated recognition, localization and detection using convolutional networks”. In: *International Conference on Learning Representations (ICLR2014), CBLS*. 2014.

[180] Hong Shang et al. “What And How Other Datasets Can Be Leveraged For Medical Imaging Classification”. In: *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*. IEEE. 2019, pp. 814–818.

[181] Shayne Shaw et al. “Teacher-student chain for efficient semi-supervised histology image classification”. In: *arXiv preprint arXiv:2003.08797* (2020).

[182] Hoo-Chang Shin et al. “Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning”. In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1285–1298.

[183] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[184] Korsuk Sirinukunwattana et al. "Gland segmentation in colon histology images: The glas challenge contest". In: *Medical image analysis* 35 (2017), pp. 489–502.

[185] Korsuk Sirinukunwattana et al. "Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images." In: *IEEE Trans. Med. Imaging* 35.5 (2016), pp. 1196–1206.

[186] Kihyuk Sohn et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 596–608.

[187] Yang Song, Hang Chang, Heng Huang, and Weidong Cai. "Supervised intra-embedding of fisher vectors for histopathology image classification". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2017, pp. 99–106.

[188] Fabio A Spanhol, Luiz S Oliveira, Caroline Petitjean, and Laurent Heutte. "A dataset for breast cancer histopathological image classification". In: *IEEE Transactions on Biomedical Engineering* 63.7 (2015), pp. 1455–1462.

[189] Fabio A Spanhol et al. "Deep features for breast cancer histopathological image classification". In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2017, pp. 1868–1873.

[190] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[191] Nikolas Stathonikos, Mitko Veta, André Huisman, and Paul J van Diest. "Going fully digital: Perspective of a Dutch academic pathology lab". In: *Journal of pathology informatics* 4.1 (2013), p. 15.

[192] Mark Stidworthy and Simon Priestnall. "Getting the best results from veterinary histopathology". In: *In Practice* 33.6 (2011), pp. 252–260.

[193] Gjorgji Strezoski, Nanne van Noord, and Marcel Worring. "Many Task Learning With Task Routing". In: *IEEE/CVF International Conference on Computer Vision (ICCV2019)*. 2019, pp. 1375–1384.

[194] Gjorgji Strezoski and Marcel Worring. "Omniart: multi-task deep learning for artistic data analysis". In: *arXiv preprint arXiv:1708.00684* (2017).

[195] Hai Su, Xiaoshuang Shi, Jinzheng Cai, and Lin Yang. "Local and global consistency regularized mean teacher for semi-supervised nuclei classification". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2019, pp. 559–567.

[196] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. "Inception-v4, inception-resnet and the impact of residual connections on learning." In: *31st AAAI Conference on Artificial Intelligence, AAAI 2017*. 2017, pp. 4278–4284.

[197] Christian Szegedy et al. "Going deeper with convolutions". In: Cvpr. 2015.

[198] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[199] Nima Tajbakhsh et al. "Convolutional neural networks for medical image analysis: Full training or fine tuning?" In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1299–1312.

[200] Nima Tajbakhsh et al. "Embracing imperfect datasets: A review of deep learning solutions for medical image segmentation". In: *Medical Image Analysis* 63 (2020), p. 101693.

[201] Chuanqi Tan et al. "A Survey on Deep Transfer Learning". In: *Artificial Neural Networks and Machine Learning – ICANN 2018*. Ed. by Věra Kůrková et al. Cham: Springer International Publishing, 2018, pp. 270–279. ISBN: 978-3-030-01424-7.

[202] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[203] Mingxing Tan and Quoc Le. "Efficientnetv2: Smaller models and faster training". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10096–10106.

[204] Syed Ahmed Taqi, Syed Abdus Sami, Lateef Begum Sami, and Syed Ahmed Zaki. "A review of artifacts in histopathology". In: *Journal of oral and maxillofacial pathology: JOMFP* 22.2 (2018), p. 279.

[205] Antti Tarvainen and Harri Valpola. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results". In: *Advances in neural information processing systems* 30 (2017).

[206] David Tellez et al. "Whole-slide mitosis detection in H&E breast histology using PHH3 as a reference to train distilled stain-invariant convolutional networks". In: *IEEE transactions on medical imaging* 37.9 (2018), pp. 2126–2136.

[207] Jordi Temprana-Salvador et al. "DigiPatICS: Digital Pathology Transformation of the Catalan Health Institute Network of 8 Hospitals—Planification, Implementation, and Preliminary Results". In: *Diagnostics* 12.4 (2022), p. 852.

[208] Hamid Tizhoosh and Liron Pantanowitz. "Artificial intelligence and digital pathology: Challenges and opportunities". In: *Journal of pathology informatics* 9.1 (2018), p. 38.

[209] David C Van Essen and Jack L Gallant. "Neural mechanisms of form and motion processing in the primate visual system". In: *Neuron* 13.1 (1994), pp. 1–10.

[210] Yves-Rémi Van Eycke, Adrien Foucart, and Christine Decaestecker. “Strategies to reduce the expert supervision required for deep Learning-Based segmentation of histopathological images”. In: *Frontiers in medicine* 6 (2019), p. 222.

[211] Mart Van Rijthoven et al. *Tumor infiltrating lymphocytes in breast cancer: the TIGER challenge*. [Online; accessed on May 31, 2022]. 2022. URL: https://tiger.grand-challenge.org/ (visited on 05/31/2022).

[212] Vladimir Vapnik. “Principles of risk minimization for learning theory”. In: *Advances in neural information processing systems*. 1992, pp. 831–838.

[213] Mitko Veta et al. “Predicting breast tumor proliferation from whole-slide images: the TUPAC16 challenge”. In: *Medical image analysis* 54 (2019), pp. 111–121.

[214] Quoc Dang Vu et al. “Methods for segmentation and classification of digital microscopy tissue images”. In: *Frontiers in bioengineering and biotechnology* (2019), p. 53.

[215] Noorul Wahab et al. “Semantic annotation for computational pathology: Multidisciplinary experience and best practice recommendations”. In: *The Journal of Pathology: Clinical Research* 8.2 (2022), pp. 116–128.

[216] Kafeng Wang et al. “Pay attention to features, transfer learn faster CNNs”. In: *International Conference on Learning Representations*. 2019.

[217] Xiyue Wang et al. “Transpath: Transformer-based self-supervised learning for histopathological image classification”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2021, pp. 186–195.

[218] Donald L Weaver. “Pathology evaluation of sentinel lymph nodes in breast cancer: protocol recommendations and rationale”. In: *Modern Pathology* 23.2 (2010), S26–S32.

[219] John N Weinstein et al. “The cancer genome atlas pan-cancer analysis project”. In: *Nature genetics* 45.10 (2013), pp. 1113–1120.

[220] WHO et al. “WHO report on cancer: setting priorities, investing wisely and providing care for all”. In: (2020).

[221] Adrian Wolny, Qin Yu, Constantin Pape, and Anna Kreshuk. “Sparse Object-level Supervision for Instance Segmentation with Pixel Embeddings”. In: *arXiv preprint arXiv:2103.14572* (2021).

[222] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. “Self-training with noisy student improves imagenet classification”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10687–10698.

[223] Xinpeng Xie et al. “Instance-aware self-supervised learning for nuclei segmentation”. In: *International conference on medical image computing and computer-assisted intervention*. Springer. 2020, pp. 341–350.

[224] Feng Xu et al. “Predicting Axillary Lymph Node Metastasis in Early Breast Cancer Using Deep Learning on Primary Tumor Biopsy Slides”. In: *Frontiers in Oncology* (2021), p. 4133.

[225] Qiang Yang, Yu Zhang, Wenyuan Dai, and Sinno Jialin Pan. *Transfer Learning*. Cambridge Unversity Press, 2020.

[226] David Yarowsky. “Unsupervised word sense disambiguation rivaling supervised methods”. In: *33rd annual meeting of the association for computational linguistics*. 1995, pp. 189–196.

[227] Meei J Yeung and Jonathan W Serpell. “Management of the solitary thyroid nodule”. In: *The oncologist* 13.2 (2008), pp. 105–112.

[228] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. “How transferable are features in deep neural networks?” In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[229] Jiahui Yu et al. “Coca: Contrastive captioners are image-text foundation models”. In: *arXiv preprint arXiv:2205.01917* (2022).

[230] Tianhe Yu et al. “Gradient surgery for multi-task learning”. In: *arXiv preprint arXiv:2001.06782* (2020).

[231] Yuhai Yu et al. “Deep transfer learning for modality classification of medical images”. In: *Information* 8.3 (2017), p. 91.

[232] Mark D Zarella et al. “A practical guide to whole slide imaging: a white paper from the digital pathology association”. In: *Archives of pathology & laboratory medicine* 143.2 (2019), pp. 222–234.

[233] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[234] X Zhai, A Kolesnikov, N Houlsby, and L Beyer. “Scaling vision transformers. arXiv 2021”. In: *arXiv preprint arXiv:2106.04560* ().

[235] Chiyuan Zhang et al. “Understanding deep learning (still) requires rethinking generalization”. In: *Communications of the ACM* 64.3 (2021), pp. 107–115.

[236] Wen Zhang, Lingfei Deng, Lei Zhang, and Dongrui Wu. “Overcoming negative transfer: A survey”. In: *arXiv preprint arXiv:2009.00909* (2020).

[237] Wenlu Zhang et al. “Deep Model Based Transfer and Multi-Task Learning for Biological Image Analysis”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’15. Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 1475–1484. ISBN: 9781450336642. DOI: `10.1145/2783258.2783304`. URL: `https://doi.org/10.1145/2783258.2783304`.

[238] Yu Zhang and Qiang Yang. “A survey on multi-task learning”. In: *arXiv preprint arXiv:1707.08114* (2017).

[239] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. “Facial landmark detection by deep multi-task learning”. In: *European conference on computer vision*. Springer. 2014, pp. 94–108.

[240] Bingchao Zhao et al. “RestainNet: a self-supervised digital re-stainer for stain normalization”. In: *arXiv preprint arXiv:2202.13804* (2022).

[241] Xiangxin Zhu and Deva Ramanan. “Face detection, pose estimation, and landmark localization in the wild”. In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 2879–2886.

[242] Yi Zhu et al. “Improving semantic segmentation via self-training”. In: *arXiv preprint arXiv:2004.14960* (2020).

[243] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.

[244] Barret Zoph et al. “Rethinking pre-training and self-training”. In: *Advances in neural information processing systems* 33 (2020), pp. 3833–3845.