Deep Learning for Content-Based Image Retrieval in Biomedical applications

Auteur : Schyns, Axelle
Promoteur(s) : Maree, Raphael; Geurts, Pierre
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2022-2023
URI/URL : http://hdl.handle.net/2268.2/17731

Master Thesis

Deep Learning for Content-based Image Retrieval in Biomedical Applications

Author:

Axelle Schyns

Supervisors:

Pierre Geurts

Raphaël Marée

Academic Year: 2022 - 2023

Place: University of Liège - School of Engineering and Computer Science

Master in

Computer Science and Engineering, Intelligent Systems

Master's thesis completed in order to obtain the degree of Master of Science in Computer Engineering by Axelle Schyns

Abstract

Due to advances in the digital field, the number of images being generated every day grows exponentially. The field of histopathology is no exception and witnesses the emergence of an increasing number of Whole Slide Images that need to be treated, analyzed and diagnosed. One way to facilitate the diagnostic process is by comparing a particular case with other similar cases. This implies, first, the accessibility to other cases, as well as the ability to retrieve the most useful ones, i.e., the most similar cases. To achieve the latter goal, the technique of Content-Based Image Retrieval (CBIR) was conceived. CBIR involves retrieving the most similar images in a database to a given query image.

The goal of this thesis is to study the different elements that compose a CBIR framework and the options available for them, with a specific focus on the feature extraction part of the framework. It offers an open-source implementation that allows the combination of the researched options to create a fully operational CBIR framework. It provides both supervised and self-supervised models as a way to accommodate all situations and datasets.

All feature extraction models are trained on a single dataset containing over 600,000 histopathological images and evaluated on approximately 200,000 different images from the same dataset. Extensive experiments are conducted to analyze the resilience of the frameworks in different situations, such as when dealing with new data or handling class imbalance.

While the supervised models have displayed great results and the self-supervised methods have demonstrated great potential, the scope of what could have been achieved is limited by the lack of evaluation by trained pathologists and by the few remaining untested combinations.

Acknowledgements

This thesis represents the end of my five years of study at the University of Liège in Computer engineering. This final work would not have been possible without the guidance and knowledge obtained from my professors during those years. I would not have arrived where I am today without the help and support of multiple persons.

I would like to first express my deepest gratitude to my supervisors, Pr. P. Geurts and Dr. R. Marée, for their guidance, support, and valuable insights throughout the entire process. I would not have been able to produce such a work without their help and advice. I am especially grateful for the time they spent at reading and commenting my drafts as I know that they have really busy schedules.

I would also like to thank M. Defraire for the access to his implementation as well as for his help and answers to my many questions.

Finally, I would like to thank my family and friends for their unwavering support and understanding. A special mention to my mom for proofreading my drafts, despite the subject not being in her main area of expertise, and to my dad for his help regarding the access to the infrastructures needed for this work.

I hope that this work will turn out to be useful for future research on the subject and that one day it will lead to a great tool in assisting pathologists in their diagnosis.

Axelle Schyns

Contents

Abbreviations

AE	AutoEncoders
AI	Artificial Intelligence
AMDIM	Augmented Multiscale Deep InfoMax
CBIR	Content-based Image Retrieval
(A)(N)C(L)	(Augmented) (Non) Contrastive (Learning)
CNN	Convolutional Neural Network
CPC	Contrastive Predictive Coding
CT	Computed Tomography
CV	Computer Vision
CvT	Convolutional Vision Transformer
db	DataBase
DeiT	Data-Efficient Image Transformer
DL	Deep Learning
DML	Deep Metric Learning
DR	Deep Ranking
FAISS	Facebook AI Similarity Search

FC	Fully Connected
ids	indexes
ILSVRC	Imagenet Large Scale Visual Recognition Challenge
ML	Machine Learning
MRI	Magnetic Resonance Imaging
MSE	Mean Squared Error
NCA	Neighborhood Component Analysis
NLP	Natural Language Processing
ResNet	Residual Network
Std	Standard deviation
TCGA	The Cancer Genome Atlas
t-SNE	t-distributed Stochastic Neighbor Embedding
VGG	Visual Geometry Group
ViT	Vision Transformer
WSI	Whole Slide Image

Chapter 1

Introduction

1.1 Context and Motivation

In recent years, the digital world has experienced tremendous progress, leading to the digitization of various fields, including the medical field. This development has given rise to a new sub-field called Digital Health, which is gaining increasing attention every day. Digital Health comprises several elements, ranging from patient access to their data through various web applications to the synchronization and data sharing among practitioners, hospitals, and laboratories, as well as the digitization of all medical information. Consequently, this digitization has resulted in an enormous amount of data, particularly in medical imaging. Cutting-edge advancements in medical technology, such as photon-counting CT scanners, multi-focus microscopes that generate detailed 3D images of cells, and MRI gloves that capture intricate movements of hand tendons, are providing unprecedented opportunities for the scientific and medical communities to acquire a wealth of images. These images, in turn, enable researchers to gain novel insights into diseases and injuries, paving the way for the development of effective remedies and solutions. As a result, healthcare and quality of life have the potential to be greatly enhanced for all.

Histopathology, one of the fields of research that involves the analysis of such medical images, focuses on the study of tissues (*histo*) to understand diseases (*patho*) and their impact. By examining microscopic images of various tissues, histopathologists can identify the presence of diseases and determine their severity. They can also identify common disease symptoms and use this knowledge to identify them in future scans, or they can uncover new symptoms or characteristics of a pathology. In this field, digitization plays a pivotal role as it enables the acquisition of tissue images. Having a larger dataset enhances confidence in research findings, as it allows for the identification of multiple occurrences of a phenomenon. Thus, digital pathology on Whole Slide Images (WSIs) holds significant promise for advancing the understanding of diseases and their effects on tissues, leading to more accurate diagnoses and improved patient care.

However, before getting to such medical advances, there are two major obstacles that complicate the analysis of the images.

First, these images, scans, videos,... need some work put into them. They need to be processed and analyzed such that they can be easily interpreted and used for more advanced purposes. To analyze them efficiently and in detail, tools must be designed. Those tools must be able to handle large amounts of various data. Indeed, the images all come in various types and aspects, different sizes and colors. They also represent different elements and organs. This variety comes primarily from their objective. A scan of a liver is not taken for the same purpose as a scan of a lung for example. It also comes from the way the images were obtained, from

the specificity of the machine or protocol used to take them. In this case, we usually refer to the versions as modalities. This variety must be taken into account in the tools used to analyze a specific case or disease. For such reasons, the focus has shifted towards Artificial intelligence (AI) and Machine Learning (ML) powered tools as those two domains have shown great promise in scenarios handling large, varied datasets.

Second, the accessibility and sharing of the images pose significant challenges. Laboratories and research centers are often geographically dispersed and may not always collaborate or share resources and data. While some reluctance to cooperate with other entities may be a factor, in many cases, the lack of infrastructure for facilitating such cooperation is the main issue, due to the large size, up to several GB, of the images, among other factors. Consequently, there is a limited level of globalization in the field, which could impede the detection of abnormalities or identification of specific diseases, particularly in the case of rare diseases or phenomena. As the few instances of such conditions may not be regrouped or accessible through a single, well-known place, it further complicates the process.

Hence, solutions to those obstacles must be designed. A first solution has been offered by the Cytomine research team[1] at the University of Liege with the deployment of a collaborative, open-source web-platform in 2010. This project offers several functionalities (Figure 1.1 (a) - Credits: Cytomine R&D website). It offers a workplace where WSIs can be uploaded, organized, and stored. Those WSIs can be shared with other users and collaboratively analyzed through the use of annotations (Figure 1.1 (b) - Credits: Cytomine R&D user guide). It also offers several machine learning algorithms for vision processing, such as tissue and landmarks recognition. The platform[2] is of interest for practitioners but also for students and teachers for a more educational purpose. Recently, the Cytomine R&D team entered the BigPicture EU IMI project. This project has for goal to collect millions of WSIs in order to obtain the biggest histopathological dataset existing. This dataset will then be used to develop AI and ML tools. In addition to the existing tools, the Cytomine R&D team continues developing new software modules to increase the possibilities of image analysis and further deepen the research on pathologies.

Concurrently with the advancement of digitization, another rapidly evolving domain is that of artificial intelligence (AI) and machine learning (ML), particularly in the area of deep learning (DL). Deep learning has gained significant momentum in recent years due to its remarkable achievements across various fields. Its adaptability and ability to efficiently handle vast amounts of data, while also learning directly from it, make it a favored technology for visual processing and analysis. In fact, deep learning has already demonstrated remarkable accomplishments in the field of computer vision (e.g. ImageNet classification tasks as in [Krizhevsky et al., 2017] or [He et al., 2015]). As such, it is considered one of the most promising options for developing tools and solutions in the medical imaging field, of interest here.

1.2 Problem statement

This master thesis's aim is to develop a novel tool for medical image analysis using deep learning, with the ultimate goal of assisting in future diagnostic processes. The tool under investigation and design throughout the remainder of this work is referred to as a "Content-Based Image Retrieval (CBIR) system".

CBIR consists in retrieving in a database of images the ones that are the most similar to a given image, called the query. It is composed of several steps that can be independent or not, depending on the elements chosen to compose it. The two main parts of the construction of a CBIR framework are:

[1] Source: Cytomine R&D website

[2] Described on the Cytomine website

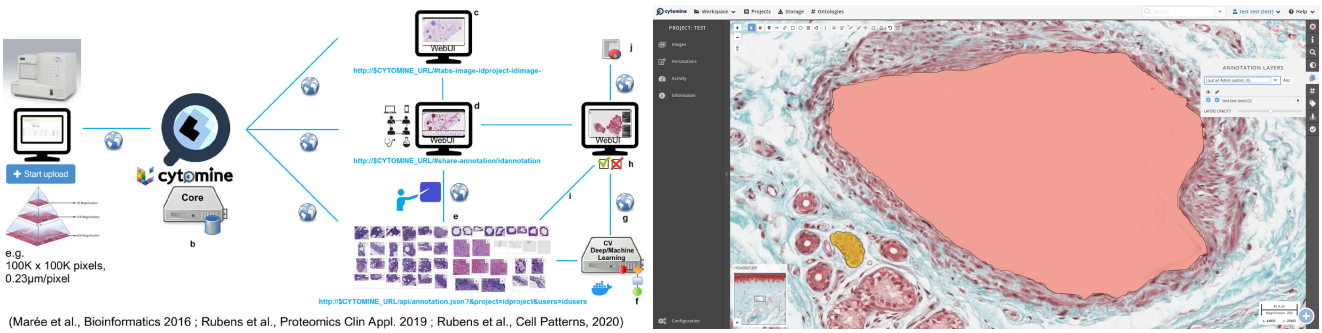

Figure 1.1: (a) Structure of the Cytomine project. (b) WSI with annotation in Cytomine Workplace

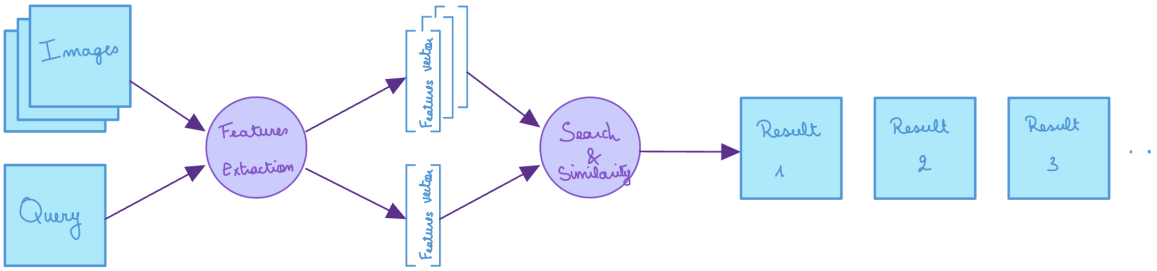

Figure 1.2: Functioning of a CBIR framework

- The extraction of a representation[3] of the images.
- The computation of the similarity between two vectors and the search algorithm used to find the most similar vectors. This also includes the indexing in the database, as the indexing will play a role in the way the vectors are accessed during the search.

A schematic illustrating the functioning of a CBIR framework is depicted in Figure 1.2.

This work primarily focuses on the first part, as it is where using deep learning is the most relevant. Nevertheless, it also provides a description of a solution for the second part.

For the design of the first part, both the architectures of the deep learning models and the training approaches are aspects to consider and are thus thoroughly investigated. While supervised models are presented, our study also investigates self-supervised and unsupervised models. This choice is motivated by two factors related to the field and task at hand.
Firstly, the goal of the CBIR framework is to retrieve images with similar content to the query, and thus it seems more relevant to train the model using only such images, rather than relying on labels that may not accurately describe the content of the images. For instance, an image labeled as "flower" does not provide sufficient information to accurately represent the content of the image. A flower can be of multiple colors or shapes, be in different backgrounds, etc. Two images labeled as "flower" may not have anything in common but this common designation.
Secondly, from a practical standpoint, data availability and cost are important considerations. There are typically more unlabeled images available compared to labeled images, which allows for more comprehensive training of unsupervised/self-supervised models as opposed to supervised models, which are limited by the availability of labeled data. Additionally, labeling all images in a large dataset is often impractical due to the time and cost associated with employing a trained

[3]In the rest of this work, it will also be called 'feature extraction' interchangeably.

pathologist for labeling. On top of it all, one future purpose of our investigation is to do region annotations in WSIs. This means that the WSIs will be cut into several pieces, pieces that will not have a label even if their WSI does.

Another aspect of this work is to make sure that the designed framework works well with digital pathology images. While many frameworks have been designed for natural images, fewer have been designed specifically for medical images. It is important to take it into account because these two types of images have different dominant characteristics which impact the elements composing the framework. The colors, shapes, and backgrounds of elements of medical images usually differ from those of natural images, as can be seen in Figure 1.3. There is also the need to take into account the different modalities under which a medical image has been taken, as explained earlier. All those differences make it important to verify the generality of the designed framework: if components working well for natural images are still proving effective with natural images and, overall, that it's able to produce good results when applied to medical images. It is also worth investigating the impact of pretraining on natural images, even if the end task involves a different type of image. Moreover, the kind of images of interest in this work, the WSIs, tend to be gigantic, up to several GB, which changes the way the data must be handled and fed to the networks. However, in the case of this thesis, the WSIs are actually patches (i.e. sub-images) and are, as such, of usual dimensions for images (Figure 1.4). Eventually, it is hoped that such

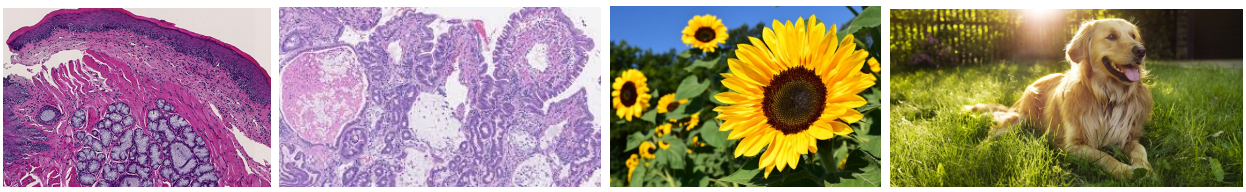

Figure 1.3: Medical images vs natural images

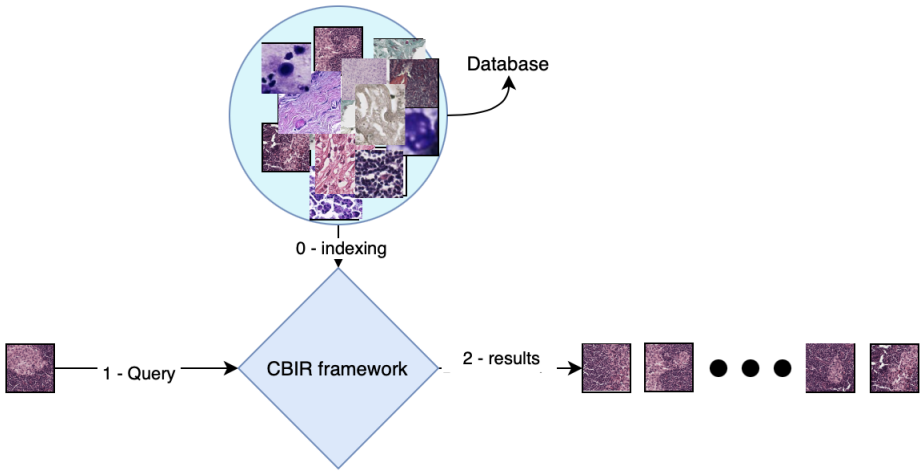

Figure 1.4: Patches retrieval process

a tool will help with the making of diagnosis and will fasten the disease identification process. A practitioner will only have to submit images of his case to receive similar images with their associated diagnosis. He will then be able to analyze those images and decide if his case fits the diagnosis assigned to the results. Similar pathologies, tissues would be available with one simple click.

This work is a continuation of a previous master thesis on the same subject by Stephan Defraire in 2021. This past thesis focused on supervised deep learning networks for the feature

extraction part, as well as on a method to prepare the dataset, specific to medical images. On the other hand, this current work has for ambition to explore methods developed since the time of publication of that past thesis and targets more specifically unsupervised methods.

1.3 Work organisation

This master thesis will follow the structure described right below.

- **Chapter 2: Theoretical Background**. This chapter presents the concepts used and investigated within the framework of this work. It starts with the presentation of the field of interest, namely *Computer Vision* (CV). Then, it discusses notions of *deep learning*, from the basis to more complex models. It finally ends with the methods used to train said models.
- **Chapter 3: Theory linked to CBIR**. This chapter introduces the notions of *Content-based Image retrieval* (CBIR) by presenting its evolution and particular methods/elements attached to it. Afterward, it continues on a more practical side with *Data Management* and *Similarity search* before ending up with the presentation of *complete frameworks* of CBIR and image representation systems.
- **Chapter 4: Dataset**. This chapter introduces and *describes the datasets* used in the training of the feature extraction models, as well as for the testing of the indexing and retrieval methods. It explains the division in classes of the main dataset and *its characteristics*, as well as the techniques used *to prepare the data*.
- **Chapter 5: Methodology & Process**. This chapter is drifting apart from the theory to focus on the more practical aspects of this work. It presents the different tested methods and the processes followed to investigate the impact of the different parameters. It first introduces the *backbone architectures* used for feature extraction. It then dives into the practicalities of *indexing, searching and retrieval*. To wrap up the CBIR process, it presents the protocols and measures that will be used to *evaluate the results* of the complete framework. It concludes with a quick description of *the implementation* (publicly available on the GitHub page of the author, `https://github.com/AxelleSchyns/cbir-tfe`).
- **Chapter 6: Results and Discussion**. This chapter presents *the results* obtained through the use of the *different techniques* for feature extraction, both quantitatively and qualitatively. It also describes the impact of the choice of *several parameters*.
- **Chapter 7: Limitations, extensions and conclusion**. This section delves into the limitations encountered in this study and proposes potential solutions for overcoming them in order to enhance the results obtained. Furthermore, it outlines possible directions for extending the model for other applications. It then wraps up this thesis by providing a quick summary of the tested methods and their results.
- The appendix contains several additional figures of interest.

Chapter 2

Theoretical Background

In this chapter, the theory behind the models and methods used in this work is presented in detail. The way those models and methods are applied is however introduced in Chapter 5.

2.1 Learning fashion

Our work uses models that rely on labels as well as models that do not use them.

The first method is known as supervised learning. It uses the labels of the data to evaluate how the model executes the task it is given.

The second way is known as unsupervised learning. Unsupervised learning is used to make a model learn without having access to the labels of the data. In specific conditions, a model is called self-supervised rather than unsupervised. Self-supervised learning is a subfield of unsupervised learning in that it works on data that do not have labels. There are plenty of definitions of self-supervised learning, making it more or less wide. In some sources[1], self-supervised learning is referred more as a predictive kind of learning or pretext learning, using the input data to predict future data. In others[2] , it relates to learning that first learns labels in an unsupervised fashion before using those generated labels to train for the rest of the task. The first definition is more related to NLP tasks, stating that vision tasks make it more complicated to define the predictive aspect.

It is globally accepted nevertheless that augmented contrastive learning (explained later) is considered as self-supervised learning. Indeed, contrastive learning relies on 'pseudo labels'. It learns by doing comparisons of the data, making the pseudo label 'similar' or 'dissimilar'. And those pseudo-labels are generated by the algorithm before the 'true' stage of learning, making it the pretext task mentioned in the first definition.

For the second method used in this thesis, the autoencoders, the situation is less clear. Some[3] state that as autoencoders work on the raw data, without any pre-task, it is unsupervised learning. Furthermore, most sources only consider as self-supervised learning two methods, namely Augmented contrastive and non-contrastive learning. However, other sources[4] state that autoencoders work quite similarly to Contrastive learning, by comparing the reconstructed

[1]Source: Neptune AI - Self-Supervised Learning and Its Applications - Deval Shah, Abhishek Jha - 19th April 2023 and Facebook AI - Self-supervised learning: The dark matter of intelligence - 4th March 2021

[2]Papers with code - Self-Supervised Learning - 28th May 2023 and Analytics Steps - Self Supervised Learning - Types, Examples, and Applications - Pragya Soni - 20th October 2021

[3]Sources: v7lab - Autoencoders in Deep Learning: Tutorial & Use Cases [2023] - Hmrishav Bandyopadhyay

[4] Autoencoders and self-supervised learning - Jonathon Hare - University of Southampton

image to the original, making it self-supervised. As it also works on comparisons, it will also be considered as self-supervised learning in our work.

2.2 Computer vision

Before going to the architectures, let us start with a quick word on the field to which this work is attached.

Computer vision comprises all the algorithms, methods, and computer science components that deal with images or other visual elements. For IBM, Computer vision is defined as "*a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos, and other visual inputs — and take actions or make recommendations based on that information. If AI enables computers to think, computer vision enables them to see, observe and understand.*" [5].

It is a very important and large field. As such, it has been decomposed into several sub-fields that are more focused on one particular task of vision like Classification, Segmentation, Representation, Object recognition, Retrieval,... Those sub-fields frequently overlap, e.g. **Generation** is frequently associated with **Representation** or **Object detection** with **Segmentation**. In this work, the main sub-field of interest is **Retrieval**, but **Representation** is also of accrued interest for the first part of the framework, feature extraction.

CV is a field in constant evolution and the techniques used in its different applications have widely evolved and changed over the years, alongside its objectives. The first use of computer vision, as per Deep North and others, was for classification. Classification has been the center of CV for a long time before other applications stemming from it appeared[6]. Nowadays, each sub-field has been shown interest and has seen models developed to cater to the problems it raises. Similarly to the evolution of the applications, the methods have evolved too. The initial methods would now be considered low-level as they consist of algorithms to detect specific features such as edges, lines and other shapes using Hough Transforms or filtering[7]. Then neural networks were introduced and quickly became the reference in vision. Their types also changed along the years, starting with MLP, which quickly proved to be lacking for such structured data, to continue with CNNs, built for such tasks and that are still today the reference in CV. Transformers have also started recently to be used for vision tasks, derived from their original purpose which was Natural Language Processing (NLP), challenging CNNs. In each type, several architectures were also devised, each proposing solutions to problems noticed in the previous ones or suggesting improvements to increase the results, such as more layers or different activation functions, adding again to the evolution of the field.

Another area that shows the evolution in CV, but also in AI and DP in general, is the way of training the models. Models were first trained from scratch for a specific end goal. Then, fine-tuning was introduced and gained progressively in popularity, first in machine learning algorithms then for deep learning language models [Radford et al., 2018] and vision models [Simonyan and Zisserman, 2014]. In the latter case, it became especially popular due to its association to transfer learning and the ImageNet dataset, which was, and still is, the biggest dataset available for training vision models. Fine-tuning and transfer learning have proven useful given the deep relations between the different sub-fields of vision that allow models designed for one task to be used on another one relatively easily. This is also what is done in this thesis, with models made for classification whose last layer was modified to make them usable for feature

[5] From IBM website - 'What is computer vision' - 19/05/23.
[6] As per Viso.ai
[7] TrendSkout

extraction.
Another evolution happened regarding the training, related to the data fed to the models. Data were originally labeled and the methods were trained in a supervised way. It is closely related to the primary goal at the time, Classification, which requires an indication of the categories onto which to classify the images. Eventually, unsupervised and self-supervised learning started to gain in popularity as a way of not having to rely on labels and focus only on the content, which also lead to getting access to more data. This last evolution is also accounted for in this work, with the first models explored being trained in a supervised fashion while the last are self-supervised.

2.3 Basis of Deep learning - Architectures

This second section mainly focuses on the first part of the framework as described in Chapter 1, and more specifically on the architectures used for feature extraction. The basis behind the architectures is based on the lecture notes from [Louppe, 2022] while the chosen architectures are described based on their related paper. Those architectures are split into three categories:

- CNNs: selection of the most popular CNNs designed for classification but used for feature extraction after modification.
- Transformers: use of basic transformer model for vision and of other improved and frequently used models.
- Autoencoders: use of basic AE and of variational AE.

2.3.1 Convolutional Neural Networks

CNNs have been designed specifically for vision. Inspired by the human visual system, they are conceived such that they cater to three important characteristics of visual data.

- Locality. This principle enforces the particularity of the pixels of an image being linked to one another when close, and independent of pixels far from them.
- Invariance to translation. This principle forces the models to treat similarly the pixels, regardless of their position in the image. In particular, the same element at two different locations must be represented similarly.
- Hierarchical compositionality (Figure 2.1). This principle is related to the extraction from the data of features of different importance depending on the layer of the model extracting them.

In order for a model to have those characteristics, convolutions are used. Convolutions consist in applying a kernel u to the input image or input feature map x such that the element-wise multiplication between the elements of the kernels and the elements of the image that match it is performed, and all the results summed up (Figure 2.2).

Mathematically:

$$\mathbf{o}_{j,i} = \mathbf{b}_{j,i} + \sum_{c=0}^{C-1} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{c,n+j,m+i} \mathbf{u}_{c,n,m}$$

with b the offset and (h, w, C) the dimension of the kernel, C being also the capacity of the image. It can be inferred from this description that the convolutions lead to the appearance of

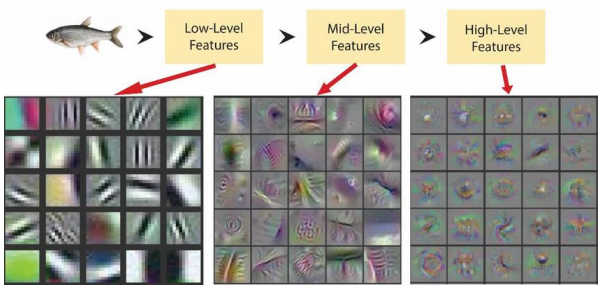

Figure 2.1: Hierarchical Compositionality - Fish example. Credits: [Siddiqui et al., 2017]

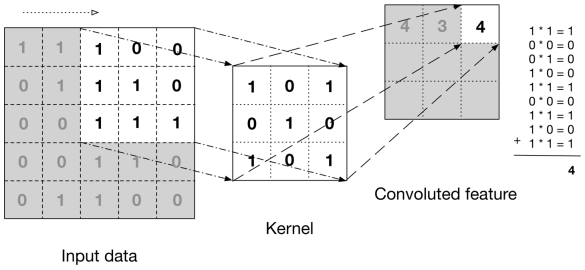

Figure 2.2: Convolution operation. *Credits:*Analytics Vidhya

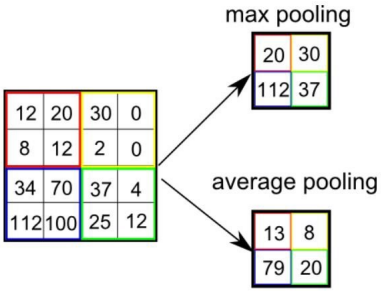

Figure 2.3: Pooling example. Credits: Analytics Vidhya

the mentioned characteristics. They enforce the first principle by using the pixels around the pixel of interest in the computation of the output. It respects the second by using the same kernel on all pixels, and just sliding it from one to another. Finally, the hierarchy principle is inherited from the multiple applications of the convolutions, leading to several different feature maps.

In combination with the convolution layers, CNNs also use pooling layers as a means to conserve the structure of the input while reducing its dimensions. The application of pooling is quite similar to a convolution: a kernel is slid onto the input elements and element-wise multiplications are performed, the results are then aggregated given a specific operation. The difference is that the kernel is composed of 1s and the operation is a mean if *average* pooling is used and consists of taking the maximum of the results if *maximum* pooling is used (Figure 2.3).

The usual structure of a CNN is obtained by combining a certain number of convolutional and pooling layers, then followed by fully connected layers.

Such a structure was proposed in [Lecun et al., 1998] and then, years later, another model was introduced by [Krizhevsky et al., 2017] that durably impacted the use of CNNS in CV. It is that model that led to the CNN models used in this work, presented by chronological order.

VGG: Visual Geometry Group

Presented in [Simonyan and Zisserman, 2014], this network is known for its important depth, greater than those of any previous CNNs. VGGs depend on a given number of layers. In this work, VGG16 (with 16 layers) and VGG11 (with 11 layers) will be used for feature extraction.

Let us detail the architecture of VGG16. It is trained for image classification on the ImageNet dataset, hence for a 1000 categories problem (Figure 2.4). 13 of those 16 layers are convolutional layers with ReLUs as activation functions. They are positioned as two blocks of two convolutional layers and one maximum pooling, followed by three blocks composed of three convolutional layers

and one maximum pooling layer. Finally, two 4096 channels fully connected layers and one 1000 channels fully connected with ReLUs are placed, followed by a softmax.

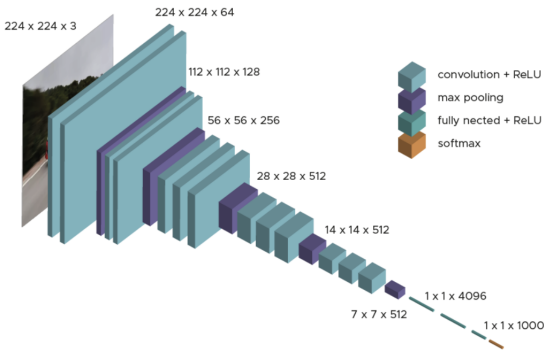

Figure 2.4: VGG16 architecture.
Credit: DataScienceTttst

The particularities of VGG, besides its great depth, are the small size of its kernels in the convolution layers (3x3) and the use of ReLUs as in [Krizhevsky et al., 2017].
VGG16 is a huge network with around 138 million parameters, despite the use of such small kernels and the absence of Local Response Normalization, found to increase the memory requirement in this case.

ResNet: Residual Network

The ResNet models were introduced in [He et al., 2015]. The innovation of this network, as indicated by its name, resides in the residual blocks composing it. Those blocks allow building higher depth networks while not increasing too much their complexity and obtaining good results.

They have been designed as a solution to an issue noticed on the previous CNNs, which is called *degradation*. This phenomenon, attributed to the depth of the networks, causes the accuracy to fall with the augmentation of the number of layers and is not due to the overfitting of the network. This degradation phenomenon is due to the difficulty of the networks to fit identity mappings, thus leading to poor results when such mappings happen to be the optimal solution. The authors specified that while they know that identity mappings rarely happen to be the optimal solution in real cases, they have shown to be close enough to real solutions such that if a network can fit them, the network is more likely to be able to fit the optimal solution.

To enable the network to approximate such mappings, the authors introduced residual functions $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$, which become the new optimization objective of the layers, with $\mathcal{H}$ the original mapping.

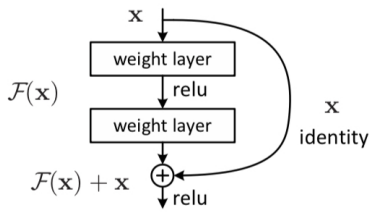

Figure 2.5: 'Residual learning: a building block'. Credits: [He et al., 2015]

To compensate for this change, the building blocks are defined as $\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$. The first term is obtained through the usual layers operations. It is the second term that makes the particularity of the residual blocks. This term is obtained through the use of *shortcut connections*. Those connections are identity mappings such that its inputs are not fed to any layers but simply added to what is output by those layers. It skips one or more layers (Figure 2.5).

Note that the rest of the block is inspired by the VGG blocks, with mostly 3x3 convolutional layers and ReLUs. It is however less complex/big, especially given the fact that the skip connections do not add complexity or parameters.

Several ResNet architectures were developed, with a different number of layers. This work uses the architecture ResNet-50 and the architecture ResNet-18.

DenseNet and KimiaNet

The DenseNet model was proposed in [Huang et al., 2018] to help with the vanishing of gradients that has become a problem with the increasing depths of the new CNNs.

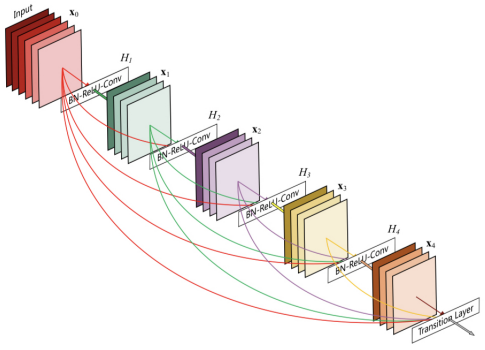

Figure 2.6: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature maps as input.
Credit: [Huang et al., 2018]

The offered solution is to connect each layer to all the other following layers, hence the term 'dense'. Instead of having only the previous layer's feature map as input, the layers take as inputs the output of each of the layers preceding it. Those outputs are not summed or summarized in any way but simply concatenated (Figure 2.6).
The complete network is composed of dense blocks intertwined with convolution and pooling layers, to finally end with a pooling layer and a softmax. A dense block is a block using the previously described connections as well as batch normalization, ReLU and a 3x3 convolution.

Density also leads to a decrease in the number of parameters. Indeed, the use of all the previous layers' outputs leads to *feature reuse*. Unlike in other CNNs where the weights change completely from one layer to another and each output must contain information from the previous one (i.e. state), the DenseNet uses smaller filters which make each layer output a smaller feature map. This is possible because all the feature maps still arrive at the last layer, so they don't need to be contained in another layer's feature map. In this work, the DenseNet-121 architecture is used, in the same way as the previous CNNs models.

Regarding KimiaNet, it is based on a DenseNet and is described in [Riasatian et al., 2021]. The particularity of this network does not reside in its architecture, as it is basically a DenseNet composed of 4 dense blocks. What sets it apart from a 'regular' DenseNet is the training method and the training data.

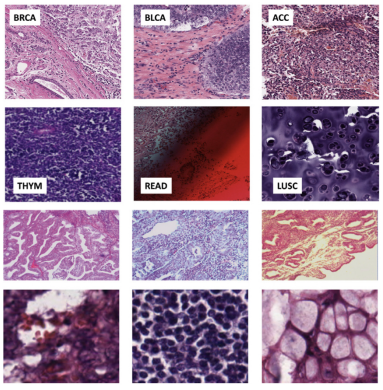

Figure 2.7: Samples from the TCGA dataset (1st and 2nd row) and endometrial cancer (3rd) and colorectal cancer (4th). Credits: [Riasatian et al., 2021]

KimiaNet was trained for the specific purpose of being used for *histopathology* and, as such, has been *finetuned* on pathology images (TCGA dataset - Figure 2.7).
7126 WSIs were used for training, split in patches for a total number of 242 202 patches of size 1000x1000 pixels, at magnification 20. Those WSIs and patches were carefully selected and obtained. For the WSIs, frozen sections were discarded as they might have led to confusion from the model. Furthermore, groups were created to sort the WSIs and the smallest groups (less than 20 WSIs) were removed from the training data. The patches are obtained from the construction of a mosaic of the original WSI by clustering and thresholding on their cellularity.

The authors report better accuracies/results, both in search and in classification, of their model compared to a DenseNet pre-trained on ImageNet when the test data is composed of

histopathological images. The specialization of KimiaNet on histopathology is the reason why it was chosen for this thesis. What's more, unlike the previous CNNs, KimiaNet was designed for representation learning rather than classification, which makes it even better suited for this thesis goal.

EfficientNet

EfficientNet models were introduced in [Tan and Le, 2020] as a way to optimally and efficiently scale up CNNs. This was proposed following the trend of CNNs to have several versions, with more or fewer layers depending on the available resources. These versions usually only differ on one parameter between three possibilities: width - depth - resolution. The authors looked into scaling up a model using the three dimensions instead of only one to reach better performances while keeping the model reasonably large (Figure 2.8). They arrived at a set of coefficients by which those three parameters should be multiplied given the available power. On top of those

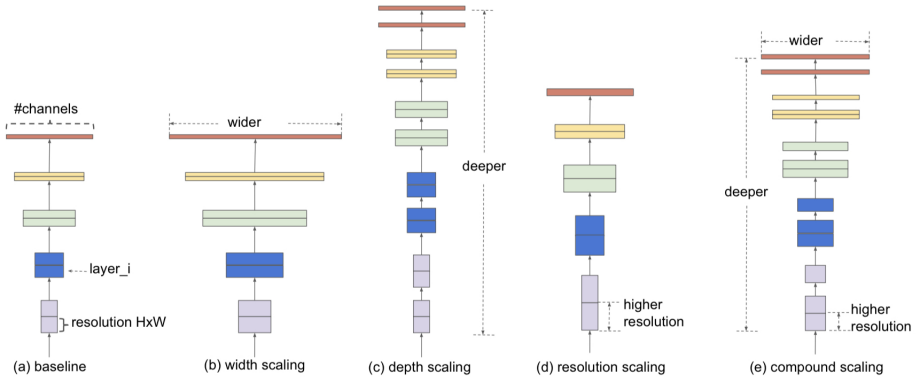

Figure 2.8: 'Model Scaling. (a) is a baseline network example; (b)-(d) are conventional scalings that only increase one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.' Credits: [Tan and Le, 2020]

coefficients, which can be applied to any CNN, the authors created their own series of models, the EfficientNets. Those models have an architecture based on a model called MnastNet, presented in [Tan et al., 2019]. In this work, the EfficientNetB0 is used, in the same fashion as the previous architectures.

2.3.2 Transformers

The transformer architecture relies on the attention mechanism, introduced in [Bahdanau et al., 2016]. The attention mechanism was first designed for NLP tasks, more specifically for machine translation. It was proposed in order to remove the bottleneck that resulted from using a single fixed-length vector as representation for a sequence, like it was done in RNN, and then use that sole vector to get the output. Instead, the input tokens are directly used to generate the output in a more dynamic way.

A context vector is computed based on weights indicating how much each input token must be taken into account for the current element of the output. Formally:

$$\mathbf{y} = \sum_{i=1}^{m} softmax_i(a(\mathbf{q}, \mathbf{K}_i; \theta))\mathbf{V}_i$$

where $\mathbf{q}$ is the query or context, $\mathbf{K}$ the key tensor and $\mathbf{V}$ the value tensor. a is a score function that can be defined in multiple ways, such as the *additive attention* or also the *scaled dot-product*.

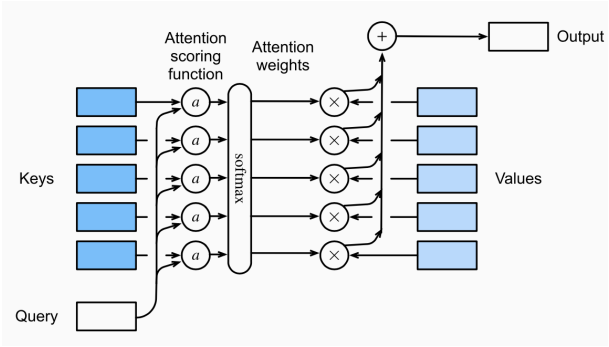

Figure 2.9: Attention mechanism. Credits: Dive into Deep Learning 11.3

The query, value, and keys, are usually obtained linearly from the inputs and Weights matrices with:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q^T$$

$$\mathbf{K} = \mathbf{X}'\mathbf{W}_k^T$$

$$\mathbf{V} = \mathbf{X}'\mathbf{W}_v^T$$

In those formulas, different inputs are used for the query and for the other two. If the same input is used for all three elements, then the mechanism becomes *self-attention*.

While attention was first simply added to existing models such as RNNs and CNNs, the invention of transformers changed the way it is used. Transformers were presented in [Vaswani et al., 2017]. The model is built following an encoder-decoder structure and is said to be autoregressive (it uses at each step all previously computed symbols).

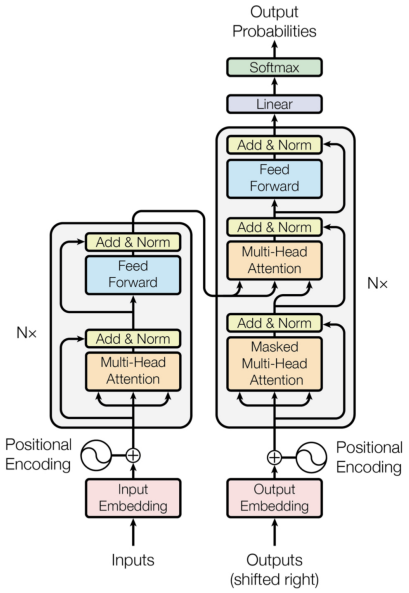

Figure 2.10: Transformer architecture. Credits: [Vaswani et al., 2017]

Both the encoder and the decoder are made using 6 times the same layers (Figure 2.10, N = 6). The encoder's layers are composed of a multi-head attention sub-layer and a feed-forward sub-layer, both with normalization afterward. Residual connections are also present around the two sub-layers. For the decoder, its layers contain as sub-layers two multi-head attention layers and one fully connected feed-forward, each with normalization and residual connections. The first multi-head is made on the outputs after they have been shifted and put through a positional encoding mechanism. The second multi-head is made both on the results of the previous one and on the outputs of the encoder.
The model is ended by a linear FC layer and a softmax. Positional encoding is also realized after the embedding of the input. It adds information about the order of the elements, as the layers of the model can't get it themselves.

Regarding the multi-head attention layers, it takes three inputs: the key, value and query. Those inputs are linearly projected such that different versions are obtained and used to compute a scaled Dot-Product Attention. The 8 results are concatenated to finally be linearly projected once more. Mathematically, the multi-head attention process is defined as follows:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, head_2, ..., head_8) \cdot W^O$$

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Vision Transformer - ViT

The transformers were at first designed for machine translation tasks. They were further developed for global NLP tasks before slowly being adapted to vision. However, most works kept using a CNN architecture as backbone and simply added attention on top, until the paper by [Dosovitskiy et al., 2020]. The authors of this work used a basic Transformer that they tried to modify as little as possible while still making it work on images.

The most important modification concerns the way the images are fed to the Transformer. For NLP tasks, inputs are split into tokens, then fed one by one to the network as a sequence. A similar split is performed on the images, with patches of dimension $P \times P$ being extracted from the original images. Those patches are then passed through a linear projection in order to make them into vectors of constant size, as required by the Transformer architecture. Those vectors form a sequence, to which is added at the end an indication about the class of the image from which they were obtained. Finally, the final embeddings are obtained by adding to the current sequence 1D position embeddings that inform of the order of the patch in the sequence of all patches, similarly to the way information about the place of a token in the sentence would be furnished in NLP tasks.

Unlike CNNs, built for vision, Transformers lack some of the characteristics specific to vision tasks (called *inductive bias*) such as the locality principle or the translation invariance. Despite this, the authors have also decided to use as little as possible the 2D structure of the input images, and as such, to learn image characteristics from the input data only, during training. Their model still shows great results, outperforming some state-of-the-art CNNs, when trained on enough data. Indeed, as a consequence of this absence of inductive bias, ViTs need a lot of data to be able to reach good results.

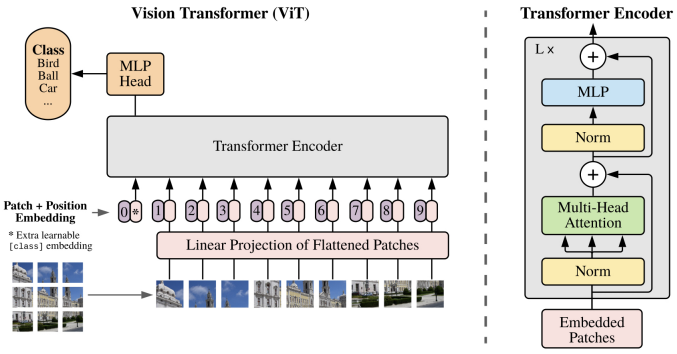

Figure 2.11: ViT model. Credits: [Dosovitskiy et al., 2020]

DeiT

DeiT was proposed in [Touvron et al., 2021] as a way to mitigate the issue noticed in ViT: the need for a very large amount of data. DeiT models are designed such that their training is efficient, they don't require too much data (Data-Efficient) and they have a reasonable amount of parameters.

The particularity of DeiT, that differentiates it from ViT, is the way it is trained. It uses a technique that the authors have called *distillation through attention*. Distillation consists in using a network as a teacher to train another network, the student. This allows the student model to get insights about a task from the teacher while keeping a smaller dimension/complexity than the teacher. In this paper, two settings are proposed, with two choices each. The first

setting is the 'strength' of the distillation, with soft or hard distillation. Soft distillation uses the probability distribution of the teacher output and compares it with the probability distribution of the student using the KL divergence. Hard distillation takes the final prediction of the teacher as the input true label and uses the Cross-entropy loss on it. Both still use the 'real' true label with Cross-entropy and combine their own computation with it.

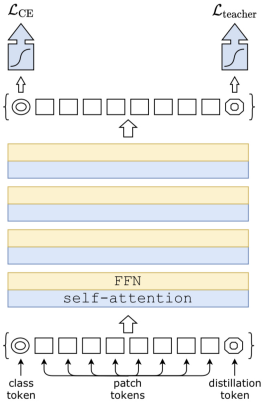

Figure 2.12: Distillation token process. Credits: [Touvron et al., 2021]

The second setting is the application of the distillation with the choice between a classical distillation and a new one, called *Distillation token*. It consists in appending to the class token and patches that are to be fed to the model an additional element, the distillation token. The distillation token will interact with all the patches during the passage through the network. It could be think that it would lead to the same result as the class token as they both represent the label but the class token's objective is to match to the true label while the distillation token objective's is to match to the output of the teacher network. A classical distillation would only use one token that it would match to the teacher output.

DeiT is also used as a feature extractor in this thesis, with its last layer removed and replaced by one fitting the number of features wanted.

2.3.3 From CNN to Transformers

Both CNNs and transformers have proven good at vision tasks. It is based on that principle that the authors of this paper [Wu et al., 2021] have created CvT. CvT is based on ViT but with convolutional elements added. Those convolutional elements are added at two positions, after breaking the original Transformer architecture into several parts to form a *hierarchy*.

The first position is at the very beginning, when splitting the images into tokens. The images are split into tokens which are shaped into a 2D structure such that a 2D convolution can be applied to them. Besides giving information on the spatial disposition of the tokens as well as taking into account the locality component, this operation reduces the size of the input by reducing the number of tokens.

The second position is inside the Transformer block, before the tokens are sent to the Multi-Head Attention layer. In the original transformer, linear projections are used to obtain the Query, Keys and Values from the input. In CvT, those linear projections are replaced by *depth-wise separable convolutional* projections. These new projections allow once again to 'encode' local spatial information into the model.

2.3.4 Auto-encoders

The last type of architecture used for feature extraction is the auto-encoder. The auto-encoder is a useful technique to generate new data or learn how to best represent the data fed to it. It does so by following the principle contained in the citation "What I do not create, I do not understand" by Richard Feynman. It tries to recreate the original input from an intermediate

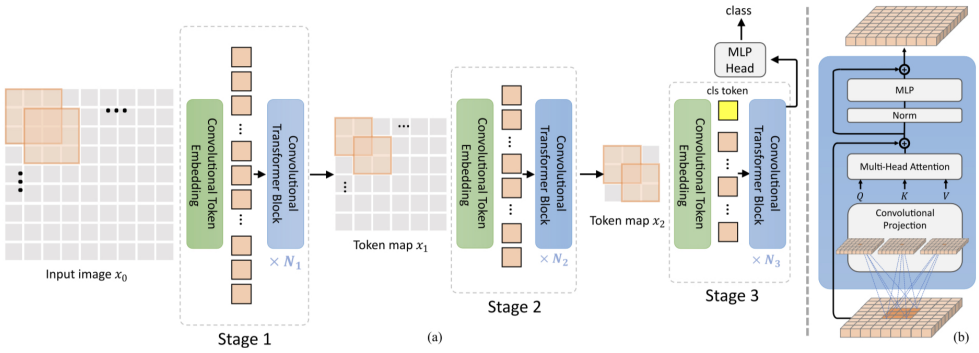

Figure 2.13: (a) CvT architecture (b) Convolutional Transformer Block.
Credits: [Wu et al., 2021]

representation and learns by comparing its creation to the original. It may be used for plenty of different tasks, from dimensionality reduction to data generation. As such, several different types of AE were designed: variational, denoising, contractive,...

The basic AE is composed of two components. The first one is the encoder that maps the input to the intermediate representation in another space. The second is the decoder that makes the opposite, it re-maps the intermediate representation to the original space. For deep AE, those two elements consist of neural networks, with usually the decoder being built as the inverse of the encoder. The AE is trained using a reconstruction loss. Such a loss has for objective to minimize the difference between the reconstructed image and the original. For example, denoting by $||.||$ the Euclidean norm throughout this thesis, the following loss

$$E_{\mathbf{x}\sim p(\mathbf{x})}[||\mathbf{x} - g \circ f(\mathbf{x})||^2]$$

relies on the squared Euclidean distance between the reconstructed image and the original one.

The variational AE was proposed in [Kingma and Welling, 2022]. It tries to approximate the posterior distribution of the input data which is often intractable. It does so by turning the original inference formula into an optimization problem, and thus by trying to find the best parameter to fit the real distribution using the KL divergence (*variational inference)*. Formally,

$$\nu^* = \arg\max_\nu E_{q(\mathbf{z}|\mathbf{x};\nu)}[\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x};\nu)]$$

For the architecture of the VAE, it translates to the use of two distinct networks. One network, called inference network NN_ϕ, is used as an encoder to approximate the parameters of the posterior $q(\mathbf{z}|\mathbf{x};\phi)$ given the input data $\mathbf{x}$. The second network, called generative network NN_θ, approximates the parameters of the likelihood $p(\mathbf{x}|\mathbf{z};\theta)$ given the input $\mathbf{z}$, sampled from the approximated posterior.

Three different implementations of an AE are used as feature extractors and are described in more details in Chapter 5.

2.4 Methods of interest

Feature extraction/image representation is a really specific task in DL and has led to the establishment of concepts centered around finding the best vectors to represent an image. Unlike in classification where probability distributions are used to compare the true labels to the predicted ones, those concepts rely on distance measures and rankings to evaluate the quality of a

representation. Therefore, those concepts were selected for the training of the feature extractor component of the framework, instead of a more classic way of supervised training. The first three following sections describe complete methods to train one of the architectures presented in Sections 2.3.1, 2.3.2 and 2.3.3. All three sections are supervised training methods, except for the the subsection *Augmentive Contrastive Learning* of the last section.

2.4.1 Deep Metric Learning

Basis

Deep metric learning (DML) is a method to train deep neural models to represent the input data such that similar data have similar representations while dissimilar data have different representations. As indicated by the name, it learns embeddings of the input images by using a distance metric to quantify the (dis)similarity between two representations. It is directly based on the pixels of the input images and aims to attribute big distances (in a general sense) to the embeddings of dissimilar points and small distances to the embeddings of similar points.

It is a method that works with all types of training: supervised using the labels to determine the similarity, semi-supervised where only the similarity between the data points is specified, and unsupervised where nothing is known about the data. It has several applications[8]: data visualization (t-SNE [van der Maaten and Hinton, 2008]), classification (k-nearest neighbor or [Lu et al., 2015]), clustering ([Nguyen et al., 2020]), face verification ([An et al., 2021]), retrieval ([Zhong et al., 2021],... and has been the subject of countless publications ([Kaya and Bilge, 2019]). It is the 'default' concept used to train the feature extractors for a retrieval framework as CBIR's whole concept is based on image similarity and how to display images the closest to a query image.

DML is characterized by three elements[9]:

- A feature extraction neural network
- A sampling strategy
- A similarity function or distance-based loss

Multiple components have been designed for each of those categories, with countless papers fitting DML to their specific application or situation by adding a new element to the long list of already existing possibilities. On top of that, DML has led to the establishment of 'sub concepts' which restrict the choices in each category to a certain type of elements. The two next sections, 2.4.2 and 2.4.3, present two of them: Deep Ranking (DR) and Contrastive learning (CL). While they are part of DML, they will be discussed separately. Consequently, in the following, when the term DML is used, it will **not** refer to those two specific sub-concepts or their elements but, instead, to what is covered in this section. The rest of this section describes the selected elements used as part of this work for the DML concept.

Feature extractor

The feature extraction network can be any neural network that leads to embeddings of the wanted size. For this work, it consists of the architectures described in 2.3.1 (ResNet - DenseNet - EfficientNet - KimiaNet), 2.3.2 (ViT - DeiT) and 2.3.3 (CvT).

[8]Credits: 'Similarity and Distance Metric Learning with applications to computer vision'by Aurélien Bellet and Matthieu Cord at the ECML/PKDD 2015 and [Lu et al., 2017]

[9]Source: 'A Beginners Guide to Deep Metric Learning' - Vijaysinh Lendave - November 6th, 2021

Sampling strategy

The sampling strategy consists in creating batches of input data that will be processed together by the model. Those batches are created such that the chosen distance metric/loss has all the information and elements it needs, such as which inputs are considered similar or not. Several strategies have been devised in order to create the most informative batches possible. The sampling strategy is directly linked with the loss as different losses need different types of batches. Hence, they will be presented alongside the losses right after.

Losses

DML uses what is called ranking losses[10]. Ranking losses have for purpose to rank the inputs given a specific task/target. With DML, they use a distance metric to make that ranking combined with information about how to compare those distances (which inputs are similar and which are not). They quantify the similarity of the input either to another input (margin-based losses, triplet, or pair losses) or to a proxy (proxy-based/classification loss)[11]. Margin-based losses are losses that basically subtract the embeddings of two inputs and compare the result to a defined margin. If the inputs are considered similar, then the result must be below the margin, else, it must be above. Proxy-based losses build representative vectors (= proxies) for each class. They will then try to get each input vector close to the proxy of the same class and far from the others. Proxies are randomly initiated and then updated during training. Triplet/pair-based losses will be discussed with DR and CL. One Margin loss and three Proxy-based losses are described in this Section.

Margin Loss

A basic Margin loss was proposed in [Wu et al., 2017] alongside a sampling strategy best suited to it. The loss is a mix of the Contrastive loss for efficiency and the triplet loss for flexibility. It tries to get positive samples close to one another with a tolerance (less strict than the contrastive loss) and separated from negative samples. Formally:
The distance measure is given by $D_{i,j} = ||f(x_i) - f(x_j)||$. The relationship between inputs is represented by $y_{i,j} = 1$ if x_i and x_j are similar and -1 otherwise.

The margin loss is thus defined as $l^{margin}(i,j) := (\alpha + y_{i,j}(D_{i,j} - \beta))_+$, with α a parameter controlling the value of the margin separating similar and dissimilar inputs and β a parameter setting the value of the boundary between the two. β is learned using the gradient $\partial_\beta l^{margin}(i,j) = -y_{i,j}1\{\alpha > y_{i,j}(\beta - D_{i,j})\}$

The margin loss needs the $y_{i,j}$ to be specified. As such, the sampling strategy must furnish the inputs as a list of triplets: one input (anchor), a similar sample from the same batch (positive), and a dissimilar one (negative). The sampling strategy discussed in the article is called *Distance Weighted Sampling*. It only focuses on the sampling of the negative elements as it has been shown that the selection of the negative part of the triplet has the most impact. The positive sample is simply a randomly selected sample from the batch, of the same class as the anchor.

To select the negative sample n to form a pair with a given anchor a, the strategy relies on probability-based drawing. The vectors embeddings are constrained to a k-dimensional unit sphere, which makes the distribution of the distances between two samples be $q(d) \propto d^{k-2}[1 - 0.25d^2]^{\frac{k-3}{2}}$. This distribution gives the weights used when sampling, with $Pr(n^* = n|a) \propto \min(\lambda, q^{-1}(D_{a,n}))$, with a the anchor, n a sample of the batch and n^* the sample considered to

[10]Source: 'Understanding Ranking Loss, Contrastive Loss, Margin Loss, Triplet Loss, Hinge Loss and all those confusing names' by Raul Gomez and 'Deep learning - loss functions' by Bernhard Kainz

[11]Source: [Kobs et al., 2021]

form the negative element of the pair. The parameter λ is used as a regularization term to avoid 'noisy samples'.

Basically, the furthest two samples are, the smallest $q^{-1}(D_{a,n})$ is and the least probable it is that those two samples form a pair. *Distance Weighted Sampling* makes it likelier to sample the hardest pairs (i.e. samples not that far from each other) while not constraining it.

ProxyNCA++

Introduced in [Teh et al., 2020], it improves the original ProxyNCA loss using six enhancements.

ProxyNCA is based on Neighborhood Component Analysis (NCA) whose objective is to minimize the probability that two data points of different classes be neighbors. Defining $p_{ij} = \frac{-||x_i - x_j||^2}{\sum_{k \neq i} -||x_i - x_k||^2}$ as the probability of x_i and x_j being neighbors, NCA's objective is given by

$$L_{NCA} = -\log\left(\frac{\sum_{j \in C_i} \exp(-||x_i - x_k||^2)}{\sum_{k \notin C_i} \exp(-||x_i - x_k||^2)}\right)$$

ProxyNCA adds proxies to that process in order to reduce the computation time. Instead of comparing each sample with all the others of the batch, it compares it to the proxy representing the class of the sample under consideration. The loss becomes

$$L_{proxyNCA} = -\log\left(\frac{\exp\left(-\left\|\frac{x_i}{||x_i||} - \frac{f(x_i)}{||f(x_i)||}\right\|^2\right)}{\sum_{f(z) \in Z} \exp\left(-\left\|\frac{x_i}{||x_i||} - \frac{f(z)}{||f(z)||}\right\|^2\right)}\right)$$

with $f(a)$ the function returning the proxy of the same class as a and Z the set of proxies of different classes than the sample. To optimize that loss, ProxyNCA is built using three blocks: a backbone architecture (any neural network, taken pre-trained), an embedding layer (randomly initialized), and proxies (randomly initialized).

ProxyNCA++ adds to the blocks of ProxyNCA:

- a global Max Pooling layer attached to the backbone model
- Layer normalization after the backbone model
- Balanced sampling in the creation of the batches of input data. N_C classes are selected and an equal number of samples from each of those classes is taken to make the batch.
- Fast-moving proxies. The learning rate used in the learning of the proxies is set higher than the learning rate of the rest of the model.
- Proxy assignment probability. ProxyNCA++ maximizes the probability of a data point being assigned to its class proxy. The difference with ProxyNCA is that ProxyNCA was trying to maximize the ratio of the distance of a sample and its proxy and the distance of a sample and the other proxies, while ProxyNCA++ maximizes the assignment probability directly.
- Temperature scaling. Temperature scaling, $q_i = \frac{\exp(y_i/T)}{\sum_j \exp(y_j/T)}$, is added to the loss function with the distance in order to refine the boundary.

The loss is then given by (with A the set of all proxies):

$$L_{proxyNCA++} = -\log\left(\frac{\exp\left(-\left\|\frac{x_i}{||x_i||} - \frac{f(x_i)}{||f(x_i)||}\right\| \cdot \frac{1}{T}\right)}{\sum_{f(a)\in A}\exp\left(-\left\|\frac{x_i}{||x_i||} - \frac{f(z)}{||f(z)||}\right\| \cdot \frac{1}{T}\right)}\right)$$

Normalized Softmax

The Normalized Softmax loss, presented in [Zhai and Wu, 2019] is similar to the ProxyNCA++ loss, the difference being the distance metric used. Normalized Softmax uses Cosine distance instead of Euclidean. The loss is expressed by

$$L_{NormalizedSoftmax} = -\log\left(\frac{\exp(x_i^T f(x_i)\cdot\frac{1}{T})}{\sum_{f(a)\in A}\exp(x_i^T f(a)\cdot\frac{1}{T}))}\right)$$

using the same notation as in ProxyNCA++. The sampling strategy is also kept identical to the one of PRoxyNCA++ (balanced sampling).

SoftTriple

This loss is described in [Qian et al., 2019]. It is inspired by both the SoftMax loss and the triplet loss.

The softmax loss is a classification loss. It maximises the conditional probability of the output label given the embedding of the input:

$$l_{softmax} = -\log\frac{\exp(w_{y_i}^T x_i)}{\sum_j \exp(w_j^T x_i)}$$

with $[w_1, ..., w_C]$ representing the last FC layer (C is the number of classes) and y_i corresponding to the label of x_i.

The triplet loss is a loss that pushes embeddings of similar samples closer and embeddings of dissimilar samples further. It is expressed as

$$l_{triplet}(x_i, x_j, x_k) = [\rho + x_i^T x_k - x_i^T x_j]_+$$

with x_i and x_j considered similar, x_i and x_k dissimilar and ρ is the margin.

SoftTriple comes from adding K centers per class. Those centers define new similarity measures and constraints. The similarity between the sample x_i and the class c is given by $\mathcal{S}_{i,c} = \max_k x_i^T w_c^k$, with w_c^j $(j = 1, \ldots, K)$ the centers. The constraint is defined by

$$\mathcal{S}_{i,y_i} - \mathcal{S}_{i,j} \geq \rho \ \forall j \text{ with } j \neq y_i$$

From those formulas, a relaxed similarity is designed with $\mathcal{S}'_{i,c} = \sum_k \frac{\exp(\frac{1}{\gamma}x_i^T w_c^k)}{\sum_k \exp(\frac{1}{\gamma}x_i^T w_c^k)} x_i^T w_c^k$ which is in turn used in the definition of the SoftTriple loss:

$$l_{SoftTriple}(x_i) = -\log\frac{\exp(\lambda(\mathcal{S}'_{i,y_i} - \rho))}{\exp(\lambda(\mathcal{S}'_{i,y_i} - \rho)) + \sum_{j\neq y_i}\exp(\lambda\mathcal{S}'_{i,j})}$$

The final form is obtained by introducing a Regularizer after making the number of clusters adaptive. The regularizer is given by $R(w_j^1, ..., w_j^K) = \sum_{t=1}^K \sum_{s=t+1}^K \sqrt{2 - 2w_j^{sT} w_j^t}$ leading to the final form:

$$\min \frac{1}{N}\sum_i l_{SoftTriple}(x_i) + \frac{\tau \sum_j^C R(w_j^1, ..., w_j^K)}{CK(K-1)}$$

It is also combined with the same sampling as the two previous proxy losses.

2.4.2 Deep Ranking

Deep ranking was introduced in [Wang et al., 2014]. The authors were concerned about finding a method to rank images based on their similarity to a given query. They wanted the images to be ranked based on what they really contain rather than simply on their label, making the distinction between *fine-grained similarity* and *category-level similarity*. As a result, they could not rely on classification models as they would not be able to look past the labels of the images. Instead, they designed the *Deep ranking method*. It is a DML method but with a fixed loss, and fixed backbone structure.

The paper uses the squared Euclidean distance between the embeddings as the similarity measure. The input images are processed as triplets, with a triplet $t_i = (x_i, x_i^+, x_i^-)$ being formed with the anchor image or query x_i, a positive image x_i^+ and a negative one x_i^-. The goal of the model is to find an embedding $f(.)$ such that

$$\|f(x_i) - f(x_i^+)\|^2 < \|f(x_i) - f(x_i^-)\|^2,$$

$$\forall x_i, x_i^+, x_i^- \text{ such that } r(x_i, x_i^+) > r(x_i, x_i^-)$$

with $r(x_i, x_j)$ the so-called *pairwise relevance score*. This score measures how much two images are similar and requires an expert knowledge to evaluate these values.

The objective here is to find an embedding that minimizes the distance between similar pairs compared to dissimilar pairs.
This leads to the definition of the objective function,

$$\min \sum_i \xi_i + \lambda ||\mathbf{W}||^2$$

$$s.t. : \max\{0, g + \|f(x_i) - f(x_i^+)\|^2 - \|f(x_i) - f(x_i^-)\|^2\} \leq \xi_i$$

$$\forall x_i, x_i^+, x_i^- \text{ such that } r(x_i, x_i^+) > r(x_i, x_i^-)$$

with λ a regularization parameter, $\mathbf{W}$ the parameters of the feature extraction model and g a gap parameter (which impacts the distance between the negative and positive pairs). Note that the left part of the first constraint is the Hinge Loss for a triplet.

To process the input data by triplets, the backbone model consists in repeating three times the same network, in parallel (Figure 2.14). It computes the embeddings of each image composing the triplet before giving them as input to the ranking part. That part simply computes the loss and back-propagates the resulting gradient to the model. The network is composed of a CNN to which two shallow networks have been attached (Figure 2.15). The shallow networks are made by a subsampling layer followed by a convolutional and a max pooling layer, and finished by a l2 normalization layer. The outputs of the three parts are embedded together before going through one last normalization.

The last element presented in that paper is the sampling strategy used to make the triplets of images. It is not feasible to consider every possible combination due to the number of input images. Instead, they establish a process that selects the most important triplets. The first step is the selection of the anchor image. It is based on what they call the relevance score

$$r_i = \sum_{j:c_j=c_i, j\neq i} r_{i,j}$$

where $r_{i,j} = r(x_i, x_j)$. The relevance score is a value attached to an image in order to characterize how it relates to the other images of the same label. The highest this value is, the likelier it is

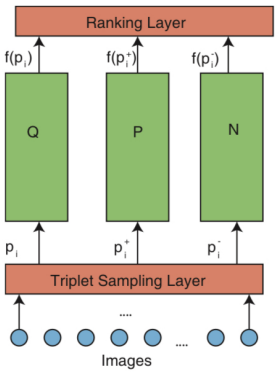

Figure 2.14: Structure of the deep ranking model. Credits: [Wang et al., 2014]

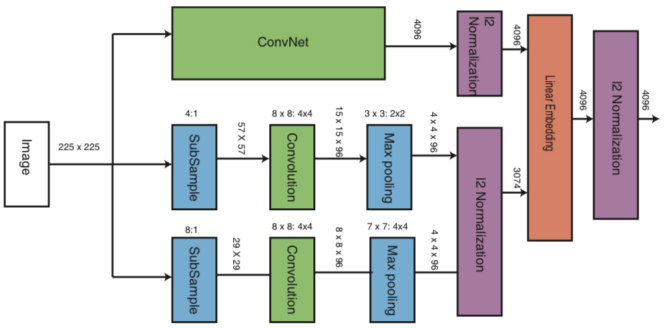

Figure 2.15: Structure of the backbone model. Credits: [Wang et al., 2014]

that the image will be chosen as the query. Then, the positive image x_i^+ is selected from the same category as the anchor such that the images with the highest pairwise relevance to the anchor are more likely to be selected. For the negative image x_i^-, it is also selected from the same class as the anchor (*in-class* sample). Like for the positive sample, the highest the pairwise relevance score between the negative image and the anchor, the likelier this image is selected. However, there is a need to have enough difference between the score of the positive/anchor pair and the score of the negative/anchor pair, which is enforced using a threshold value. This sampling is possible because the images have been hand-annotated and ranked which makes the values $r(x_i, x_j)$ known. In our work, such scores are not available. As such, the sampling had to be adapted and a simple random sampling is used. With this sampling, the positive image is selected at random from the same class as the anchor and the negative is randomly taken from the images of all the other classes.

2.4.3 Contrastive Learning

Contrastive learning [12] is another branch of DML that consists in using pairs or triplets of inputs to learn embeddings that discriminate well between similar and dissimilar samples. It relies on comparisons between a query image and a positive or/and a negative image to learn the embeddings, based on how 'humans' process images: when seeing two images of cats and one of a dog, we intuitively know that the cats are similar and different from the dogs, and are capable of distinguishing which features differentiate them. As triplet comparisons were already explored in the previous section, only pairs will be considered for this technique. This restricts the possibilities for the elements characteristic of a DML system to the following categories:

- A siamese Network
- A pair sampling strategy
- A pair-based loss

This does not however prevent the existence of multiple options in those smaller categories, depending again on the specific task the concept is applied to. For example, in [Wiggers et al., 2019], a Siamese Network constituted of two AlexNet is used with a sigmoid cross entropy loss to do

[12] Source: Towards Data Science - Understanding Contrastive Learning - by Ekin Tiu and [Le-Khac et al., 2020]

image retrieval and pattern spotting. In [Wang et al., 2023], custom encoders are combined with InfoNCE loss for WSI image retrieval. Contrastive learning can also be used in a supervised or unsupervised fashion. Indeed, as the methods work with pair of images, the only information they need is those pairs and how their elements relate to one another (which is positive or negative). This is done through the pair sampling strategy that either uses labels to form the pairs ('basic' contrastive learning) or dot not require them (*Augmented* Contrastive Learning).

Feature extractor

As the method is based on pairs of images, a specific structure is used, called *Siamese Network*. To build a Siamese Network, a selected network is repeated twice, usually identically the two times (same parameters and architectures, weights shared). This selected network can be any deep neural network. For this work, it has been restricted to any of the architectures presented in Sections 2.3.1, 2.3.2, and 2.3.3.

Losses

A wide range of losses exists to be used with CL. Four of them were selected based on a quick review of the literature.

Contrastive Loss

This first loss is the most classic loss of contrastive learning, as indicated by the name. There exist different definitions of it but one of the most recent was proposed in [Hadsell et al., 2006]. This article uses the contrastive loss with the objective of dimensionality reduction. They developed such a loss because they were in need of a way to make similar elements map to close points in the new dimensional space while dissimilar elements would map to more distant points.

The loss is based on the Euclidean distance of the embeddings/feature vectors.

$$D_W(x_i, x_j) = ||f(x_i) - f(x_j)|| := D_W$$

with W the weights of the feature extractor, x_i, x_j the inputs and $f(.)$ the embedding function.

Setting $y = 0$ if x_i and x_j are similar, $y = 1$ otherwise, the contrastive loss final expression is given by

$$L(W, y, x_i, x_j) = (1 - y)\frac{1}{2}(D_W)^2 + y\frac{1}{2}\{\max(0, m - D_W)\}^2$$

with m a margin.
The first term of the right-hand side corresponds to the loss attributed to similar pairs ($y = 0$ so $1 - y = 1$), L_S, while the second term corresponds to the loss attributed to dissimilar pairs, L_D. The margin limits the dissimilar pairs that can contribute to the loss. If the distance between the two inputs of a dissimilar pair is too big, then the term $m - D_W$ will be negative and 0 will be picked, making the total loss null. This is the wanted behavior because if the distance for a dissimilar pair is big it means that the model does well in separating dissimilar inputs and it does not need to be further corrected. Note that L_D is required in order for the model not to collapse. Without it, a trivial solution for minimizing the loss would be to get the same embedding for all input ($f(.)$ constant) which would lead to a dimensional collapse.

Cosine Embedding loss

While the Euclidean distance stays one of the most common distances used in similarity computation, another renowned one is cosine similarity:

$$similarity = \cos(x_i, x_j) = \frac{x_i \cdot x_j}{\|x_i\|\|x_j\|}$$

The cosine loss has been designed based on that distance and used in several papers such as in [Nguyen and Bai, 2010]. There exist again several definitions of this loss. In this work, the definition used is quite similar to the formula of the contrastive loss but adapted to the cosine distance, and based on PyTorch implementation[13].

Setting $y = 1$ if x_i and x_j are similar and $y = -1$ otherwise, the loss is given by

$$l_{CEL} = \frac{1+y}{2}(1 - \cos(f(x_i), f(x_j))) + \frac{1-y}{2}\max(0, \cos(f(x_i), f(x_j)) - m)$$

with m a margin and $f(.)$ the embedding function. The first term of the right hand-side is again the loss corresponding to similar pairs while the second term is the loss corresponding to dissimilar pairs. It can be seen that the same type of clipping of the distance to 0 is observed when the distance of the dissimilar pair is too big (the highest the distance, the smallest the cosine). The advantage of that loss compared to the previous one is the limitation of the value that can take the loss, as the cosine is limited between -1 and 1.

Binary Cross Entropy Loss

Cross entropy is a commonly used loss in deep learning, especially in classification tasks. Binary cross entropy is a specification of that loss when only two classes are present in the dataset. It can be easily adapted to similarity computation by setting the 'binary category' to similar-dissimilar and feeding to the sigmoid function the distance values. Basically, the loss is obtained following the definition of the Binary cross entropy:

$$l_{BCE} = \frac{1}{N}\sum_{k=1}^{N}(1 - y_k)\log(1 - p(y_k)) + y_k \log(p(y_k))$$

where the y_k are defined as $y_k = 1$ if the kth pair (x_i, x_j) is dissimilar, 0 otherwise and the $p(y_k)$ are defined as $p(y_k) = sigmoid(D)$ with D the Euclidean distance.

Similarly to the two previous losses, the first term represents the loss associated with a similar pair while the second term represents the loss associated with a dissimilar pair. Indeed, if a dissimilar pair has a big distance, then the sigmoid will output a value close to 1 which, once put into the log, will give zero and cancel the loss.

InfoNCE loss

The infoNCE loss[14], or Noise Contrastive Estimation of mutual information, uses the principle of mutual information. Mutual information is interesting for similarity tasks as its value directly relates to the similarity of two inputs. Indeed, mutual information is defined as the amount of information that two variables share or the degree of dependency between two variables[15]. Hence, two variables that are similar will be more dependent than two variables that are not and will have a higher mutual information value.

Several works in CL have been using mutual information-based losses to train their models. In those, InfoNCE is quite a popular implementation, being used notably in [Chen et al., 2020] and [van den Oord et al., 2019].

Multiple variations of this loss have been proposed. The one used in this work, called NT-XentLoss, is obtained through the following formula:

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K}\exp(q \cdot k_i/\tau)}$$

[13]Cosine embedding loss - pytorch

[14]Info NCE - Papers with code

[15]Sources: Medium - A deep conceptual guide to mutual information and Information and coding theory lecture - 2022

where q represents the parameters of the models, k_+ is the cosine similarity of the positive pair, and k_i is the cosine similarity of a pair. τ is an additional parameter called the Temperature. Note that the cosine similarity is not only computed between the different views of a single image but also between all views of all images (two views of different images will create a negative pair).

In order to work, the NT-Xent loss requires the presence of at least one positive sample per anchor image. As pair-based contrastive learning leads to half the pairs being only composed of the anchor and a negative sample, it will not be possible to combine it with the NT-Xent loss. As a result, for this loss, it will be used with a triplet network (like in Deep ranking but without the two shallow networks).

Sampling strategy

As in deep ranking, several strategies exist to form pairs of inputs. For feasibility reasons, due to the lack of pairwise 'similarity score' in the dataset, the creation of the pairs relies on random draws. However, two situations must still be distinguished. The first one corresponds to a classic application of contrastive learning while the second is the Augmented contrastive learning mentioned earlier. On top of that distinction, there is also a second one that must be taken into consideration. Contrastive learning is based on pairs of inputs, both negative and positive pairs. One of the parameters to set in the sampling strategy is the proportion of each type to make. In one specific case, called *Non-Contrastive* Learning, the proportion is set to 0-1, with only positive pairs being made.

Non-Contrastive learning is explained in [Tian et al., 2021]. It has been introduced following the realization that the selection of the negative pairs could have a tremendous impact on the results. If the mining of the pairs fails to take into account some characteristics or links between the inputs, and wrongly put dissimilar inputs into similar pairs, then the model will learn to put together the dissimilar pairs which is the opposite of what is hoped for. As a result, it was decided to learn a representation based only on positive pairs. However, negative pairs are required to avoid what is called *dimensional collapse* which is what happens when the solution is so trivial that only a very small subset of the embedding space is taken. To avoid that and still only use positive pairs, two elements are added to the architecture: a stop gradient and a predictor. The method uses two networks, one of them being the predictor and the other the target. Those networks are then trained such that their outputs would be identical. The stop-gradient element intervenes in the training of the target network. The target network cannot be trained using gradient descent as it has been shown that it was the gradient propagation through the target network that was leading to dimensional collapse. The stop gradient is thus placed such that the gradients are not propagated to it and its weights are updated based on the weights of the predictor network, through different techniques depending of the model (Exponential moving average for example).

With those distinctions explained, the end of this section is presenting the sampling strategies of interest.

'Supervised' Contrastive Learning

To create the pairs, each image of the batch is used as an anchor once. The positive image is randomly selected from all the images of the dataset of the same class as the anchor. The negative image is randomly selected from all the images of the dataset with a different class than the anchor. If the anchor has an even index in the batch, then the returned pair is positive, otherwise, the returned pair is negative. This leads to a proportion of 50-50 between the negative and positive pairs.
In addition to that sampling, for two experiments on the dimensional collapse effect, the proportion is changed to 0-100 with only positive pairs being returned.

To create the pairs, ACL does not rely on the labels of the images. Instead, it applies different transformations to the anchor image in order to create different views that are then used as positive sample and/or as anchor. When negative pairs are created, the negative sample is randomly drawn from the rest of the images of the dataset, regardless of their class.

Several transformations as well as ways to apply the transformations (= pipeline) have been devised over the years[16]. Three different ways are presented in the following.

1. Unique. This way is the simplest way of creating a pair. The transform is applied to the anchor in order to generate a positive sample. The positive pair is then formed with the original anchor.

2. AMDIM (Augmented Multiscale Deep InfoMax). This pipeline is introduced in [Bachman et al., 2019], alongside a complete framework for representation learning, of same name. As indicated by the name, this framework uses the principle of mutual information to learn representations of the input data. The mutual information is computed on the representations of two views extracted from the same input image. The creation of those views is done by the AMDIM pipeline. The idea behind this pipeline was actually introduced in [Dosovitskiy et al., 2016]. In this paper, a set of transforms are defined and a vector containing the values of each of their parameters is randomly initialized a certain number of times. One set of transforms per parameter vector is applied to the initial image to give a new view of the image and lead to a set of transformed patches. In AMDIM, the parameters of the transforms are fixed and they are applied only two times, to make the anchor and the positive sample.

3. CPC (Contrastive predictive coding). This pipeline is used in a representation learning framework of same name, presented in [van den Oord et al., 2019]. It is also based on mutual information combined with a prediction model. The CPC pipeline is used to process the input image such that it can be used by the model in the correct format. The input image is split into overlapping patches and the patches are then fed one by one to the model and then used to predict future image patches. This pipeline was slightly adapted to form pairs of data to be used in contrastive learning with the anchor and the positive sample being one of the patches of a given image and the negative sample being a patch from another image.

Several works have been based on the AMDIM pipeline and offer slight variations of it. Two of them are presented in Section 3.4 of Chapter 3, one for ACL and one for ANCL.

[16]Sources: A Framework For Contrastive Self-Supervised Learning And Designing A New Approach - William Falcon, Medium, 2020 (based on [Falcon and Cho, 2020]) and Self-Supervised Learning - Part 3: The idea of Amdim and comparison with two other contrastive learning approaches, 2021.

Chapter 3

Theory linked to CBIR

3.1 Content-based Image Retrieval

3.1.1 Introduction

Content-based image retrieval (CBIR) is a task that consists in retrieving from an image database the images the closest given their content to a given image, called the query. CBIR is a really global task in the sense that it involves a lot of aspects of the computer science field. The algorithms for the search and representation of the images are part of the AI, ML, and DP branches of computer science. The indexing of those representations calls to notions of database and data management. In order for the system to be operational and usable, it must be effective which requires basics in optimization, infrastructures, and distributed systems. In our work, it is the first aspect that is under investigation. The efficiency of the system is also taken into account but more for the efficiency of the different algorithms and models used.

As stated in Chapter 1, a CBIR framework is composed of two big sections. The feature extraction or representation learning part composes the first section while the indexing and search operations compose the second section. The architectures and methods used to realize the first section have been introduced in 2. They have for purpose to lead to a model that returns a vector of features as a representation for an image. It is the section on which our work experiments the most as it is also the section that requires the most use of AI/DP/ML. The second section is built upon those feature vectors. They are the elements indexed in the database as well as the elements onto which the similarity computation is executed. The methods and infrastructures used for it are described in the following sections, Section 3.3 and 3.2.

For the use of the CBIR framework, i.e. the CBIR process, it can be split into three 'operations', strongly linked to the two sections composing the framework, and represented in Figure 3.1. Each of those operations can be again subdivided into smaller time steps/operations.
The first operation makes the CBIR framework operational. It is the training of the feature extractor. The training can be separated into 5 actions (Figure 3.2). Two are related to the preparation of the data to be fed to the model: *Dataset* creation and *Data processing*. Two are related to the building of the model: *Model acquisition* and *Model alterations & parameters setting*. The final step is the training itself.
The second operation is the indexing of the data in the database. It can be split into 5 actions (Figure 3.3). The first two actions consist in obtaining the elements to be indexed in the database, *Data processing* and *Feature extraction*. In parallel, the database is initialized and activated (3^{rd} action). Then, the vectors are indexed into the database and the related structures. The fifth and last action is optional. The index of the database can be trained to make the search more

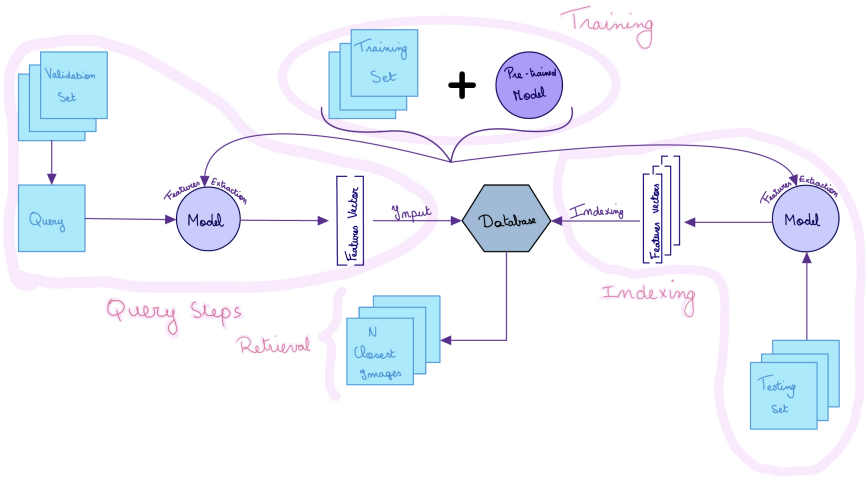

Figure 3.1: Scheme of the CBIR process

efficient, at a small cost of accuracy.
The third operation is the retrieval itself. It is composed of 3 to 4 actions (Figure 3.4). The first two are the same as for the indexing, with the acquisition of the feature vector of the query. This vector is then given to the database where similarity computations based on the selected search algorithm are executed. An additional step, when the purpose of the retrieval is the evaluation of the CBIR framework, is the assessment of the results through quantitative measures and/or qualitative tests.

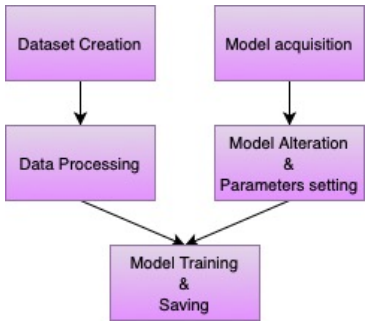

Figure 3.2: Training process

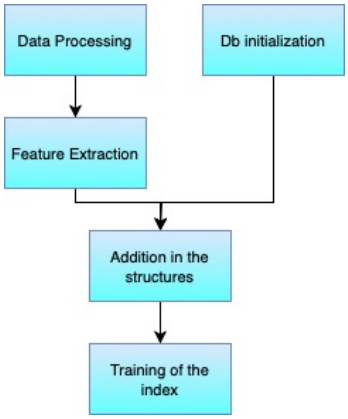

Figure 3.3: Indexing process

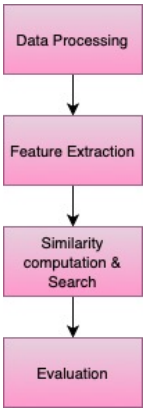

Figure 3.4: Retrieval process

3.1.2 Evolution & frequent elements

CBIR is a concept that has been researched for a very long time due to its many uses, especially for commercial purposes (e.g. recommendation systems). During this time, CBIR has reflected the evolution seen in the technologies and the concepts, with different approaches being used at different times. On top of the diversity brought by the evolution of the technique, CBIR has also shown a wide variety in the options/elements of the systems conceived for that purpose. It is neither feasible nor in the scope of this project to track and list all the different architectures and techniques used for CBIR. Instead, a brief description of the general state of the field, with its evolution and its most popular elements is proposed in the following.

The biggest shift that has been observed in CBIR frameworks is the way the features are retrieved from the images ([Kumar et al., 2013]).

The earliest works were more focused on what is called *local or low-level features* ([Dewan and Thepade, 2020] Those features were in a way manually extracted from the images. It was specified to the model which type of features it had to extract such as color [Jain and Vailaya, 1996], shape [Jain and Vailaya, 1996], texture [Manjunath and Ma, 1996], edge,... Some works ([Singh and Srivastava, 2017]) also use slightly more intricate features through derivation and manipulation (color moments, skewness, fusion,...) of the lowest-level features. The techniques used to retrieve such features vary too. In [Jain and Vailaya, 1996], it uses histograms of color and of edge directions. They build the histograms for each image and the similarity is obtained by computing the Euclidean distance of the intersection of the histograms of the query with the histograms of the images. In [Manjunath and Ma, 1996], a filter based on the Gabor wavelet transform is used to retrieve coefficients at different scales. The mean and standard deviation of the magnitude of the coefficients are then used as features. The vector of the query and an image are then compared using the sum of the absolute values of the differences between each of their elements, normalized. In [Bay et al., 2006], a new model called SURF is introduced. It is used to extract local features such that they are invariant to scale and image rotation. Two aspects are mainly considered, the detector (where in the image to extract features) and the descriptor (the feature vector to build). The descriptor is based on the Hessian matrix while the descriptor is obtained through the use of wavelets and Gaussian filters. Other techniques include the use of Zernik/Chebyshev/... moments, Fourier transforms, different thresholds or filters for pattern recognition, bag-of-words, quantizations, SIFT features extractor, etc.
For local features, the most renowned techniques found in CBIR frameworks are the SIFT and SURF extractors, sometimes combined with a bag-of-words technique. Histograms and moments are also widely used but in even earlier works. The search and retrieval method associated is frequently a brute-force search algorithm followed by a ranking based on the computed distances.

With the advances in deep learning, this tendency shifted to the extraction of *global or abstract features*, which are of a higher level than the features obtained through the previous methods. Those features are obtained through the use of a deep neural network as the feature extractor. In [Minarno et al., 2021], an autoencoder using a CNN as backbone learns how to represent the input images. The latent representation is then used as the feature vector. The similarity is then computed by taking the Euclidean distance between the vector of the query and the vector of another image. In [Chen et al., 2022], a self-supervised model, SISH, is introduced. It works on WSI and uses K-Means clustering to split them into patches. The patches then go through two models. The first one is a Vector quantized-Variational AutoEncoder (VQ-VAE) that is used to produce an integer representing the index of the patch. The second is a DenseNet121 that produces feature vectors, later binarized. Those values are put into a Van Emde Boas tree that is used during the search (a variation of nearest neighbor based on the Hamming distance). In [Zheng et al., 2021], a new deep metric learning model is proposed, using a relational module. First, a CNN extracts a feature vector from an image. This feature vector is then passed through K fully connected layers that extract a specific individual feature. Those individual features represent both intra and inter-class variations. The CNN is trained using classic deep metric learning methods. To train the layers, an autoencoder-like training is put in place, with the inverse architecture built to reconstruct the original feature vector and a reconstruction loss comparing them. Three such networks are placed in parallel, with weights shared. The first one gets the individual features while the others are used to retrieve relations between the features using the relational module. All the information is put together in a graph, with the individual features being the nodes and the relations the edges. Based on that graph, an embedding is conceived for each image. Euclidean distance is used as metric during the retrieval process.
Globally, CNNs and transformers are popular architectures for feature extraction and are generally trained using deep metric learning and its sub-fields (CL and DR). Autoencoders are also a popular choice when labels are not available. For the search method, tree-based methods are used in the big majority of the cases, with k-nearest neighbors being the favorite.

Some works also add a dimensionality reduction step after the feature extraction, through the use of PCA, hashing, or other coding techniques.

3.2 Search algorithms and indexing

An important consideration in CBIR is how to search in the database of images for the most similar ones.

Similarity search is different than a traditional search operation. In the first instance, the complexity of the search operation is linked to the indexing of the data in a chosen structure. In the second instance, the search algorithm is the key to the success of the operation. Indeed, in the second case, the goal of the search is to find a specific element, that is known. In our case, the goal is to retrieve the most similar element to a given query, but the searched element is not known before checking all available ones. By default, the most accurate search in such a case is the *brute-force search*, which consists in browsing through the entirety of the content of the database and computing the similarity to the query for each element. However, at the cost of some accuracy, the data may be placed in the database in a way such that the search operation would only need to access some elements and not all. Hence, the time efficiency of the whole operation is improved.

A similarity search method must compromise between accuracy and efficiency. It must provide three essential elements: the indexing method, the similarity distance, and the search algorithm. Given the problem it addresses, these elements must satisfy different requirements such as efficiency, accuracy, memory footprint, scalability,...

The similarity search field is quite active, with many publications proposing indexing systems for efficient search. For example, in [Maliki et al., 2019], they define a method named GNAT - Geometric Near-neighbor Access Tree. This method consists in building a tree that partitions the embedding space. Images whose embeddings are similar are close in the tree. The retrieval is then done by executing the k-nearest neighbors algorithm on the tree. Diverse libraries[1] have been established in different programming languages to offer such services. In our work, the FAISS library was selected based on the work in [Defraire, 2021] as well as further research after which the other possibilities did not appear to bring further advantages.

3.2.1 FAISS

FAISS is a library designed for similarity search through the implementation of several indexing methods and associated search algorithms based on nearest neighbors search[2]. It was introduced in [Johnson et al., 2017]. It was conceived as a response to the need for a system that is based on vector representations and capable of handling very large datasets. As such, it provides a speed and memory-optimized, vector-based indexing structure. It is capable of handling a large number of vectors, vectors that can be high-dimensional without disturbing the process. The structure can be used alone (in-memory database) or combined with compatible databases. Finally, it is highly customizable, with several indexing methods proposed, different search algorithms, and similarity measures.

Four backbone indexing techniques are proposed[3], leading to 30+ indexing structures by varying the combinations, similarity measures, and search algorithms.

[1] These two websites: github - Awesome vector search and Libhunt - Faiss alternatives offer a selection of such libraries

[2] Sources: Engineering at Meta - 'Faiss: A library for efficient similarity search', March 2017 and Medium - 'I used FAISS, so you don't have to', July 2022, Kacper Lukawski

[3] Source: FAISS documentation

- Flat: The vectors are indexed in the database sequentially.
- Inverted index ([Sivic and Zisserman, 2003]): one vector represents a group of vectors. The index is trained to form clusters of vectors through K-means clustering. One vector per cluster is computed based on the vectors composing the cluster and acts as a representative.
- Product quantization ([Jégou et al., 2011]): the vectors are encoded using product quantization in order to reduce the memory needed for storage as well as the similarity computational time.
- Hierarchical Navigable Small World Graph ([Malkov and Yashunin, 2018]): a graph is built based on the similarity between vectors. The search is optimized by simply passing through the graph.

As part of this thesis, two indexing structures are used. The first indexing structure is known as *IndexFlatL2*. It implements the first backbone technique and uses as the similarity measure the Euclidean or L2 distance. The search algorithm is brute force, where all vectors of the database are accessed in order to retrieve the k most similar. It is the basic index of FAISS and also the most accurate as it does not approximate the search. Formally, with x a query vector and $y_i, i \in [0, l]$ the vectors indexed in the database, the pairwise distance matrix $D = [||x_j - y_i||^2]_{j=0:n, i=0:l}$ is computed using the practical formula $||x_j||^2 + ||y_i||^2 - 2 < x_j, y_i >$. The second index is *IndexIVFFlat*. It combines the two first backbone techniques and also uses the Euclidean distance as the similarity measure. The vectors are split into C clusters using K-means clustering. The τ closest clusters are retrieved by comparing the query to the centroids:

$$\mathcal{L}_{IVF} = \tau\text{-argmin}_{c \in C} ||x - c||_2$$

The vectors of the retrieved clusters are then compared through brute search like in a flat index.

3.3 Data management

While FAISS offers an in-memory database to store the vectors, the CBIR framework still requires a database to conserve the mapping between the indexes used in FAISS and the file names they correspond to. The selected database must have similar characteristics to the search infrastructure. It must be able to scale well to a high number of inputs and it must be efficient such that the access to the data is fast.

Several structures have been developed to serve as databases, in multiple languages such as MongoDb[4], DragonFly[5], Apache Cassandra[6], etc. For this work, the selected database is Redis, based on [Defraire, 2021] and because further research did not bring the need to replace it with another alternative.

3.3.1 Redis

Redis[7] is a database that can be either in-memory or cached. It can also be used as a message broker. It is persistent and handles load distribution through clustering methods. The more important in our case is that Redis is fast (used in-memory) and scalable as it can deal with huge loads of data. It can handle several data types including basic strings and integers, which

[4]mongoDb website
[5]Dragonfly website
[6]Apache Cassandra website
[7]Official website

are what is used for the mappings in this work. It also offers several structures for the database such as lists, hashes, and dictionaries. The structure we are concerned about is the key-value structure. One element is registered as the key in the database and a value is associated with it. When searching for a pair, the database only needs to be given the key to retrieve the correct pair, leading to an efficient search. Redis also supports json, allowing to set several values for one single key by registering them in a json file.

Many libraries[8] were implemented to be able to use Redis with a specific programming language, allowing to easily integrate it in a framework. The library used as part of this work is *redis-py*. It furnishes an API allowing to use Redis functionalities directly from python code. It implements all of Redis functionalities and structures.

3.4 State of the art - Complete Frameworks

This section introduces both full CBIR frameworks and image representation models. The first two models, Yottixel and Smily are just briefly described. A more complete description can be found in [Defraire, 2021].

3.4.1 Yottixel

Yottixel, [Kalra et al., 2020] is a search engine specifically designed to deal with WSIs. It was built as a response to a lack of WSI-based search frameworks that work with unlabelled data. It is also designed as an alternative approach that does not rely on real-valued features or hashing that would alter the capacity of a framework to both scale and generalize well.

The paper offers two contributions. The first one is a method that splits a WSI into a mosaic of patches, that will serve as inputs for the network combined with an original representation of the patches, called *Bunch of Barcodes*. The mosaic of patches is created through the use of a K-means algorithm based on colors. Another K-means clustering is again applied to each obtained cluster, this time based on the spatial localization of the patches in the WSI. Random elements of each resulting cluster are selected to form the mosaic. The patches are then fed to a Densenet that extracts a 1024-features vector for each. Those vectors are then binarized using a discrete differentiation algorithm, leading to the barcodes (Figure 3.5). The second contribution

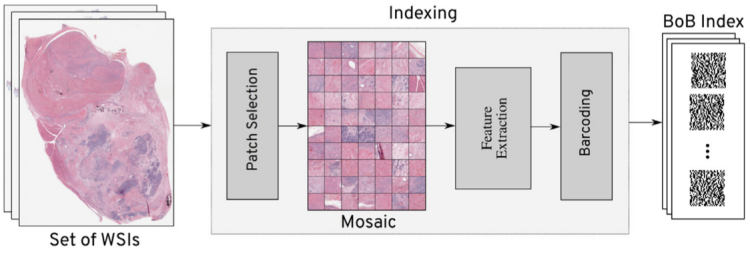

Figure 3.5: Yottixel's mosaic method and feature encoding

is the framework itself. Yottixel proposes two types of search. In the vertical search, the images to which the query is compared have the same primary site. In the horizontal search, all images are considered. To carry out the search operation, the query's barcode is computed. Hamming distance and nearest neighbors search are then combined to retrieve the most similar images.

3.4.2 Smily

SMILY (Similar Medical Images Like Yours), [Hegde et al., 2019], is another search engine for histopathology images. Smily was developed following the emergence of new imaging techniques

[8]Redis client libraries

leading to an increasing amount of histopathology images. Its purpose is to serve as a search engine for generic tasks, in opposition to architectures that specialized in a certain type of diseases or tissues. Smily has been tested by comparing retrieved results to the query and also by conducting a prospective study where pathologists had to evaluate the results of SMILY.

Smily first separates the input images into patches through overlapping splitting. Those patches go through some augmentations that lead to eight versions of each (rotations and mirrors). All the versions of all patches are then fed to a pre-trained CNN. This network is a DR network, with a CNN backbone that was pre-trained on non-histopathological images. It is trained by using triplet sampling and the loss is obtained by contrasting the distance of a similar pair to the distance of a dissimilar pair. This network returns 128-features vectors which are then indexed in a database.

For the search process, the feature vector of the query is first retrieved. It is compared to the vectors indexed in the database using the L_2 distance. The retrieved patches are filtered in order to not return several times the same patch with only a different orientation. They also filter the retrieved patches such that the selected patches are all slightly different from one another (if several results are too close to one another, only one of them is returned).

3.4.3 DINO: Emerging Properties in Self-Supervised Vision Transformers

DINO (self-DIstillation with NO labels), [Caron et al., 2021], is a self-supervised training method combined with a transformer-based architecture. The reason behind the conception of DINO is the lack of success of transformers in vision, which the authors attribute to the supervised way of learning that is frequently adopted. They argue that on the opposite, transformers encounter great success in NLP thanks to self-supervised learning, as has been seen in Bert and GPT models. Thus, they designed a self-supervised training method, based on other self-supervised training methods used in vision, that is used to train a ViT model.

Method

The method is inspired from BYOL (3.4.5), but uses a different loss and a slightly different architecture. For the techniques used, it extends knowledge distillation by applying it to unlabeled data and co-distillation by updating the teacher using the student.

The principle of knowledge distillation is to use two networks. The first one is the teacher network g_{θ_t} while the second is the student network g_{θ_s}. The student network is trained to match the output of the teacher network by minimizing the Cross entropy loss computed on the output probabilities of the two networks.

$$P_s(x)^{(i)} = \frac{\exp(g_{\theta_s}(x)^{(i)}/\tau_s)}{\sum_{k=1}^{K} \exp(g_{\theta_s}(x)^{(k)}/\tau_s)}$$

with τ_s the temperature setting the sharpness of the probability distribution.

$$\text{objective} = \min_{\theta_s} H(P_t(x), P_s(x))$$

with $H(a, b) = -a \log b$.

Dino uses a slightly different objective in order to make the method self-supervised:

$$\min_{\theta_s} \sum_{x \in \{x_1^g, x_2^g\}} \sum_{x' \in V; x' \neq x} H(P_t(x), P_s(x')$$

This new objective is due to the acquisition of V different views of a given input image. Two of those views, x_1^g and x_2^g, are global views, i.e. they are of resolution (224,224) and contain at least 50% of the original image. The other views, x', are local views. They are of resolution (96, 96) and contain smaller areas of the original image. All crops are fed to the student network but only the global ones are given to the teacher, as a way to force the network to match local areas to global ones.

Both networks have the same architecture and weights. Those weights are obtained by applying stochastic gradient descent to the objective function. However, the teacher weights are obtained from the student weights in a way such that they are frozen during an epoch (stop gradient present in the teacher to stop the gradient propagation). They are then updating using Exponential Moving average: $\theta_t = \lambda\theta_t + (1-\lambda)\theta_s$.

The network is composed of a backbone architecture f, which is chosen in the paper to be either ViT or ResNet. To that backbone architecture are added a 3-layer MLP with L_2 normalization and a FC layer. Those three elements compose the projection head h, such that the student and teacher networks are obtained by $g = h \circ f$.

Finally, the last important element of the method is the addition of sharpening and centering of the teacher output g_{θ_t} to prevent the collapse of the solution. The centering is applied by adding a bias term c to the teacher, with $c \leftarrow mc + (1-m)\frac{1}{B}\sum_{i=1}^{B} g_{\theta_t}(x_i)$ where B is the batch size and m a rate parameter. The sharpening is done through the τ_t parameter previously cited.

DINO is pre-trained on Imagenet (without using the labels and using BYOL data augmentations). It is then evaluated on different tasks (Image retrieval, Copy detection, segmentation, classification) with frozen features and with fine-tuning.

Results

Their diverse evaluations (3.6 and 3.7) show that their method leads to great results in all tasks. It manages to frequently outperform the current SOTA models or at least perform in a similar range. Specifically in retrieval, DINO outperforms all other methods shown on three out of four metrics and is only slightly behind for the fourth. In classification, DINO scored 80.1% in top-1 accuracy on Imagenet using linear classification.

			$\mathcal{R}$Ox		$\mathcal{R}$Par	
Pretrain	Arch.	Pretrain	M	H	M	H
Sup. [57]	RN101+R-MAC	ImNet	49.8	18.5	74.0	**52.1**
Sup.	ViT-S/16	ImNet	33.5	8.9	63.0	37.2
DINO	ResNet-50	ImNet	35.4	11.1	55.9	27.5
DINO	ViT-S/16	ImNet	41.8	13.7	63.1	34.4
DINO	ViT-S/16	GLDv2	**51.5**	**24.3**	**75.3**	51.6

Method	Arch.	Dim.	Resolution	mAP
Multigrain [5]	ResNet-50	2048	224^2	75.1
Multigrain [5]	ResNet-50	2048	largest side 800	82.5
Supervised [69]	ViT-B/16	1536	224^2	76.4
DINO	ViT-B/16	1536	224^2	81.7
DINO	ViT-B/8	1536	320^2	**85.5**

Figure 3.6: DINO results and results of the SOTA models for (a) retrieval (b) copy detection, [Caron et al., 2021]

Method	Data	Arch.	$(\mathcal{J}\&\mathcal{F})_m$	$\mathcal{J}_m$	$\mathcal{F}_m$
Supervised					
ImageNet	INet	ViT-S/8	66.0	63.9	68.1
STM [48]	I/D/Y	RN50	81.8	79.2	84.3
Self-supervised					
CT [71]	VLOG	RN50	48.7	46.4	50.0
MAST [40]	YT-VOS	RN18	65.5	63.3	67.6
STC [37]	Kinetics	RN18	67.6	64.8	70.2
DINO	INet	ViT-S/16	61.8	60.2	63.4
DINO	INet	ViT-B/16	62.3	60.7	63.9
DINO	INet	ViT-S/8	**69.9**	**66.6**	**73.1**
DINO	INet	ViT-B/8	**71.4**	**67.9**	**74.9**

Figure 3.7: DINO results and results of the SOTA models for video segmentation, [Caron et al., 2021]

From those experiences, they also noticed the emergence of two particularities, particularities that do not seem to appear when the training is supervised or when the backbone architecture

is a CNN. The first particularity is the apparition of segmentation masks in the extracted features. The features seem to capture the different boundaries contained in the images, allowing to differentiate between the different objects and making the method particularly adapted to segmentation tasks. The second particularity is how well the retrieved features fit with a k-nearest neighbor classifier or search, making it this time particularly adapted to image retrieval tasks.

3.4.4 SimCLR: A simple Framework for Contrastive Learning of Visual Representations

SimCLR, [Chen et al., 2020], is a network that learns how to represent images through the use of contrastive learning. It is self-supervised as it uses different views of the same image as the base for the comparisons. The authors focus on showing how four elements: data augmentation composition, a learnable nonlinear transformation between the loss and the representations, normalized embeddings, and large batch size, lead to improved results.

Method

To learn image representations, SimCLR uses an augmented contrastive approach. Basically, it takes as input an image and starts by generating two views of it through stochastic data transformations. These two views are then fed to the same neural network (a ResNet) that extracts feature vectors. The feature vectors are then given to a projection head that maps them to a different embedding space. After which, the pairwise similarities between all views[9] are computed. This pairwise similarity matrix is then used to compute the value of the contrastive loss function, itself used to upgrade the network parameters.

The first important element is the sampling strategy. N sample images are randomly selected and submitted to the augmentation process, leading to $2N$ images, and N positive pairs. Contrary to the 'usual' sampling strategy for contrastive learning, no samples are further drawn to create negative pairs. Instead, for each positive pair, all other views will be considered as a negative sample. This leads to 1 positive pair and $2(N-1)$ negative pairs for each of the N input images. They experimented with various values for N, from 256 to 8192.

The composition of the data augmentation used to get the two views is of great importance, as shown by the authors of the paper. They tried different combinations, containing elements of two families of transforms: transforms for spatial transformation such as rotation or cropping/resizing, and transforms for visual aspects such as color distortions or Gaussian blur. The final choice of augmentations comprises transformations from both families, as the authors determined that it was essential. It sequentially applies: cropping and resizing, random color distortions, and random Gaussian blur.

The ResNet network is used for extracting the feature vectors corresponding to the final, wanted, representation. However, for training, a non-linear projection head g is added to the network to project the representations h into another embedding space corresponding to the space of the loss: $z_i = g(h_i) = W^{(2)}\sigma(W^{(1)}h_i)$. The head makes it easier to apply the loss but the representations before applying it are better quality, hence the removal of the head after the training.

Finally, the loss function is the NT-Xent loss (a variation of the InfoNCE loss) which is a normalized, temperature-scaled cross entropy loss.

$$l_{i,j} = -\log \frac{\exp(sim(\mathbf{z_i}, \mathbf{z_j})/\tau)}{\sum_{k=1}^{2N} 1_{[k\neq i]} \exp(sim(\mathbf{z_i}, \mathbf{z_k})/\tau)}$$

[9] 'All views' means that views of two different input images are compared together, and not only the two views of a single input image

with $sim(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^T\mathbf{v}}{||\mathbf{u}||||\mathbf{v}||}$ and τ the temperature parameter. The normalization and τ parameter have shown to bring greater results than other losses without.

Results

SimCLR achieves 76.5% top-1 accuracy on ImageNet ILSVRC-2012 when combined with a linear evaluation protocol (weights frozen). When fine-tuned on 1% of the ImageNet labels, it reaches 85.8% of top-5 accuracy. On other classification datasets, it performed in a similar or better accuracy range than the selected SOTA model (Figure 3.8).

	Food	CIFAR10	CIFAR100	Birdsnap	SUN397	Cars	Aircraft	VOC2007	DTD	Pets	Caltech-101	Flowers
Linear evaluation:												
SimCLR (ours)	**76.9**	**95.3**	80.2	48.4	**65.9**	60.0	61.2	**84.2**	**78.9**	89.2	**93.9**	**95.0**
Supervised	75.2	**95.7**	**81.2**	**56.4**	64.9	**68.8**	**63.8**	83.8	**78.7**	**92.3**	**94.1**	94.2
Fine-tuned:												
SimCLR (ours)	**89.4**	**98.6**	**89.0**	**78.2**	**68.1**	**92.1**	**87.0**	**86.6**	**77.8**	92.1	**94.1**	97.6
Supervised	88.7	98.3	**88.7**	**77.8**	67.0	91.4	**88.0**	86.5	**78.8**	**93.2**	**94.2**	**98.0**
Random init	88.3	96.0	81.9	**77.0**	53.7	91.3	84.8	69.4	64.1	82.7	72.5	92.5

Figure 3.8: Results for the SimCLR model and ResNet50 model on different classification datasets.

3.4.5 BYOL: Bootstrap Your Own Latent. A new Approach to Self-Supervised Learning

BYOL, [Grill et al., 2020], is another representation learning model based on non-contrastive learning. The network was developed due to two 'issues' that were noticed in previous self-supervised representation learning models such as SimCLR. The first issue is the difficulty to sample negative images to form negative pairs such that those pairs help the model instead of sabotaging the learning process. The second issue is the impact that the image augmentations have, with a few changes in them leading to completely different results.

Method

To learn a representation, BYOL uses a technique similar to knowledge distillation. It is composed of two networks placed in parallel, the online network and the target network. The online network is trained to output a representation that matches the representation output by the target network.

The online network has three parts: an encoder f_θ, a projector g_θ, and a predictor q_θ. Its weights are denoted by θ and are different from the weights ξ of the target network, otherwise disposing of the same architecture minus the predictor (Figure 3.9). The two sets of weights are related through an exponential moving average: $\xi \leftarrow \tau\xi + (1-\tau)\theta$ with τ the decay rate. The encoders f are ResNet-50 models while the projectors g are an MLP combined with batch normalization, ReLUs, and a fully connected layer (output dimension equal to 256), following the projection head architecture of SimCLR. The predictor q_θ is the same as the projectors.

Before going to the network, the input image must go through a data augmentation pipeline to generate two different views of the same image. The augmentations chosen are the same as in SimCLR (random cropping and resizing followed by color distortions and Gaussian blur).

The first view v is fed to the online network and the second v' to the target network. It leads to the output of two representations, $y_\theta \triangleq f_\theta(v)$ and $y'_\xi \triangleq f_\xi(v')$ by the respective encoder, then

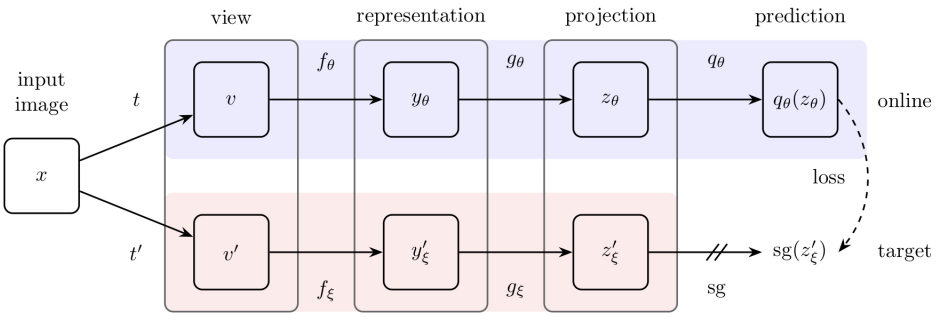

Figure 3.9: BYOL's architecture

followed by the output of two projections, $z_\theta \triangleq g_\theta(y)$ and $z'_\xi \triangleq g_\xi(y')$. The θ projection is fed to the predictor to try and match the ξ projection. Both elements are l_2 normalized and used to compute the loss.

$$\mathcal{L}_{\theta,\xi} \triangleq ||\bar{q_\theta}(z_\theta) - \bar{z'_\xi}||^2 = 2 - 2\frac{< q_\theta, z'_\xi >}{||q_\theta(z_\theta)||||z'_\xi||}$$

The loss is made symmetric by inverting the views and refeeding them to the network, leading to $\tilde{\mathcal{L}}_{\theta,\xi}$. The final loss is $\mathcal{L}^{BYOL}_{\theta,\xi} = \mathcal{L}_{\theta,\xi} + \tilde{\mathcal{L}}_{\theta,\xi}$ which is minimized with respect to θ only (ξ being updated based on the exponential moving average).

Results

Several experiments were conducted to assess BYOL performance. They first tested it on ImageNet by training a linear classifier and keeping BYOL's weights frozen, which lead to 74.3% top-1 accuracy using ResNet-50 as encoder, surpassing the several other models tested (SimCLR, MoCo v2,...). They then tested it on Imagenet but after fine-tuning it on a part of it using the labels (semi-supervised) which this time gave a top-1 accuracy of 53.2%, 68.8%, and 77.7% when trained respectively on 1%, 10%, and 100% of the ImageNet train set. The next experiment was to train it on other classification datasets which lead to the results shown in Figure 3.10. They finally tested it on two different vision tasks, segmentation and depth estimation, where it surpassed SimCLR, MoCo, and Supervised-IN in each configuration/metrics.

Method	Food101	CIFAR10	CIFAR100	Birdsnap	SUN397	Cars	Aircraft	VOC2007	DTD	Pets	Caltech-101	Flowers
Linear evaluation:												
BYOL (ours)	**75.3**	91.3	**78.4**	**57.2**	**62.2**	**67.8**	60.6	82.5	75.5	90.4	94.2	**96.1**
SimCLR (repro)	72.8	90.5	74.4	42.4	60.6	49.3	49.8	81.4	**75.7**	84.6	89.3	92.6
SimCLR [8]	68.4	90.6	71.6	37.4	58.8	50.3	50.3	80.5	74.5	83.6	90.3	91.2
Supervised-IN [8]	72.3	**93.6**	78.3	53.7	61.9	66.7	**61.0**	**82.8**	74.9	**91.5**	**94.5**	94.7
Fine-tuned:												
BYOL (ours)	**88.5**	**97.8**	86.1	**76.3**	63.7	91.6	**88.1**	**85.4**	**76.2**	91.7	**93.8**	97.0
SimCLR (repro)	87.5	97.4	85.3	75.0	63.9	91.4	87.6	84.5	75.4	89.4	91.7	96.6
SimCLR [8]	88.2	97.7	85.9	75.9	63.5	91.3	88.1	84.1	73.2	89.2	92.1	97.0
Supervised-IN [8]	88.3	97.5	**86.4**	75.8	**64.3**	**92.1**	86.0	85.0	74.6	**92.1**	93.3	**97.6**
Random init [8]	86.9	95.9	80.2	76.1	53.6	91.4	85.9	67.3	64.8	81.5	72.6	92.0

Table 3: Transfer learning results from ImageNet (IN) with the standard ResNet-50 architecture.

Figure 3.10: Results of BYOL and other SOTA models on different classification datasets., [Grill et al., 2020]

Chapter 4

Datasets

This chapter introduces the datasets used in the context of this work. The first two are only used as part of the training of the representation models while the third one is used for both training the representation model and testing the full framework.

4.1 ImageNet

Description

The ImageNet dataset[1] is a dataset of natural images obtained from the web and labeled by hand, with each image being given a single label. Those images belong to around 22 000 classes and are around 14 million making the full dataset take up to 1.3 TB of space. It is the biggest labeled dataset of natural images that exists and it has opened up new horizons for deep learning when it was created by allowing to train much deeper networks.

Due to that size as well as other disadvantages of the full dataset (class imbalance, non-exclusive labels, and lack of official splitting), a narrowed-down subset of images of this dataset has been officially established and is the one used as the reference by everyone. This smaller dataset, called Imagenet_1K, is 'only' composed of 1.2 million images, belonging to a total of 1000 classes, for a total size of 150GB. It is that dataset that is used in this work.

Figure 4.1: Samples from the ImageNet dataset

[1] https://www.image-net.org/index.php

Use

Ever since their creation, ImageNet_21K (complete) and ImageNet_1K have been used as references in the field of computer vision for the training of large neural networks. They frequently serve as benchmarks in image classification papers, with even a classification competition, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where the best classification models are competing against one another.

Given its importance, it is also the dataset (ImageNet_1K) that is the most widely used when pre-training a network, for transfer learning, and thus the one that has been used to obtain the weights of the pre-trained models readily available on PyTorch, Huggingface or Tensorflow (major python libraries for deep learning models). This is also its use in this work. ImageNet_1K is the dataset that is used when one of the backbone architectures used for feature extraction needs to be pre-trained. It has not been directly downloaded as the weights of the models trained with it are directly available using one of the libraries aforementioned.

4.2 The Cancer Genome Atlas (TCGA) - KimiaNet

Description

TCGA[2] has been created in 2006 and is a publicly available dataset of WSI representing healthy and cancerous tissues, the largest dataset of this type. The data is of several types: genomic, proteomic,... and has several subdivisions: primary sites ("=the anatomic location in a cancer patient that identifies the site of origin of a tumor"[3]), cancer types, genes, ...

In this work, the dataset used is a subset of that bigger dataset. It is composed of 7 126 WSI that have then been reworked into 240 000+ patches of size 1000x1000 and magnification x20. The original selected images are images that have all been obtained using the same preparation protocol, known as 'formalin-fixed paraffin-embedded'.

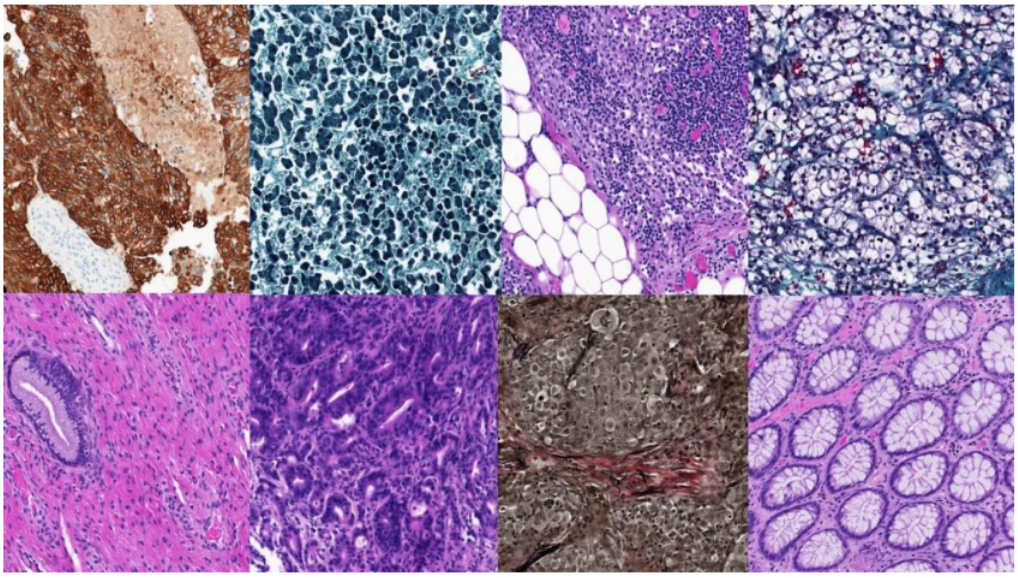

Figure 4.2: Samples from the KimiaNet dataset

[2]https://www.cancer.gov/ccg/research/genome-sequencing/tcga
[3]https://www.lawinsider.com/dictionary/primary-site

Use

Similarly to ImageNet, this dataset has only been used to pre-train a backbone architecture used for feature extraction. The difference between the two (besides the type of images) is that in its case, only one single architecture has been trained using it, with the objective to investigate the impact of the nature of the images in the pretraining step. This architecture is the Densenet neural network, though it takes the name KimiaNet [4] once trained with this dataset, as explained earlier. It has again not been downloaded directly but used through the downloading of the weights of the pre-trained network, immediately accessible through the Kimia lab website.

4.3 Histopathology

This dataset is the most important dataset of the three used for this thesis. It is used to train every feature extraction model used in the framework as well as for testing the indexing and search functionalities of it.

4.3.1 Acquisition and division

The images composing this dataset have been obtained through various sources and gathered by Romain Mormont as part of his Ph.D. thesis [Mormont et al., 2021]. Those sources include public datasets that were available for download as well as images from projects created on the Cytomine platform (patches extracted from WSI at different magnifications and positions).

Several organs are represented such as the lungs or kidneys as well as several pathologies such as breast or thyroid cancer. This variety is explained by the fact that this dataset is a regrouping of several other datasets that were each built for their own specific purpose. Those purposes differ in their objectives, both in types: classification - object detection - segmentation and in their specific end purpose: detection of diseased elements in a specific tissue type, detection of mitosis, cancer type classification, and so on. Another variability present in the dataset comes from the modality used to obtain the images (i.e. stains). The images were obtained through several different processes: H&E, IHC,... which impact the visualization of their content.

In total, this dataset is composed of 825 914 images for a size of around 107 GB. They are divided into 3 parts: the training set containing 633 499 images, used to train the feature extraction models, the test set containing 106 281 images, indexed in the database for the search part and the validation set of 96 066 images used as queries during the retrieval part. In the following, the test set will be referred to as the indexing set and the validation set as the query set. The images belong to 19 distinct datasets/Cytomine projects (later referenced as projects). Each of these projects contains two to nine classes. The class name is obtained by taking the project name + _ + 'class number in the project'. There is one more level, later referenced as 'similar', which is caused by some existing links between some of the projects. All projects are not completely independent, with some belonging to the same Cytomine project for example. This is important to take into account as images of projects which are 'similar' tend to be closer in content as opposed to images belonging to projects completely unrelated (though other factors are at play as explained in the next section) because they represent the same organ and have been taken following the same methodology.

All projects are presented in Table 4.1 with the number of classes they contain and the number of images. Similar projects are grouped together and separated by a single line while

[4] https://kimialab.uwaterloo.ca/kimia/index.php/data-and-code-2/kimia-net/

independent projects are separated by 2 lines. In the Annexe A, Table 7.1 gives the numbers of the classes composing the different projects (original numbers and new).

Project	$\#_c$	$\#_i$	Project	$\#_c$	$\#_i$
Camelyon16	2	292 226	ulg lbtd2 chimio necrose	2	882
cells no aug	2	3 637	ulg lbtd lba	8	4 284
patterns no aug	2	1 857	ulg bonemarrow	8	1 291
glomeruli no aug	2	29 213	tupac mitosis	2	77 853
iciar18 micro	4	4 800	warwicj crc	2	2 500
janowczyk1	2	31 725	umcm colorectal	8	5 000
janowczky2	2	3 402	mitos2014	3	64 873
janowczyk5	2	24 870	ulb anapth lba	9	5 420
janowczyk6	2	277 456	lbpstroma	2	2 313
janowczyk7	3	2 244	Total	67	835 846

Table 4.1: Division of the histopathology dataset $\#_c$ = number of classes, $\#_i$ = number of images

4.3.2 Visualization and analysis

Image samples

As previously stated, the images are really diverse[5] and this diversity is mainly explained by the different labels, but there is also some intra-class diversity that has to be considered.

Nevertheless, most images in a same class have a similar visual appearance, i.e. follow a similar theme, be it in the elements it contains or the main color scheme. This can be observed when looking at several samples of each class, as represented in Figures 7.68 to 7.82 in Appendix D. From those figures, the previous separation in classes, projects and 'similarity' is making sense as it can be seen that images of two classes of a same project are looking almost identical, with sometimes images from the other class more resembling to one of the class considered than other images of the same class. Links between some of the projects are also visible, with all the *janowczyk* projects having the same overall feeling (7.73, 7.74, 7.75) or the projects *patterns no aug* (7.69) and *cells no aug* (7.70) sharing very similar images.

A small representation of the dataset is available in Figure 4.3 with the images in their original format to show the differences in dimensions. They are also represented in Figure 4.4 after being resized to the same dimension to see more clearly the content. Each image represents one different class and has been drawn randomly. The order of the classes is the same on both figures, although the labels have been added only on the first figure.

Class Imbalance

While the researchers having built this dataset tried their best to have a similar number of images in each class, they had to deal with too much disparity in the datasets they managed to obtain to be able to reach this goal. As it did not impact their work too much, they decided to leave it as it is currently, leading to a great imbalance in the dataset. In particular, two classes, *janowczyk6_0* and *camelyon16_0*, contain together more than 50% of the total number of images in the dataset as can be seen in Table 4.2.

[5]Visually different, from a non expert point of view

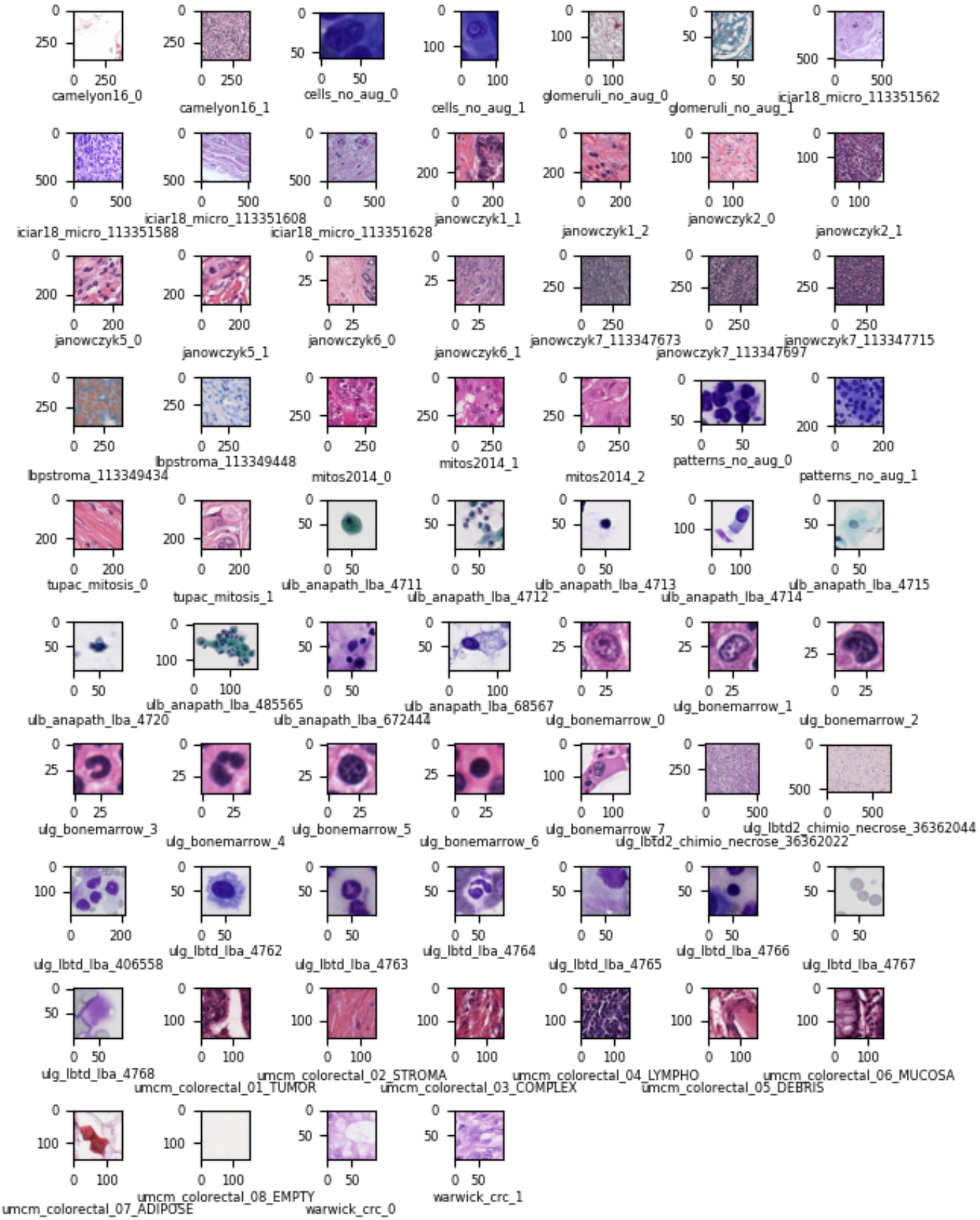

Figure 4.3: Sampled images in their original format of the dataset

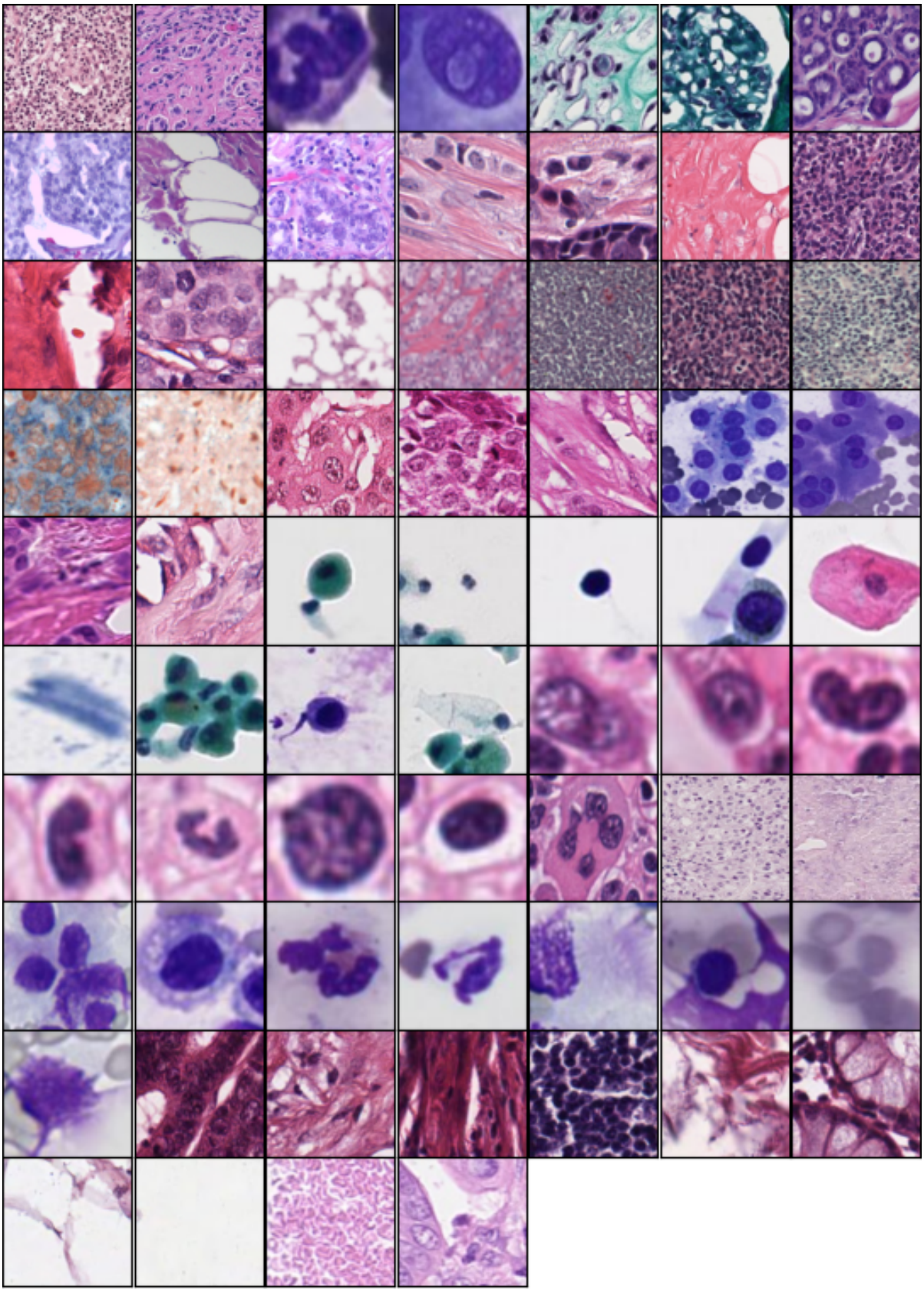

Figure 4.4: Sampled images from each class of the dataset

Consequently, the other 65 classes, having to share the remaining images of the datasets, have a very low images count. The division of those remaining images in the other classes can be

	Train	Test	Validation	Total
Camelyon16_0	218 938 - 34,56%	23 810 - 22,4%	25 697 - 26,73%	268 445 - 32,11%
Janowczyk6_0	160 790 - 25.38%	15 089 - 14,2%	22 791 - 23,78%	198 738 - 23,77%
Together	379 728 - 59,94%	38 899 - 36,6%	48 488 - 50,54%	467 115 - 55,89%
All	633 499	106 281	96 066	835 846

Table 4.2: Number of images and percentage for each of the 2 majority classes as well as together

found in Figure 4.5, where *camelyon16_0* and *janowczyk6_0* have been removed for readability and clarity reasons. As expected, those classes only contain a small number of images compared to *camelyon16_0* and *janowczyk6_0*. Furthermore, it can be seen that the other classes of the related projects (*camelyon16_1* and *janowczyk6_1*) belong to the remaining classes with the most images, being at least in the top 10 classes in terms of the number of images. Those two projects are at risk of taking more importance in the feature representations learned. On the other hand, several projects do not reach 1% with all their classes added like *ulg_lbtd_lba* or *ulg_bonemarrow*. *Umcm_colorectal* does not even manage to reach 0.01%. Those projects tend to not be able to meaningfully impact the feature representation such that they would be well represented. In Section 6.6, a study of the impact of that imbalance on the results is conducted.

Diversity

The diversity in the images is quite clear in Figures 4.3 and 4.4 with very different images depending on the class to which they belong. However, on top of that diversity, there is also some variance inside one same class, with some classes being much more variable than others. For example, in Figure 4.6 are represented 4 images that can all be found in the class *camelyon16_0*. Similarly, Figure 4.7 represents images that can be found in the class *janowczyk6_0*. All those images seem really different and yet belong to the same class. *Iciar18_micro_0* also has some variability as can be seen in Figure 4.8 though less than the 2 previous classes. There is also 'less meaningful' variability. If we take a look at images from *ulb_anapath_5* in Figure 4.9, we can see that the core element of the different images is very different, both in color and shape. However, the background of all the images is identical, which was less the case for the previous classes (usually common color schemes but different shapes of elements and intensity of colors).

Finally, there are the classes like *janowczyk7, 1 to 3* or *umcm_colorectal_5* whose content is quite similar, with the most visible difference being the colors, as can be seen in Figure 4.10 and Figure 4.11.

Due to that diversity, some images seem closer to images from other classes than from images of their own class. The two images on the left part of Figure 4.12 are coming from the class *Camelyon16_0* while the two at the right are coming from *Tupac_mitosis_0*. In Figure 4.13 are images that feel similar to the image right above them, in order from the classes *Janowczyk7_0*, *Camelyon_1*, *Janowczyk1_1*, *Tupac_1*.

It is interesting to note that the diversity inside a class is potentially linked to the size of the class. The largest classes tend to be more diverse than the smallest. This makes sense considering that they contain much more images and are thus less likely to have that many images really similar. At the opposite of that diversity though, there are also very similar, nearly identical images in each class. Indeed, some images only differ in some pixels because they represent the same original image but are cut at slightly different positions, with overlap.

This diversity inside the class is one of the motivations behind the use of unsupervised/self-supervised learning algorithms to reduce the impact that the labels would have on the feature representation. As the labels won't be used, two images of the same class but that do not

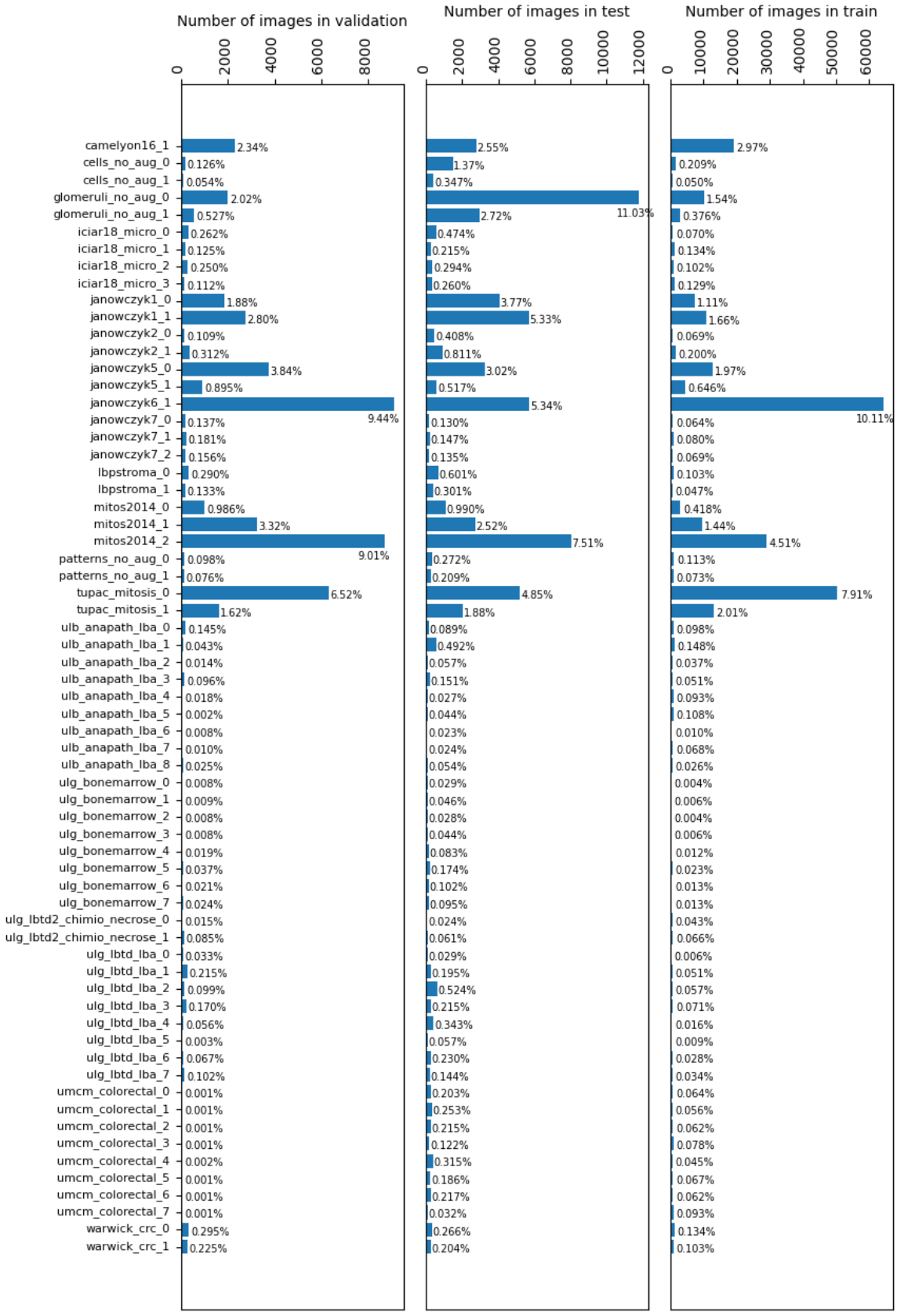

Figure 4.5: Images division per class per set

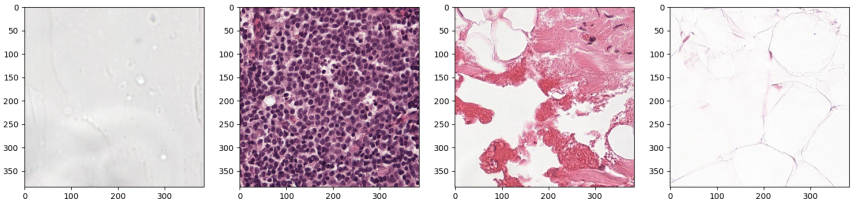

Figure 4.6: 4 different images belonging to *Camelyon16_ 0*

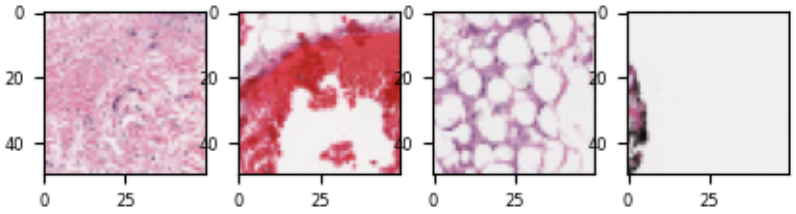

Figure 4.7: 4 different images belonging to *janowczyk6_ 0*

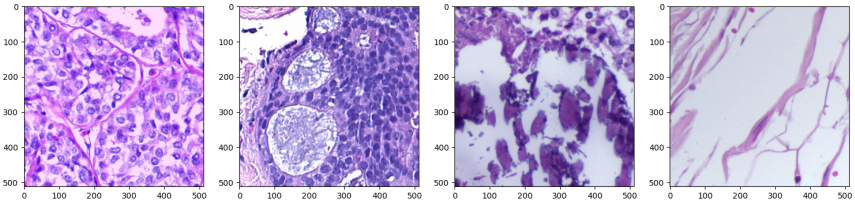

Figure 4.8: 4 different images belonging to *Iciar18_ micro_ 0*

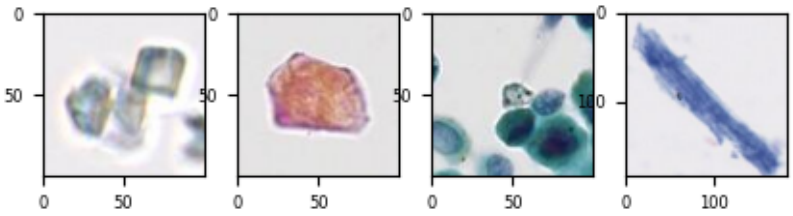

Figure 4.9: 4 different images belonging to *ulb_ anapath_ 5*

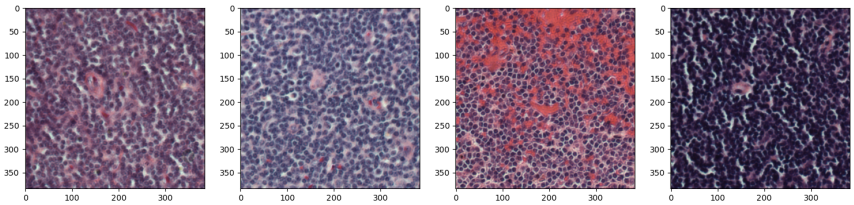

Figure 4.10: 4 different images belonging to *janowczyk7_ 1*

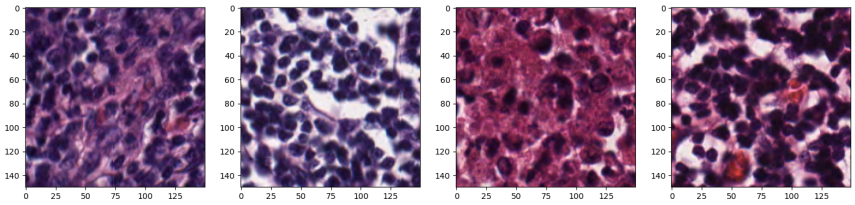

Figure 4.11: 4 different images belonging to *umcm_ colorectal_ 5*

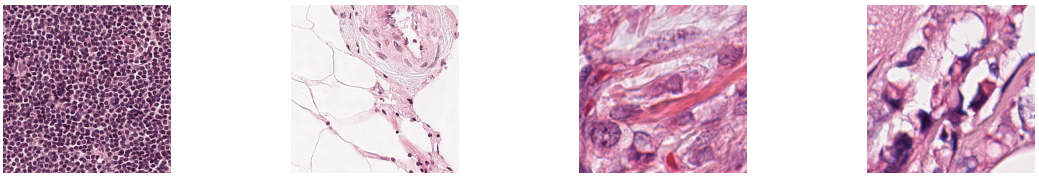

Figure 4.12: The two left images are from Camelyon_0, the two right images are from Tupac_mitosis_0

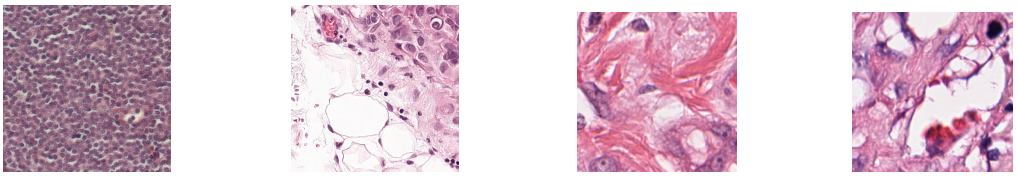

Figure 4.13: Images from Janowczyk7_0, Camelyon16_1, Janowczyk1_1 and Tupac_mitosis_1

look similar will not necessarily be considered similar by the model, removing a bias that, in a situation with such intra-class diversity, could be detrimental. It has especially motivated the use of the K-means algorithm.

Note that Figures [7.68 to 7.82] in the Appendix represent carefully selected samples of each class, chosen to represent the diversity present in each class. Each image has been chosen such that it would be different from the ones already selected for that class.

Another type of diversity present in this dataset is the difference in dimensions and shapes of the images. A first demonstration of that aspect can be observed in Figure 4.3 where we can see that the axis scales are not all the same and that some images are rectangular rather than square. This diversity is mainly between the classes rather than in the classes like in the previous one. Table 4.3 presents the mean width and mean height per class of the dataset with their standard deviations (the class names were shortened for the Table to fit). Several observations can be made.

- Diversity in a same project is small. Classes belonging to a same project have the same dimensions both in height and width in most of the cases. If there is a difference, there are three groups. One where it is frequently around only 25 pixels or less, as in *Glom*, *Lbps*, and *Ulb*. The second where the biggest differences are seen, 100 in *Patt*, 70 in *Ulg_lbtd2*. And finally the last, where most classes of the project have similar dimensions except one where the difference is huge, more than 100 pixels, in *Ulg_bone* and *Ulg_lbtd*.
- Presence of diversity between similar projects. Similar projects do not tend to have similar image dimensions as can be seen with the *Jano* or with *patt/cells*.
- Diversity of shape is small. Most classes are composed of (perfectly or almost) squared images, with only 4 out of the 67 with a rectangular form (2 of *Patt* and 2 of *Ulg_lbtd2*).
- Diversity between projects is big. Projects tend to have very different image dimensions from one another, with the smallest dimension being 34.2x34.2 and the biggest $542.7x492$, more than 500 pixels of difference.
- The mean width is 203.14 and the mean height is 200.3.
- In the majority of the classes, the diversity in a class seems small with most of them having a std of 0 or close (<5). In the others, browsing through the images composing them confirms the variety, with images of the same class having very different dimensions. For example, in *Cells_0*, the biggest image is (1754, 1456) while the smallest is (19, 19)

Name	(Width + std, Height + std)	Name	(Width+std, Height+std)
Cam_0	(384 ± 3.33, 384 ±3.33)	Ulb_4	(129.4 ±51.63, 128.66 ±50.68)
Cam_1	(384 ± 0, 384 ±0)	Ulb_5	(124.5 ±87.5, 123 ±91.7)
Cells_0	(83.9 ±81.5, 81.4 ± 72)	Ulb_6	(121 ±37.7, 121.2 ± 37.3)
Cells_1	(82.4 ± 29.9, 80.4 ± 29.8)	Ulb_7	(100.4 ±3.9, 100.3 ±2.98)
Glom_0	(120.5 ± 185.6, 121.5 ± 177.4)	Ulb_8	(124+5 ±32.7, 128.2 ±35.31)
Glom_1	(142.7 ±54.2, 145.5 ±54.4)	Ulg_bone_0	(49.25 ±6.1, 49.2 ±6.1)
Iciar_0	(512 ±0, 512 ±0)	Ulg_bone_1	(44.2 ±5.31, 44.2 ±5.29)
Iciar_1	(512 ±0, 512 ±0)	Ulg_bone_2	(44.6 ±5.78, 44.6 ±5.76)
Iciar_2	(512 ±0, 512 ±0)	Ulg_bone_3	(42.2 ±5.25, 42.2 ±5.25)
Iciar_3	(512 ±0, 512 ±0)	Ulg_bone_4	(42.5 ±5.04, 42.6 ±5)
Jano1_0	(250 ± 0, 250 ±0)	Ulg_bone_5	(39.5 ±5.12, 39.5 ±5.14)
Jano1_1	(250 ±0, 250 ±0)	Ulg_bone_6	(34.2 ±5.71, 34.2 ±5.73)
Jano2_0	(200 ±0, 200 ±0)	Ulg_bone_7	(150 ±35.65, 150 ±35.71)
Jano2_1	(200 ±0, 200 ±0)	Ulg_lbtd2_0	(542.7 ±154.53, 492 ±136.27)
Jano5_0	(250 ±0, 250 ±0)	Ulg_lbtd2_0	(469.7 ±190.45, 404 ±167.23)
Jano5_1	(250 ±0, 250 ±0)	Ulg_lbtd_0	(230.15 ±120.98, 231.33 ±147.66)
Jano6_0	(50 ± 0.06, 50 ±2.67)	Ulg_lbtd_1	(100.12 ±1.48, 100.15 ±1.76)
Jano6_1	(50 ±0.07, 50 ±0.15)	Ulg_lbtd_2	(100 ±0, 100 ±0.19)
Jano7_0	(384 ±0, 384 ±0)	Ulg_lbtd_3	(100 ±0, 100 ±0)
Jano7_1	(384 ±0, 384 ±0)	Ulg_lbtd_4	(103.3 ±7.11, 104.8 ±7.97)
Jano7_2	(384 ±0, 384 ±0)	Ulg_lbtd_5	(100 ±0, 100 ±0)
Lbps_0	(448 ±99.14, 448 ±99.14)	Ulg_lbtd_6	(101.6 ±9.62, 101.96 ±10.91)
Lbps_1	(423 ± 117.39, 423 ± 117.39)	Ulg_lbtd_7	(124 ±67.4, 126.3 ±75.6)
Mitos_0	(323 ±0, 323 ±0)	Umcm_0	(150 ±0, 150 ±0)
Mitos_1	(323 ±0, 323 ±0)	Umcm_1	(150 ±0, 150 ±0)
Mitos_2	(323 ±0, 323 ±0)	Umcm_2	(150 ±0, 150 ±0)
Patt_0	(212.19 ±212.8, 176,6 ±156.59)	Umcm_3	(150 ±0, 150 ±0)
Patt_1	(306.55 ±262.42, 260.8 ± 200.86)	Umcm_4	(150 ±0, 150 ±0)
Tupac_0	(250 ±0, 250 ±0)	Umcm_5	(150 ±0, 150 ±0)
Tupac_1	(250 ±0, 250 ±0)	Umcm_6	(150 ±0, 150 ±0)
Ulb_0	(111.4 ±21.49, 111.2 ± 21.73)	Umcm_7	(150 ±0, 150 ±0)
Ulb_1	(100 ±0.72, 100 ±1.02)	Warw_0	(100 ±0, 100 ±0)
Ulb_2	(100 ± 0, 100 ±0)	Warw_1	(100 ±0, 100 ±0)
Ulb_3	(125 ±38.6, 125.5 ±37.06)		

Table 4.3: Mean width and height per class

and the mean (83.9, 81.4) is different from the median ((67, 67)), indicating a large range, confirmed by the values of the standard deviations.

This diversity might impact the results as all images will be resized to a constant size and thus the 'quality' of the input will vary given the class.

4.3.3 Image preparation

An important step in CV systems is the preparation of the data. In order to make the model robust to small changes in the data, it is customary to apply transformations to it before feeding it to the network. In this work, the following data transformations[6] are applied to the data in the training part of the process and in the majority of the scenarios (the only exception is when using Augmented Learning as it will be discussed in the related section). For indexing and retrieval, only the normalization is applied.

- Random flips, both vertical and horizontal, with $p = 0.5$.
- Color alterations by changing the chromaticity of the image (saturation ($p = 0.2$) and hue ($p = 0.1$)). The hue represents the type of color (values of the RGB components) while the saturation is the 'purity' of the color. When changed, the new value is randomly picked inside an interval defined by the min and max value of the image considered.
- Normalization, with each channel being applied a different mean and standard deviation, with the means = [0.485, 0.456, 0,406] and the standard deviations (std) = [0.229, 0.224, 0.225] which are obtained from the analysis of the ImageNet dataset.

Note that in the case of the normalization step, the means and standard deviations were kept unchanged even though it might have been better to use values obtained from this dataset as the nature of the images is completely different from the nature of the images of Imagenet. However, for comparison purposes with the previous work, they were left untouched. Figure 4.14 represents one image and its different transformations.

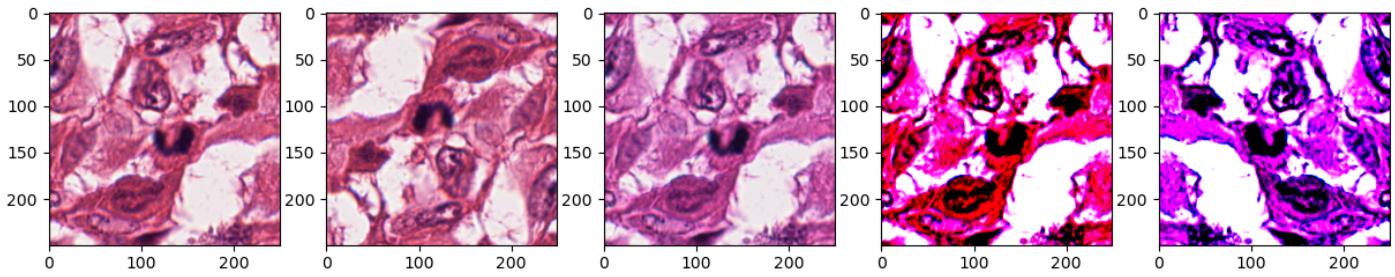

Figure 4.14: From left to right: Original image - Image with random vertical and horizontal flips - Image with random change in saturation and hue - Normalized image - Image with all previous transformations applied

In addition to the use of data transformations for robustness, their usage is actually required to resize the input to a constant size acceptable by the networks used in this work. Indeed, due to the architectures used in this work, the images must be of similar and constant dimensions equal to 224x224x3, as they are kept colored (RGB).

It is particularly important here because the images all have different dimensions and even shapes as explained earlier. 224x224 has been chosen because it is a common choice in CV for resizing images and it is close enough to the mean width and height of the histopathology dataset used. To resize the images, a squared area of size randomly picked between 80% and 100% of the original image is selected at a random position in the image. Randomness is again used to increase robustness to variation[7]. This area is then resized to 224x224.

[6] Those transformations are obtained from S. Defraire's Master thesis [Defraire, 2021] and kept unchanged.
[7] Only for training. In indexing and retrieval, the entirety of the image is kept

Chapter 5

Methodology & Process

This section presents the methods and elements that have been investigated, implemented, and tested as part of this work following their identification in the second chapter. All the methods presented in this chapter are available for use in the implementation related to this thesis.

This process for CBIR is not fixed, it is built such that it is modular. As explained in Chapter 3, CBIR is composed of several distinct parts. The methodology presented here treats those parts as if they were independent, making it then possible to change the elements/concepts of one part while keeping the other unchanged. It particularly focuses on the **first part, i.e. feature extraction**, while only offering one option for the other part, i.e. search and similarity. The modularity is present both in the selection of the main structure (e.g. backbone neural networks) and in the choice of less consequent structures (e.g. composition of the training data) or parameters (e.g. feature number).

The CBIR framework is designed as displayed in Figure 5.1. The organization of this chapter

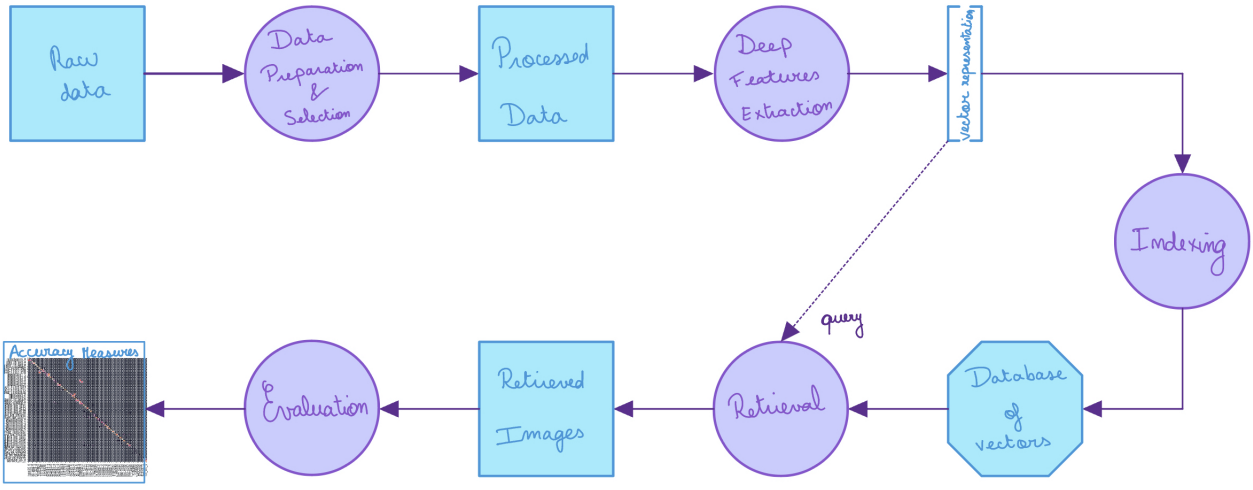

Figure 5.1: CBIR Process

follows the steps of the framework except for the first step, *Data preparation and selection*, which is explained in Chapter 4. It starts with the options for the *Deep features extraction* step in Section 5.1. Then it presents the *Indexing* step in Section 5.2 before explaining the step *Retrieval* in Section 5.3. It continues in Sections 5.4 and 5.5 with the *Evaluation* step, with the protocols and metrics. The final section is a quick description of the implementation in Section 5.6, with a more detailed one in an annexed file.

5.1 Features extraction - Design

Several deep learning architectures are tested to extract the most representative features for the histopathology dataset. On top of the architectures, several concepts impact the way the data is provided to them. The main division between the approaches investigated in this work is the type of learning with the first approaches explored being trained in a supervised fashion while the latest are trained in a self-supervised way.

5.1.1 Supervised learning

The models and concepts of this section rely on the use of labels to obtain the final feature extraction. Two types of architectures are used, CNNs and Transformers, presented in Parts A and B. The architectures have to be combined with a training concept, described in Part C.

A. CNN Architectures

Three CNN architectures were selected for various reasons and described in Chapter 2.

- Resnet50: Chosen because it is one of the most popular CNNs, winner of ILSVRC in 2015 and frequently used in papers as a benchmark. It is also a model used in [Defraire, 2021] and the one that provided the best results.
- Densenet121: Chosen because it is designed as a response to an issue noticed in the ResNet and as such, hoped to achieve better results. It is also one of the models used in [Defraire, 2021].
- EfficientNet_B0: Chosen because it is a more recent architecture than the two previous and it is supposed to be more computationally efficient than them, which, given the amount of data to treat here, is an important advantage.

There is one more backbone model that is considered which is *KimiaNet*. As explained previously, KimiaNet is a DenseNet121 that has been trained from scratch on medical images (2^{nd} dataset presented in Chapter 4). It has been chosen because it is the only architecture that is pre-trained on the same kind of data as the data of interest.

All those architectures are made for classification on Imagenet_1K (1^{st} dataset in Chapter 4), as such they have as their final layer a fully connected with 1000 outputs. To use them for feature representation, this last layer was removed and replaced by a fully connected layer with 67 (number of classes of the histopathology dataset) outputs.

B. Transformers Architectures

As for the CNNs, several architectures were explained in Chapter 2 for the Transformers.

- ViT: Chosen because it is the first Transformer model adapted and used for vision which makes it the original/baseline model.
- DeiT: Chosen because it is a Vision transformer that has been designed to be data efficient, which, again, given the size of our dataset is a non-negligible advantage.

- CvT: Chosen because it combines convolution elements with a Transformer architecture which makes it at first sight particularly adapted to vision tasks.

Again, all those architectures are available as classification models and need to be adapted for feature representation. This is done the same way as for the CNN, by removing their last layer and replacing it with a FC of 67 outputs.

C. Concepts

Now that the backbone architectures have been presented, it is the turn of the concepts used to train them. As explained in Chapter 2, feature extraction/representation learning is a very distinct application of DL and requires different training methods than those commonly used in classification for example. Three of such concepts, *Deep metric learning* (DML), *Deep Ranking* (DR), and *Contrastive Learning* (CL), have been explained previously. They will all be used in this implementation as a way of training the supervised networks. Note that they will also be of use for some of the unsupervised methods as will be explained in due time.

Deep Metric Learning

DML allows the training of a network such that it takes into consideration the distance of the model output to a predefined goal or the similarity between elements. It is an 'overall' concept that comprises a lot of different structures as explained in the theoretical part. A training structure is considered to be DML if it includes 3 elements, a feature extraction model, a ranking loss, and a sampling strategy.

The feature extraction models have been described in Parts A and B of this section.

The losses have been described in Section 2.4.1. Three were from the previous thesis implementation, [Defraire, 2021], and kept unchanged: Custom Margin Loss[1] (It combines the margin loss with the contrastive loss given the value of a probability given as a parameter.), ProxyNCA++, and Normalized Softmax. On top of those three, the SoftTripleLoss is also tested.

Finally, for the data sampling strategy, simple balanced sampling is used. n samples per selected class will be taken until all classes of interest have been browsed. Then, one sample per class will be retrieved until the batch is full given the chosen batch size[2].

Deep Ranking

DR is a kind of specialization of DML that limits the options in each of its three categories to a specific type. A model trained using DR has: a backbone architecture accompanied by two shallows networks, repeated 3 times, a triplet Loss function, and a triplet sampling strategy. While several options could be considered for the backbone architectures, to respect the model developed in the paper presenting DR [Wang et al., 2014], only the CNN architectures are used with DR.

The loss function consists of the basic triplet loss function, also used in said paper and described in Section 2.4.2.

Regarding the sampling, for simplicity and computational efficiency, only random sampling was implemented. It was not possible to use the sampling strategy described in the paper as our dataset did not dispose of the relevance score needed to perform the computations leading to the more sophisticated sampling. It would also not have been possible to obtain those scores as part of this master thesis.

[1] Based on the implementation of Deep Metric Learning Baselines, Github.

[2] Based on the implementation furnished in Deep Metric Learning Baselines, Github which is based on [Roth et al., 2020]

Note that while the triplet loss and sampling, or the addition of the two shallow networks, could also be used with the two other concepts, they will be strictly reserved for the DR concept. There will thus be no model using one without also using the other.

Contrastive Learning and Non-contrastive learning

Contrastive learning (CL) learns about the images by contrasting them with similar or dissimilar images of the dataset by pair or triplet. The methods implemented here are mostly pair-based. In the following, CL will thus names approaches that use: A Twin network or Siamese Network, a pair-based loss, and a pair sampling strategy.

For the backbone networks, it can be any of the ones presented in Parts A and B. Not all will be tested with CL though as the number of models would be too important to properly investigate everything (see Section 6.1).

Regarding the losses, they are the ones that have been described in Section 2.4.3: CosineEmbeddingLoss, BCE loss, Contrastive Loss, InfoNCE loss[3]

Finally, for the sampling strategy, it will be the same as for DR except that only one of the positive and negative samples will be kept. Both are drawn randomly, but when the index of the anchor is even, the negative sample is returned with the anchor, otherwise, the pair is composed of the positive sample. This proportion was selected to have 50% of the pairs dissimilar and 50% similar but different proportions can be chosen. It was not tuned at all in this work.

Regarding Non-Contrastive Learning, a true NCL architecture has not been implemented for supervised learning. Without the results of the contrastive methods first, it was thought not pertinent. Instead, a study of the necessity of the negative pairs *without* changes to the architectures is realized to look at the dimensional collapse effect. This is done with two losses, the contrastive loss (the most classic one) and the NT-Xent loss (as it seems to grant more importance to positive pairs).

D. Summary of supervised learning

Figure 5.2 shows how to obtain an operational feature extractor using supervised learning as implemented in this work. In short, one of the 3 concepts: DR - DML - CL/NCL, must be combined with one backbone architecture, either a CNN: ResNet - DenseNet - EffNet - KimiaNet or a transformer: ViT - CvT - DeiT. In addition, the concepts have different options to consider for the loss function.

5.1.2 Self-supervised Learning

A. Auto-encoders

Auto-encoders were the first unsupervised architecture thought of for this master thesis. Their principle of learning an image representation by trying to reconstruct the image, so just by basing themselves on the content of the image, made them the seemingly best-fitted architecture for our purpose, especially given the diversity of the data, between and inside the classes. AE is a concept that can be implemented using very different architectures, losses, parameters,... While AEs have been used in articles for CBIR, there was never a single framework used. Instead, each article introduced its own structure. As a consequence, there were plenty of implementations available that needed to be sorted according to their strength and use in this thesis. Finally, three implementations of AE have been selected, each with its strengths / drawbacks[4].

[3]This loss requires the use of positive pairs. It will thus be used either with triplet pair or with NC learning. It cannot be used for pair-based CL as it would lead to some pairs being exclusively negative.

[4]Based on the following websites: ImageNet-autoencoder, github, pytorch-example and geekForgeek

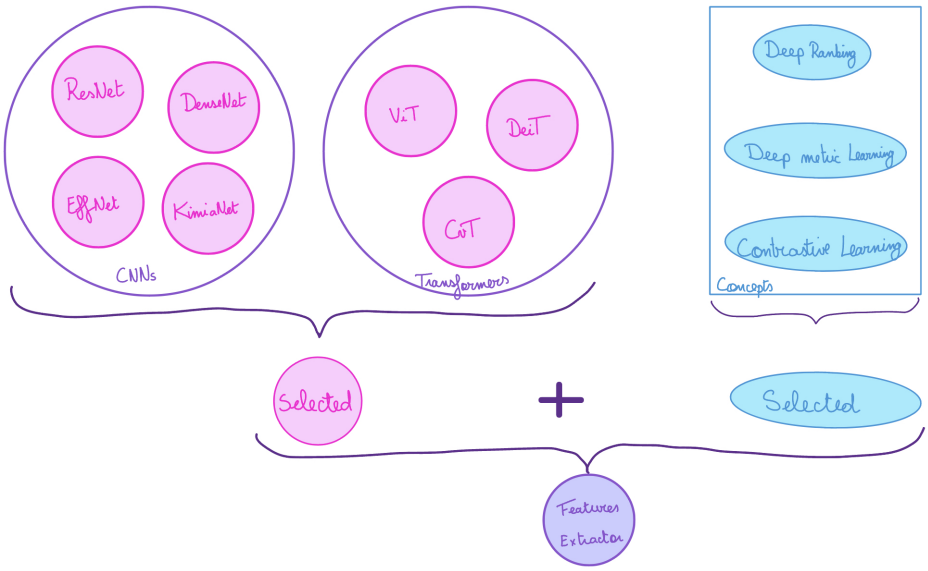

Figure 5.2: Summary of Supervised features extraction

Implementation 1: Complex, Resnet-based

This implementation[5] offers 8 auto-encoders, 4 are versions of VGG and 4 of ResNet, coded to be used on ImageNet or other datasets similar to it. The loss used is the Mean squared error loss, which is one of the most classical reconstruction losses.
This implementation was selected because it had the particularity of offering ResNet and VGG structure as the backbone of the AE. An effective, known-to-be successful in vision tasks, backbone architecture was a strong requirement at the start of the research on AE as it was thought to lead to better results than a newly, built-from-scratch architecture. Furthermore, it even offered pretraining on several datasets, including ImageNet, for one of the 8 versions. Regarding the quality of the implementation, there were no problems running the code and the architectures respect the description of the official ResNet and VGG networks. However, it is not based on any paper and it does not seem to be much known.

Different experiments were realized with that implementation, to try and achieve the best results possible, or fix issues that previous experiments had raised (those results will be discussed in more depth in Chapter 6).

1. VGG-16 as the backbone, kernel sizes as well as input/output dimension of each layer modified such that the number of features in the latent space would be equal to 128. Trained on the entire training dataset.

2. Basic VGG-16 and VGG-11 implementations, not modified. Trained on the entire VGG-16 dataset.

3. Testing of the basic implementations with ResNet18 as the backbone architecture, not modified. Trained on the entire training dataset. (a) is using a cosine scheduler as offered by the implementation from whose the model has been obtained and (b) is trained using an exponential scheduler in the framework built for this work.

4. VGG-16 architecture, pre-trained on ImageNet, no modifications of the architecture. Trained on the entire dataset.

[5]Horizon2333's Github page

5. Basic VGG-16 architecture, no modification. Trained on the entire dataset except for the Janowczyk6_0 class.

6. Basic VGG-16 architecture, no modification nor pretraining. Trained on only one project and on only one class to see the learning done by the autoencoder.

7. Modification of the AutoEncoder class of the Resnet model to include the three additional layers present in the ResNet classification model, to reduce the number of features of the latent space without modifying the parameters of the layers. The backbone architecture is the ResNet-50, the number of end features is 1000 (kept the same parameters as in the ResNet classification model) and it was trained on the entire dataset.

Note that with all experiments, the loss used was the Mean squared error (MSE) loss, between the original and the reconstructed images.

Implementation 2: Variational AE

Given the poor results obtained with the previous implementation (see Chapter 6), other implementations, of simpler backbone architecture, were sought out. This implementation[6] is one of the two that were kept and implemented in the framework. It is a basic implementation of a variational autoencoder. The appeal of this implementation first resides in its seriousness (made by Pytorch) and can be thus deemed trustworthy. Furthermore, it was a different type of AE (variational) that could prove useful for this task, and the number of end features was by default smaller than with the previous AE. On the other hand, variational AEs are usually used for data generation which is not the end goal here. It is also not available pre-trained and it is built for the Mnist dataset which has a very different image format compared to our dataset (28x28x1 against 224x224x3). It may thus not lead to the expected results.

The implementation is based on the paper 'Auto-Encoding Variational Bayes ', [Kingma and Welling, 2022]. It uses a Multi-Layer Perceptron to learn the parameters of a multivariate Gaussian Distribution from the input $\mathbf{x}$. Hence, the outputs are the mean μ and std σ of the approximated Gaussian $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$. Mathematically,

$$\log q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \log \mathcal{N}(\mathbf{z}; \mu^{(i)}, \sigma^{2(i)}\mathbf{I})$$

From those two outputs, it generates a random variable $\mathbf{z}$ through the use of what they call the 'reparametrization trick':

$$\mathbf{z}^{(i,l)} = g_\phi(\mathbf{x}^{(i)}, \epsilon^{(l)}) = \mu^{(\mathbf{i})} + \sigma^{(\mathbf{i})} \circ \epsilon^{(l)}$$

where μ and σ are obtained from the MLP, given the input $\mathbf{x}$ and $\epsilon \sim \mathcal{N}(0, I)$. This random variable is then fed to the decoder part of the architecture which is again two simple FC layers such that it returns an output of same dimension as the input, so in this case, the reconstructed image.

This reconstructed image is finally compared to the original one to compute the loss. This loss is composed of two terms. The first one is given by the binary cross entropy between the original and the reconstructed images, which acts as the reconstruction error. The second term is given by the Kullback-Leibler (KL) divergence between the approximation of the posterior and the true distribution, which makes it a regularization term. The loss is computed using the following formula:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) \approx \frac{1}{2} \sum_{j=1}^{J} (1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) + \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$$

[6] Pytorch-examples

with $\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$ obtained from the model, the first term the KL divergence and the second is the cross entropy.

Note however that the Pytorch implementation deviates slightly from the original paper by including ReLUs among other changes.

For this implementation, 3 experiments were designed:

1. Keep the original implementation and modify the way of viewing the inputs to fit the implementation. Each image is fed to the model by pieces of 28x28 dimension, flattened. The outputs of each piece are then concatenated. The dimension reductions are as follows: $784 \rightarrow 400 \rightarrow 20$. This leads to $\frac{224^2 \cdot 3}{28^2} \cdot 20 = 192 \cdot 20 = 3072$ features per image.

2. (a) Modify the dimensions of the layers of the original implementation such that the images are fed in their original dimension (224x224x3) but flattened: $224x224x3 \rightarrow 400 \rightarrow 20$. It leads to 20 features. (b) Add one more layer to diminish the impact of the original decrease in dimensions: $224^2 \cdot 3 \rightarrow 784 \rightarrow 400 \rightarrow 20$, also ending with 20 features. (c) Change the reductions to end up with the usual number of features: $224^2 \cdot 3 \rightarrow 1500 \rightarrow 750 \rightarrow 128$.

3. Add convolutional and deconvolutional layers and modify the dimensions of the original layers of the original implementation such that the images are fed to the model in their original dimensions and not flattened: $224x224x3 \rightarrow^{\text{convolutions}} 128x56x56 \rightarrow 1000 \rightarrow 128$.

Implementation 3: Contractive

The third and last AE implementation uses a simple MLP as the backbone architecture like the previous one. It was likewise motivated by the intention of going with a simpler architecture compared to the one used in the first implementation. This architecture comes from the paper: 'Contractive auto-encoders: Explicit invariance during feature extraction'[Rifai et al., 2011][7]. This implementation was chosen because it was focused on feature extraction rather than image reconstruction, unlike many other implementations. It is said to be able to retrieve more robust features to represent an image, i.e. features that would be less sensitive to small variations. However, similarly to the previous one, it is not pre-trained and was built for the Mnist dataset which does not have images of the same dimensions as the dataset of interest here.

To reduce the impact of the variations in the data or, as said in the paper, to reduce the sensitivity of the model to input data x, one more term, a regularization or penalization term, is added to the reconstruction loss. This added term is the *Frobenius norm of the Jacobian* $J_f(x)$ *of the non-linear mapping.* Mathematically:

$$||J_f(x)||_F^2 = \sum_{ij} \left(\frac{\partial h_j(x)}{\partial x_i} \right)^2$$

with x the input, f the encoding function and h the latent space representation. This term contracts the feature space, i.e. ensures that the points in the latent space are closer than the points in the original space. It helps against the small variations of the data by giving points closer to one another similar outputs. This makes it generalize better as well as helps against overfitting.

The final objective function is given by the combination of this term with the reconstruction loss:

$$\mathcal{J}_{CAE}(\theta) = \sum_{x \in D_n} (L(x, g(f(x))) + \lambda ||J_f(x)||)_F^2)$$

[7] And is presented in This website.

For the implementation, it uses the MSE loss as the reconstruction loss. The second term is computed using the following formula:

$$\lambda \cdot \left(\sum_{i=1}^{d_h} (h_i \cdot (1 - h_i))^2 \cdot \sum_{j=1}^{d_x} W_{ij}^2 \right)$$

which is given in the paper as a simpler expression once the sigmoid function is used as activation.

Two important elements need to be highlighted. The original implementation is not completely coherent with respect to the paper. It uses ReLUs instead of sigmoid, without changing the loss formula. Furthermore, it uses the single-layer MLP formula while using several layers for the model. Unfortunately, these adaptations with respect to the model described in the paper were only discovered after the design of the experiments and after most models were trained. Nevertheless, the model based on the adequate loss was then implemented and tested to see the potential impact of these changes. For that, the Frobenius norm of the Jacobian corresponding to the specific network was computed, using the chain rule for derivatives. The details of the computation can be found in Appendix B. This new implementation gives slightly better results than the original one as will be discussed in Chapter 6. However, it is quite impossible to apply to all the designed experiments as it would require far too much memory to store the weight matrices. It was thus decided to keep the simplified original implementation to study the effect of the experiments while presenting the results of the more complex implementation only for the first experiment to illustrate the difference in range.

Therefore, after testing the implementation based on the adequate computation of the Frobenius norm, considered as Experiment 0, the following experiments are realized

1. The original architecture is kept: $784 \rightarrow 64 \rightarrow 32 \rightarrow 16$, It leads to $\frac{224^2 \cdot 3}{28^2} \cdot 16 = 192 \cdot 16 =$ 3072 features.
2. The original input dimension is kept but smaller steps are made for the reduction: $784 \rightarrow 400 \rightarrow 200 \rightarrow 16$, 3072 features
3. The original input dimension is kept but the number of features is higher: $784 \rightarrow 400 \rightarrow 200 \rightarrow 50$, 192 * 50 = 9600 features
4. The input dimensions are changed to fit our dataset, and the rest of the architecture is kept as the original: 224^2*3 $\rightarrow 64 \rightarrow 32 \rightarrow 16$, 16 features.
5. Same as the previous but with more features at the end: $224^2 * 3 \rightarrow 512 \rightarrow 256 \rightarrow 128$, 128 features
6. Same as previous, but with an even higher number of features: $224^2 * 3 \rightarrow 3000 \rightarrow 1500 \rightarrow 750$, 750 features.

B. K-Means

The second architecture implemented and tested in this thesis that is trained in an unsupervised fashion is using clustering as its base. Due to the diversity present inside the classes themselves of the dataset, as discussed in Chapter 4, an application of the technique of K-means clustering was applied to the dataset, directly to the pixels of the images. The purpose of this operation was to see how would the images be grouped based on their content only.
Afterward, it was thought that those new groups created using K-means clustering could be used as labels for the previously introduced supervised models. The K-means method is then composed of two steps. First, the K-means model is trained on the histopathology training dataset

and new labels are retrieved for each image composing it. Second, those images and their new labels are used to train one of the supervised feature extraction models described in 5.1.1.

Two different configurations of the K-means models are being proposed, based on the number of clusters selected.

- $n\_clusters = 67$. This was selected to make a parallel with the original labels. This allows to look if the original 67 classes are more or less making the 67 new or if they are all split and regrouped differently. An analysis of this split will be presented in the next chapter.
- $n\_clusters = 10$. This number was obtained through the use of the elbow technique, with 10 being the last number with the highest improvement as shown in Figure 5.3.

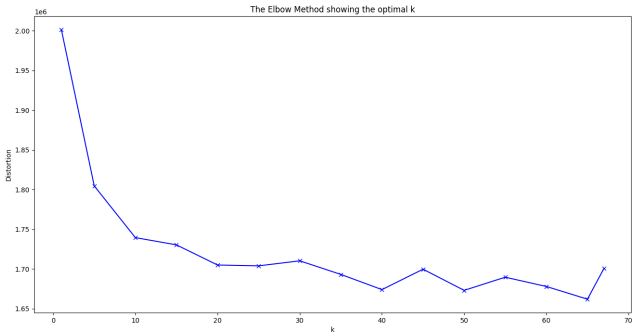

Figure 5.3: Elbow Plot for K-means clustering

C. Augmented Contrastive Learning

These two methods, Augmented Contrastive Learning and Augmented Non-Contrastive Learning (ACL and ANCL) are the last way used in this thesis to train the framework in a self-supervised fashion.

ACL is contrastive learning with a different type of sampling methods. In ACL, the positive sample of the pair is obtained by applying data transformations to it. The negative sample is randomly selected from all other images of the dataset, including the images of the same class as the sample image under consideration. In the manner of applying the transforms to the samples, different approaches are available as was discussed in Chapter 2[8]. Those approaches are the following:

- Custom transforms and unique application. The transforms are applied *only* to the anchor to form the positive sample. The anchor and the negative sample are used in their original aspect. The transforms in question are the following and have been selected arbitrarily taking into account the usual transforms used in similar projects.
  - Random vertical and horizontal flips, with $p = 0.5$.
  - Color changes in saturation, brightness, contrast, and hue, with $p = 0.4$ each.
  - Gaussian blur with probability 0.5 and random grayscale with $p = 0.05$.
  - Random rotation between 0 and 360 degrees.

[8]based and inspired by the paper A Framework For Contrastive Self-Supervised Learning And Designing A New Approach [Falcon and Cho, 2020] and its related blog

  - Resizing to 224^2 after random cropping of an area between 80 and 100% of the original image.
  - Normalization with ImageNet means and stds.

- Custom transforms, AMDIM application: The transforms of the previous points are reused but this time applied to the 3 images[9]. Hence, the anchor is no longer the original image but another version of it, like the positive sample.

- AMDIM transforms[10], AMDIM application. The following transforms are used:
  - Random horizontal flip with probability 0.5
  - Resizing to 224^2 after randomly selecting an area between 8 and 100% of the original one.
  - Random color changes in brightness, contrast, and saturation with $p = 0.4*0.8 = 0.32$ and in hue with $p = 0.1 * 0.8 = 0.08$.
  - Random grayscale with $p = 0.25$
  - Normalization with ImageNet means and stds.

- SimCRL transforms[11], AMDIM application. The following transforms are used:
  - Random horizontal flip with $p = 0.5$.
  - Resizing to 224^2 after randomly selecting an area between 8 and 100% of the original one.
  - Random color changes in brightness, contrast, and saturation with $p = 0.8*0.8 = 0.64$ and in hue with $p = 0.2 * 0.8 = 0.16$.
  - Random grayscale with $p = 0.2$ and Gaussian Blur with kernel size of 23 and $p = 0.5$.

Note that the CPC approach was not selected as it didn't seem appropriate to divide the images in patches given their sizes. The other approaches were chosen because of their proposed different application techniques and different choice of transforms, with the two last being carefully selected and proposed in papers.

For the feature extractor and the loss function, all options presented for supervised Contrastive Learning can be used.
On top of those options, for NACL, a complete framework, BYOL, is also implemented and tested. There is also one model that will just show the impact of using only negative pairs, without related architectural changes, similarly to the experiments explained in the supervised Contrastive learning section.

5.2 Indexing

This section describes the indexing part of a CBIR framework, the insertion in a chosen database of the vectors of the images of interest. In this work, the images added to the database are most of the time the ones present in the indexing set. For some time trials and to check the efficiency of the framework for a higher number of indexed images, the training data and the test data are indexed.

[9]No mention of the treatment of the negative image is made, by default the transform will also be applied to it.
[10]Based on Pytorch lightning bolt - Self-supervised models
[11]Based on Pytorch lightning bolt - Self-supervised models

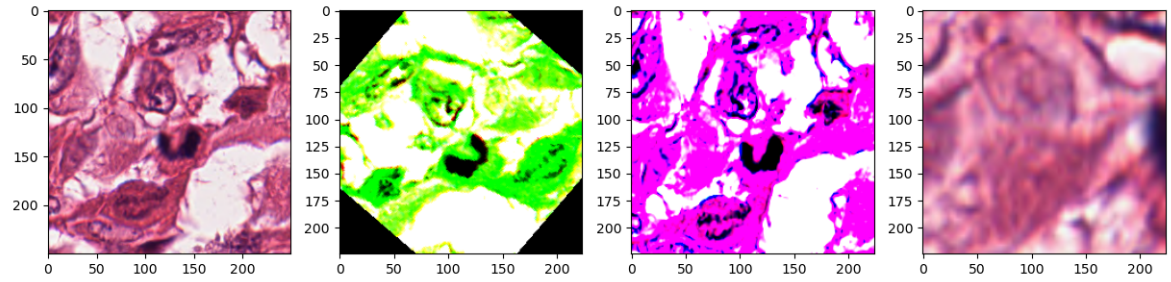

Figure 5.4: Original image -with Custom transforms - with AMDIM transforms - with SimCRL transforms

The indexing of the images uses the two structures described in Chapter 3: FAISS and Redis. As explained, FAISS is a structure developed for similarity search between vectors while Redis is an in-memory database. Both of those elements had already been used in [Defraire, 2021]. After further research and given the quality of their fit, it was decided to keep both of them in this work as well.

FAISS

While FAISS's main interest is for the next section, i.e. search, this functionality is only possible thanks to its indexing structures. As discussed in Chapter 3, FAISS offers different indexing methods that have a direct impact on the type of search performed. Due to their important link with the search methods, those indexings will be discussed in the next subsection. What is of interest here is the access to the structure and its information.

FAISS stores the vectors by attributing them two index numbers. The first one is referred to as the FAISS index. It is an index set by FAISS and uniquely accessible by FAISS. Hence the need for the second index number which is referred to as the 'external index'. This index is used to access externally the vectors stored in FAISS. FAISS keeps a mapping between the external indexes and the FAISS indexes.

While it may seem counterintuitive to use Redis in addition to FAISS as FAISS could be used alone, it was not possible in this case for two reasons.

- Absence of string indexes. FAISS only offers integers as indexes which makes it impossible to keep the link between the vectors and the images' names, crucial for the evaluation of the framework. Be it for the quantitative or the qualitative results, the original image must be retrieved.
- Difficult access to vectors. FAISS does not seem to have a direct, efficient, and *reliable* way to retrieve all vectors indexed in its structure. There are ways to do it but they are neither efficient nor reliable, especially in the case of removal of vectors.

For those two reasons, Redis was selected as an additional database.

Redis

Redis is used to conserve the mapping between FAISS external ids and the filenames as well as for retrieving the vectors saved in FAISS. It saves that information using a Key-Value structure

(or dictionary). It is the default structure though others can be used and one in particular, the *Sorted index and hashing* has been an envisaged option to replace Key-Value.

The mapping between the filenames and external ids is the main mapping stored in FAISS, with both the filename and the ids being registered as keys to allow efficient access (hence, each mapping is stored twice in Redis). However, when the K-means method is employed, another type of mapping is used. This mapping still links the filenames to the external ids and vice-versa but this time, when retrieving the filename based on the id, the K-means label of the corresponding image is also returned.
In addition to the mappings, Redis is also used to store the value of the last external id attributed, allowing insertions to be done in several steps.

Process

The indexing process is a global way to refer to the management of the set of images available for search. It includes the insertion of images, the removal of images, and simply access to the saved content (vectors mostly). Note that for all three, the database and FAISS structure are considered instantiated and running.

Addition

1. The dataset is prepared and the images are separated in batches of 128 to avoid saturating the memory[12]. The following operations are done per batch.
2. The images are fed to the feature extraction model and their features vectors are retrieved.
3. The next available 'external' id is retrieved from Redis using the key *last_id*.
4. The vectors are added to FAISS along with a range of ids: [last_id ; last_id + #images]
5. One by one, the filenames and ids mappings are added to Redis.
6. Value associated to the key *last_id* is set to last_id + #images + 1 in Redis. The next batch of images is processed.

Removal

1. The id corresponding to the given filename is retrieved from Redis.
2. It is given to FAISS that then removes the corresponding entry.
3. The entries in Redis are removed using the retrieved index and the filename.

Vector access

For one single vector:

1. Retrieve the key corresponding to the given filename from Redis.
2. Retrieve the vector from FAISS using INDEX.RECONSTRUCT().

[12] if more than 128 images, otherwise they are processed together

For all vectors:

1. Retrieve all keys from Redis using KEYS("*").
2. Iterate on each key and discard the entries whose keys are filenames or the last id used (i.e. keys with an "\" or an "_").
3. Reconstruct the vector from FAISS using INDEX.RECONSTRUCT and add it to the list of vectors.

5.3 Search and retrieval

Settings

The search is entirely handled by FAISS as a variant of k-nearest neighbors, depending on the index used for the vectors and the distance measure chosen, as explained in Chapter 2. For the quantitative evaluation, the 5 nearest results are retrieved while for the qualitative evaluation, the 10 best results are displayed.

The indexes used were presented in Chapter 3, IndexIVFFlat and IndexFlat. The distance measure used is the Euclidean or L2 distance. It is the default one offered by FAISS and seemed to offer good results in the previous master's thesis. It was thus kept in this work.

Processes

The search is internal to FAISS and does not need any more actions than calling the appropriate function. However, to use IndexIVFFlat, it first requires to be trained using K-means clustering and needs several 'outside' steps. Similarly, the retrieval process is done in several steps, described after.

Training of the index

1. The vectors are retrieved from the current index structure following the *Vector access* process of section 5.2.
2. The index is instantiated and its parameters set, with the number of clusters set to the square root of the number of vectors retrieved.
3. The index is trained on the vectors using the TRAIN() method of the index.
4. The vectors are indexed to the newly trained index by batches of 128.

Retrieval

1. The query is subjected to the data transformations discussed earlier and then put through the features extraction model to retrieve its features vector.
2. Using the SEARCH method of the index, the 5 / 10 ids of the closest vectors are returned, along with their distance to the query vector.
3. Using the ids, the filenames (and labels in the case of the use of the K-means model) are retrieved using the Redis database (retrieve the values whose key is in the ids).

5.4 Training and Testing Protocols

Several techniques have been used when training the model to investigate two important elements in deep learning and their impact, pretraining/transfer learning, and generalization. Similarly, several divisions of the query dataset have been implemented concerning the evaluation of the frameworks.

Training on other datasets

It has been shown in different papers ([Oztel et al., 2019]) that taking a network already trained on another image dataset leads to better results than using a network trained from scratch. Regardless of the type of images on which the model is trained compared to the ones on which it will be used, pre-training improves the results by furnishing a model that has already learned some specificities of the processing of images. Especially in CBIR and representation learning, it is quite common to take a network trained on classification tasks and use it as a feature extractor ([Caron et al., 2021], [Grill et al., 2020]).

When dealing with pre-trained networks, two types of training can be performed ([Louppe, 2022]). The first one is called *transfer learning* and consists in preventing the weights of the pre-trained network to be further updated when training the new model (pre-trained network and added output layer) on new data. We say that the weights of the network are frozen. The second one is called *fine-tuning* and unlike for *transfer learning*, the weights of the pre-trained network are updated through backpropagation. They are not frozen. A representation of it can be seen in Fig 5.5. A comparison of training from scratch, transfer learning, and fine-tuning

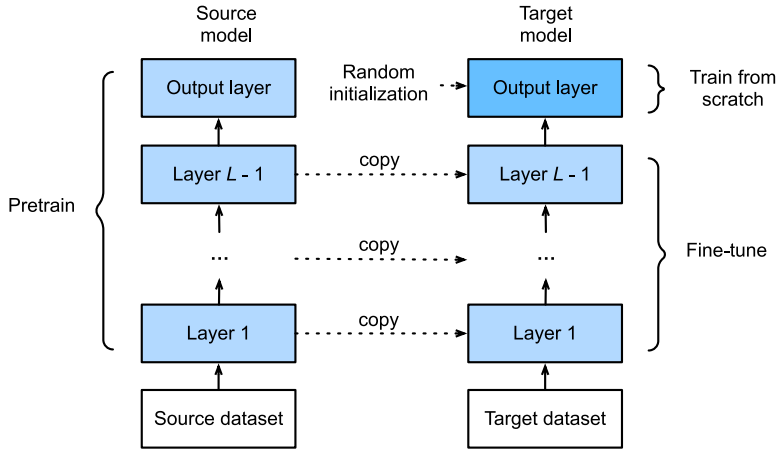

Figure 5.5: Fine-tuning: a network is trained on a source dataset. Its layers except the output layers are taken to form a new model that is then trained on the dataset of interest. Those layers have their weights updated, i.e. fine-tuned while the new output layer is trained from scratch. Credits: Dive into Deep Learning, 06/05/2023

(in particular the last two) was made in *Deep Transfer Learning for Art Classification Problem* [Sabatelli et al., 2019] as can be seen in Figure 5.6.

All three types of learning have been tested as part of this thesis and are available for most of the models, with the pretraining being made on ImageNet as stated in Chapter 4. There are however a few exceptions. Implementations 2 and 3 of the autoencoders are only trained from scratch as they were not made available pre-trained. When KimiaNet is used as backbone

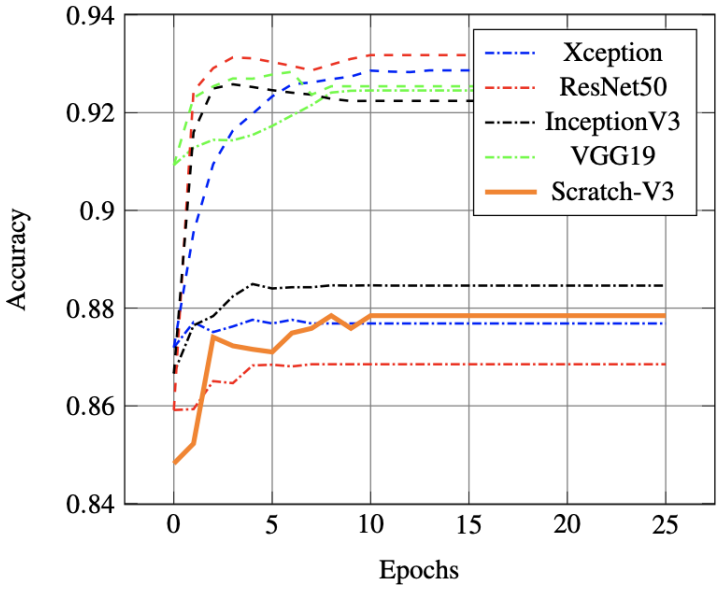

Figure 5.6: ‘ Comparison between the fine tuning approach versus the off the shelf one when classifying the material of the heritage objects of the Rijksmuseum dataset. We observe how the first approach (as reported by the the dashed lines) leads to significant improvements when compared to the latter one (reported by the dash-dotted lines) for three out of four neural architectures. Furthermore, we can also observe how training a DCNN from scratch leads to worse results when compared to fine-tuned architectures which have been pre-trained on ImageNet (solid orange line).’ Credits: [Sabatelli et al., 2019] (Deep Transfer Learning for Art Classification Problem)

architecture, it is by definition pre-trained on histopathological images rather than on ImageNet (it is also not available from scratch as it would be a basic DenseNet).

The pretraining is not done in the implementation but rather the weights of the models are retrieved directly from the same libraries as the models were obtained from, except for KimiaNet whose weights were downloaded from here. Fine-tuning is used by default but training from scratch is possible by not downloading the models’ weights and transfer learning by setting the parameter ‘freeze’ to True.

Split of the histopathology dataset

It has proven necessary during training and during testing to split the sets of the dataset into subgroups. This section explains the different splits used when training and when evaluating the framework. Note that those splits are the only way the training set was modified. No modifications were made to resolve the class imbalance during the training of the framework. This is due to several reasons: the type of data considered that would make it hard to generate new data that would be with certainty representative for a given class, the ‘gravity’ of the imbalance that would have led to a dataset either extremely reduced or extremely artificial, and finally the focus of this master thesis on *unsupervised* techniques that makes it in some way irrelevant to fix the labels when those are not supposed to be used.

Generalisation

As the end data on which the models are used in real applications can be data that have not been seen by the model during training, it needs to be able to process it. To test the way our model generalizes, two splits of the dataset have been devised. Both of them consist in using only half the classes of the dataset for training the feature extraction models, with the other classes being used for the indexing and the retrieval (Note that the separation in training - indexing - retrieval sets is still holding). While it leads to the non-use of some images, it allows comparison with the models trained on all the classes.

- The first method, the simplest, keeps half of the classes for training, randomly selected. This method was retrieved from the previous master's thesis. The classes used can be found in Table 5.1, for a total of 458 861 images, hence 72.4 % of the images of the training set. This allows us not to lose too many training samples while keeping an appropriate number of images for testing. However, some classes of the same project are separated as well as most classes of similar projects.
- The second method takes around half of the training images and a little bit more than half of the classes, with the first 10 classes kept (all classes from *camelyon16*, *Cells_ no_ aug*, *Glomeruli_ no_ aug* and *Iciar18_ micro*) as well as the classes number 21 to 36 (all classes from *Lbpstroma*, *Mitos2014*, *Patterns_ no_ aug*, *Tupac_ mitosis*, *ulb_ anapath_ lba* and *ulg_ bonemarrow*. The difference compare to the previous method is that no class used for training is in the same project as a class not selected, to avoid some interference. However, classes from similar projects could end up separated. This separation was chosen such that only half of the images would be kept and especially such that the two most populated classes would be separated.

janowczyk2_0	ulg_lbtd_lba_0	umcm_colorectal_0	patterns_no_aug_0
janowczyk2_1	ulg_lbtd_lba_1	umcm_colorectal_1	patterns_no_aug_1
lbpstroma_0	ulg_lbtd_lba_2	umcm_colorectal_2	tupac_mitosis_0
lbpstroma_1	ulg_lbtd_lba_3	umcm_colorectal_3	tupac_mitosis_1
mitos2014_0	ulg_lbtd_lba_4	umcm_colorectal_4	iciar18_micro_0
mitos2014_1	ulg_lbtd_lba_5	umcm_colorectal_5	iciar18_micro_1
mitos2014_2	ulg_lbtd_lba_6	umcm_colorectal_6	iciar18_micro_2
camelyon16_0	ulg_lbtd_lba_7	umcm_colorectal_7	iciar18_micro_3
camelyon16_1	warwick_crc_0		

Table 5.1: Classes used for the first method of generalization

Potential bias in results

As discussed in Chapter 4, there is a severe imbalance in the dataset between the different classes. This imbalance can impact the results as more important classes, being seen more, could have better representations, in turn leading to better results on average as they would be 'hiding' the other classes' results by taking up a more important space in the average. For this reason, different splits of the indexing and query dataset have been picked (in addition to the 2 previous ones).

- Default: All images of all classes. This setting leads to the most summarized but complete results, by taking into account all the images and summarizing them. However, it can be impacted by class imbalance and does not inform on the results of the individual classes.

- Weighted: All images of all classes but the measure are multiplied by a weight associated with the class, set to $\frac{1}{\#\text{ elements in the class}}$. It leads to summarized results using the entire dataset but 'artificially' modified to reduce the impact of class imbalance.
- Remove: All images for all the classes except the classes *Camelyon16_0* and *Janowczyk6_0*. This setting furnishes summarized results, less subject to overfitting as the two major classes have been removed. However, it is not as complete as the previous as about 50% of the images of the two sets are included. Furthermore, it completely disregards two classes of the dataset without taking them into account at all in the results which is also biased in some way.
- Random: 1020 images are drawn at random from the dataset, with an almost equal number per class. One image per class is retrieved till at least 1000 images are reached (the cycle is still completed before ending the retrieval of images) or no classes have images left in them. All classes do not have an equal number of images. Classes that do not contain enough images are removed from the cycle once all their images have been taken. This setting still leads to condensed results and is a partial solution to the class imbalance. It is also much faster to obtain. However, it is not reproducible due to the randomness of the image selection and only a small portion of the dataset is used. To reduce the impact of randomness, when using this protocol, 50 repetitions are made and the means and stds of the results are given.
- Per class: All images of the class given as argument. This allows to have specific results per class but it is not condensed (67 times each evaluation metric). A method exists to retrieve all the results of each of the 67 classes in an Excel file.

None of the protocols is perfect, as is summarised in Table 5.2. In the following, the results will mostly be given using the default and weighted protocols as they are the most representative. The *random protocol* will be used instead of the *default* and *all* for the AE models whose number of features is greater than 128 as the number of features makes the search process too slow to use the most complete protocols. For the two best models, detailed results per class will be available in the appendix as well as results using the *remove* protocol.

Protocols	Default	Weighted	Remove	Random	Per class
# Images	96 134	96 134	47 578 (49,5%)	1020 (1%)	96 134
# of results	1x	1x	1x	1x	67x
Impact of imbalance	High	Small but artificial	Reduced	Small	None

Table 5.2: Summary of characteristics of the protocols

5.5 Evaluation protocols

To evaluate the quality of a model, different measures are computed to retrieve quantitative results. In parallel to those quantitative results, qualitative ones are also obtained through the use of various figures and graphs.

Regarding the data onto which those measures are computed, the indexing set is indexed in the database while the queries are obtained from the query set, following the protocols explained just before. The training set is only used to train the feature extraction models. The only exception is for the time measures, with a few experiments being made with the indexing set and training set indexed into the database to see the impact of the number of vectors indexed on the search and retrieval time.

Qualitative

Qualitative evaluations are used for two different purposes. The first purpose is to evaluate the quality of the representation made by the feature extraction models.

To evaluate it, the first technique used is called t-SNE. It was introduced in 2008 by L. Van der Maaten and G. Hinton in [van der Maaten and Hinton, 2008]. It is a technique that decreases the dimensionality of a given vector such that it can be displayed in 2 to 3 dimensions. It is also a kind of feature extractor that works in an unsupervised fashion and conserves the relative distances between the data it is given (vectors close to one another originally are expected to still be close in the new 2D/3D space). Unlike PCA, it is a non-linear technique. It uses Stochastic neighbor embedding (SNE) as the backbone but has changed some elements to improve the visualization/results. SNE consists in using probabilities to represent the distance or similarity between data points, with one probability $p_{j|i}$ for the original distance and one for the distance in the new dimension $q_{j|i}$. Those probabilities are computed as follows:

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2}$$

$$q_{j|i} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq i} exp(-||y_i - y_k||^2}$$

where the x are the original data points and the y are the data points in the new dimension.

The reduction is considered optimal if the 2 probabilities are equal, making the reduction of the difference between the two the optimization goals of (t-)SNE. This difference is computed using the Kullback-Leibler divergence and is minimizing gradient descent (over y_i).

It leads to the kind of figures displayed in 5.7.

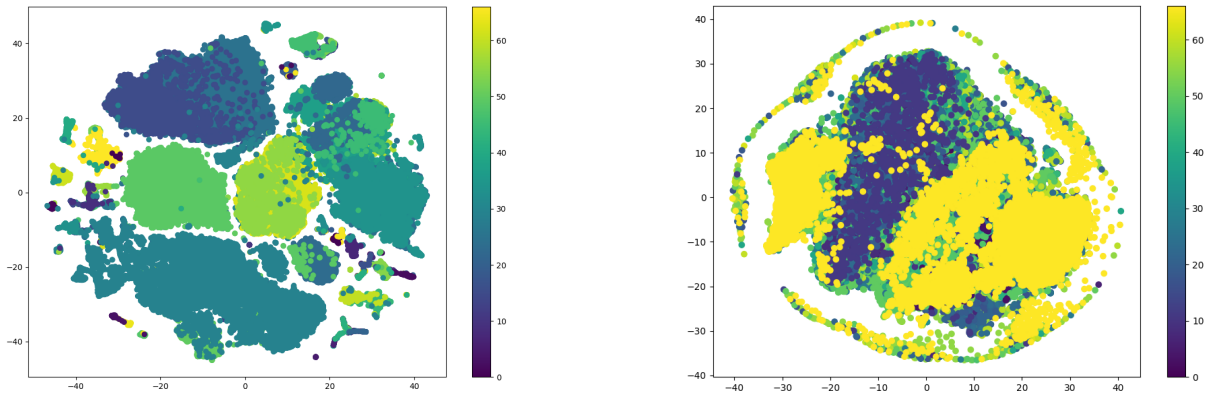

Figure 5.7: t-SNE on embeddings obtained using two different feature extraction models (DeiT on the left, the first implementation of the AE on the right). The colors represent the class to which the point belongs. See how the left model leads to a better distinction between classes than the right model.

Still for the evaluation of the representation, the second technique used is specific to the autoencoder models as it consists in comparing visually the reconstructed image with the original image. Such visualization allows to know how well the model has learned the important features of the image (if it cannot rebuild it, chances are that the representation is not complete enough). However, a good reconstructed image is not necessarily promise of great results

because a too-good reconstructed image may indicate overfitting rather than the existence of a good representation. An example of such a comparison is displayed in Fig 5.8.

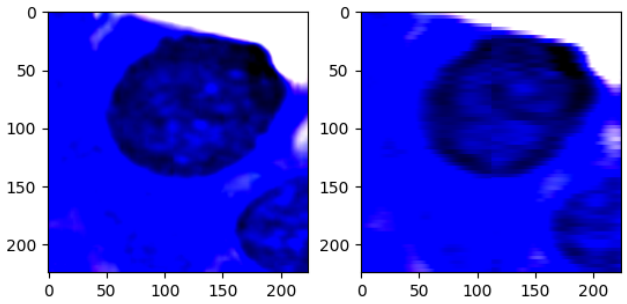

Figure 5.8: Original image and its reconstruction by one of the autoencoder implementations.

Finally, regarding the qualitative evaluation of the whole framework, it consists in retrieving and displaying the 10 best results retrieved by the framework. Those results are displayed from best (closest) to worst (furthest), right after the query and with their distance to it given too. All those retrieved images are saved in a selected directory and renamed such that the class to which they belong appears in the name. This qualitative analysis is important because the quantitative measures, using labels, can not grasp the issue of intra-class diversity while here, it can be seen if the images retrieved look similar to the query.[13] An example of such results can be seen in Figure 5.9.

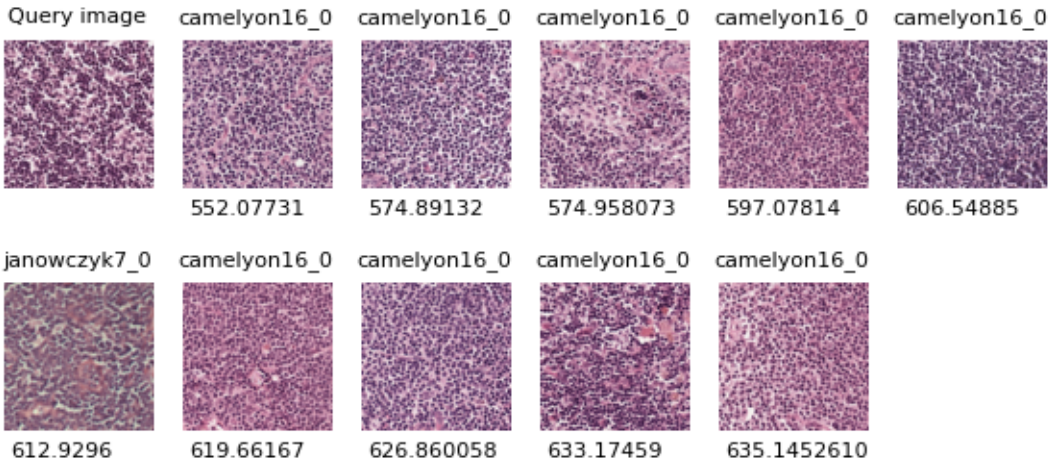

Figure 5.9: Query (camelyon16_0) and the 10 retrieved images

Quantitative

Retrieval results

When evaluating an image retrieval system, several measures have been used in papers such as (mean average) precision ([Caron et al., 2021]), recall ([Zheng et al., 2021]), F1 score

[13]Note that for this work, due to the time limitations, the qualitative analysis is not judged by experimented people.

([Riasatian et al., 2021]), top-k (top-1 in [Kalra et al., 2020], top-5 in [Hegde et al., 2019]), majority vote ([Chen et al., 2022]), ... with the most popular being the Mean Average Precision (MAP) and top-k accuracy. For this work, this last one was selected as the main evaluation measure, with $k = 1$, $k = 5$ as well as the majority vote in the 5 first results.

The top-k accuracy consists basically in checking whether at least one image in the k first results belongs to the same class as its corresponding query (summed and averaged over all queries). Mathematically:

$$top_k = \frac{\sum_{x_i \in \mathcal{X}} \mathcal{I}(\sum_{j=1}^{k} \mathcal{I}(y_i = y_i^j) > 0)}{|\mathcal{X}|}$$

where x_i is the ith query, y_i the associated 'label'[14], y_i^j the 'label' of the jth best result retrieved for the query x_i and $\mathcal{I}(\text{cond}) = 1$ if cond is True, 0 otherwise. In the case of $k = 1$, then it simplifies to the number of times the best result has the same class as the query. Mathematically:

$$top_1 = \frac{\sum_{x_i \in \mathcal{X}} \mathcal{I}(y_i = y_i^1)}{|\mathcal{X}|}$$

The Majority vote is quite similar to top-k accuracy but instead of being satisfied with only one result being of the correct class, the Majority vote requests that at least half of the results be of the correct class. The results must be good on average. Mathematically:

$$maj_k = \frac{\sum_{x_i \in \mathcal{X}} \mathcal{I}(\sum_{j=1}^{k} \mathcal{I}(y_i = y_i^j) > \lfloor \frac{k}{2} \rfloor)}{|\mathcal{X}|}$$

These three measures are applied on three different levels, defined as follows:

- Class level: the label y in the previous formulas represents the class of the image. It is the strictest measure, that focuses on the validity of the retrieval for one single class at a time.
- Project level: the label y represents this time the project of an image. This means that if the query and the retrieved image do not belong to the same class, as long as their two classes belong to the same project (eg, *camelyon16_1* and *camelyon16_0*), then it will be valid per this measure. This measure is less strict than the previous one. It is a good indication of whether or not the framework can at least retrieve an image of the same group as the image of interest. Even if it is not totally correct, it is at least not totally out of touch. Furthermore, it also indicates how many of the incorrect results per the first measure are related to a project classification (correct project, wrong class) which may be of great importance when, for example, the two classes of a project are 'healthy' vs 'sick'.
- Similar projects level: the label y represents the project of an image as for the previous measure but this time, the $=$ in the condition is not a 'strict' equality but signifies that the projects are similar (see Chapter 4). It will be thus valid if the query and the retrieved image belong to projects that are considered similar (Table 7.1, projects separated by a single line). It is the loosest measure, used to see if the framework captures information about the 'provenance' of an image, if it realizes that similar projects are connected.

In total, 9 measures will be used as shown in Table 5.3.

As it could quickly crowd the results, especially given the number of protocols available, most of the models will only use the first two measures in Table 5.3 or those two with the $4th$ and $5th$ too instead of all. The full results will be reported only for the best models. Also note that for the K-means, results with both the 'old' labels and 'new' labels will be available.

[14] See later what label refers to.

top1_class	top5_class	maj_class	top1_proj	top5_proj	maj_proj	top1_sim	top5_sim	maj_sim

Table 5.3: Quantitative measures

Computing times

On top of the measures for the quality of the retrieval, three measures for the computational efficiency of the framework are used, based on the time certain actions take.

The first one concerns only the feature extraction model and its training. It does not concern itself with the efficiency of the search and retrieval. The time that a model takes to finish one epoch, t_e, is considered in the choice of the feature extraction element.

The second one is relative to the indexing element. It indicates the time it takes to index all given elements in the database (t_i). This time is split into 2: the time taken by the model to retrieve the image representation, t_i^m and the time taken by the indexing of the vectors in the different structures (Redis and Faiss), t_i^{db}.

The last time is the one relative to the search and retrieval. It is the time taken to retrieve the supposedly most similar images to the queries given, t_s. As the number of queries depends on the protocols used, it will be indicated per query. It is also split into two values, one for the time taken by the model, t_s^m, and one for the time of the search and retrieval by FAISS, t_s^{db}.

Note that the two times corresponding to the model comprise both the time to 'turn' the image into its feature vector as well as the time to transfer the vector to the CPU from the GPU. It is also considered as the time taken by the model as the size of the model directly impacts the time of that transfer.

5.6 Implementation

This section presents the material used for the testing of the frameworks as well as the different libraries it used/ was inspired from.

5.6.1 Material

The set-up onto which the different tests have been run is composed of

- 2 NVIDIA GeForce RTX 4090 GPUs, each with 23.988Gb of memory.
- One Intel CPU. The CPU model is Intel(R) Core(TM) i9-13900KF (13th generation) running at a clock speed of 5.5GHz. It has 24 physical cores, supporting hyper-threading, making it 32 logical processors. The CPU cache size is 36.8MB.
- 64GB of RAM.

Except when the training is paralleled, the codes were all run on the first GPU. For time measures, no other applications were running during the tests to ensure the reliability of the results.

5.6.2 Libraries & external codes

This implementation was made of original code as well as codes/functions from several libraries and other sources. The different elements used are cited in the following.

One of the two most important components is Pytorch. This implementation is PyTorch-based and solely uses torch elements to build and train the models (layers, loss, models,..). The second most important element is the implementation corresponding to [Defraire, 2021]. It was used as the base for this implementation, with the new elements being added to the existing structure (Implementation).

For the supervised models, pretrained architectures were retrieved from Torchvision models (DenseNet121, ResNet500, VGG19, ViT), HuggingFace (DeiT, CvT), Kimia Lab (KimiaNet) and EfficientNet-PyTorch (EffNet).

For the self-supervised models, the first AE implementation is from a Github implementation of an ImageNet autoencoder, the second implementation is from Pytorch while the third is from an article about Contractive AE on a data-science website. The K-means model is from sklearn and Byol is from Byol-pytorch.

For the dataset preparation, the data augmentations are composed using Torchivision transforms. For AMDIM and SIMCLR data augmentations, the lists were obtained from Lightning-Bolt github (AMDIM) and documentation (SimCLR). For the sampling method in the Dr and CL concepts, the DR sampling was obtained from Tejaswi's Github page and the samplings for (A)CL were built upon it. The informative sampling is obtained from the source code of [Roth et al., 2020].

For the losses, the margin loss and softmax loss come from Revisiting Deep Metric Learning Pytorch. and Deep Metric Learning Baselines, source code of [Roth et al., 2020] and other DML papers. The ProxyNCA++ loss comes from the source code of [Teh et al., 2020]. The contrastive loss and BCE loss are based on Siamese Network using Pytorch by Pritam Chanda. The soft triple loss is obtained from the source code of [Qian et al., 2019]. Finally, the triplet margin loss and the CosineEmbedding loss come from torch while the NT-Xent loss comes from pytorch metric learning.

The manipulations of the FAISS indexes are made through the faiss library while the manipulation of the Redis database is made possible by redis-py.

Chapter 6

Results and Discussion

This chapter presents the results obtained through the empirical experiments described in the previous chapter. It first discusses the results obtained using supervised models, presenting each backbone architecture, concept, and each loss one by one (Section 6.1). It then talks about the unsupervised methods, making a distinction between the three methods, and then each of the experiments of each method (Section 6.2). After those two sections, their results are compared and discussed (Section 6.3). The next four sections (6.4, 6.5, 6.6, and 6.7)present the remaining parameters/elements that may affect the results, to finally end with a general conclusion on all the elements and their impact (Section 6.8).

All time results are retrieved in the exact same conditions. Furthermore, unless specified otherwise, the FAISS index was not trained.

6.1 Results per Supervised model

This section presents the results of the supervised models. It first starts with the architectures, split between CNNs and transformers. For each model, the results are obtained using the *random (Rand.)*[1], *default (Def.)* and *weighted (Weig.)* protocols (Section 5.4). Then the results using the different concepts and varying their parameters are introduced. It ends with a general discussion of the impact of the different concepts, backbones, and parameters.

The models are described in Table 6.1. The first 7 test the different CNN and Transformer architectures, then the next 3 the losses of DML[2]. The 11th tests DR and the last models test the loss of CL and NCL. Numbers in the first column will be used in other tables to refer to these models.

Note that all architectures are taken pre-trained and without the weights frozen. They lead to vectors of 128 elements and are trained on the entire training set. The scheduler is Exponential each time. The number of epochs and batch size vary because all models were not launched at the same time and those parameters were thus changed frequently during this work. As they do not impact the results too much, it was kept as presented. For the time taken by an epoch though, it has been obtained in the same conditions (Section 5.6.1). Note that for kimiaNet, it is trained using parallelism because the official KimiaNet implementation uses parallelism.

CNNs

Before going to the comparison of the architectures, a quick note on the metrics and protocols themselves. First, it can be seen from Table 6.2 that for all models, the weighted and random

[1] To show the range obtained with the protocol.

[2] Resnet50 has been chosen as default for the architecture based on the results in [Defraire, 2021]

Nb	Architecture	Concept + Loss	# epochs	Batch size
1	Resnet50	DML + Margin	15	32
2	Densenet121	DML + Margin	20	32
3	EfficientNet_B0	DML + Margin	20	32
4	KimiaNet	DML + Margin	50	32 + parallelism
5	ViT	DML + Margin	20	32
6	DeiT	DML + Margin	15	32
7	CvT	DML + Margin	20	32
8	Resnet50	DML + Proxy	15	32
9	Resnet50	DML + Normalised softmax	15	32
10	Resnet50	DML + SoftTriple	15	32
11	Resnet50	DR + triplet	15	32
12	Resnet50	CL + contrastive	15	128
13	Resnet50	CL + Cosine	15	128
14	Resnet50	CL + BCE	15	128
15	Resnet50	CL + NT-Xent	15	64
16	Resnet50	NCL + contrastive	15	128
17	Resnet50	NCL + NT-Xent	15	64

Table 6.1: Models tested in supervised learning

Nb	Prot.	top1 %	top5 %	top1_proj %	top5_proj %	t_s^m s
1	Rand.	72.06 ± 0.9	87.7 ± 0.9	95.35 ± 0.4	97.92 ± 0.28	2.85 ± 0.077
2	Rand.	74.26 ± 1.02	88.7 ± 0.87	95.44 ± 0.35	98.2 ± 0.3	5.61 ± 0.07
3	Rand.	73.12 ± 1.1	88 ± 0.6	95.85 ± 0.24	97.4 ± 0.31	2.71 ± 0.074
4	Rand.	54.98 ± 1.06	80.94 ± 0.78	91.46 ± 0.66	96.23 ± 0.5	11.4 ± 0.079
1	Def.	75.89	92.93	94.83	98	260.72
2	Def.	79.27	93.96	95	98.46	537.473
3	Def.	77.78	92.33	94.94	97	253.16
4	Def.	72.6	92.16	93.18	97.15	1061.23
1	Weig.	72.46	86.56	95.42	97.93	260.1
2	Weig.	76.36	88.53	96.5	98.6	536.7
3	Weig.	74.76	87.84	96.4	97.78	248.88
4	Weig.	56.92	79.36	92.64	96.44	1071.22

Table 6.2: Accuracy Results for the models using a CNN as backbone + total time of the model for the search

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
1	20	51.4	2.72
2	25	63.07	5.6

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
3	18	38.45	2.64
4	17	42.22	11.05

Table 6.3: Time results for the models using a CNN as backbone

protocols offer very similar ranges of accuracy (especially visible with the top-1 accuracy of the 4th model). From this, it can be concluded that the random protocol leads to a good approximation of the 'true' accuracy even when using only 1020 images instead of the whole set. It validates the decision of using the *random protocol* when a high number of features make the other two protocols undoable in an appropriate time. Similarly, as expected, the *default* protocol offers higher results than the other two, leading to believe that it is indeed impacted by the class imbalance of the dataset which is taken into account in the two other protocols. Second,

it might be surprising at first sight that the indexing time t_i^m is much smaller than what could be expected seeing the search time per query t_s^m/query. Indeed, as the indexing set is made of around 100 000 images, the indexing time per query becomes less than 1 ms in all cases, with the search time per query at least 2-3ms. This is due to the indexing being made by batches of 128 images whereas the queries are given to the model one by one (The times' signification can be found in Section 5.5).

Let us now turn to the discussion on the performance of the four CNN-based frameworks using the accuracy results of the same Table 6.2. As can be seen, the four models have very close results for the three protocols for the accuracy made at the project level (columns 5 and 6), with around 94-95% top-1 accuracy and 97-98% top-5. They also have similar results at the class level for the *default* protocols but when it comes to the *random* and *weighted* protocols, it can be seen that KimiaNet is lagging far behind. KimiaNet is also slightly behind in the three other metrics but it is particularly visible with top-1 accuracy on the class scale. This seems to indicate that, on top of globally being less good than the other models, KimiaNet is also more impacted by class imbalance and tends to classify better the dominant class than the minority ones[3]. Furthermore, as seen in the last column of 6.2 and in Table 6.3, KimiaNet is much less efficient than its counterparts, taking more than twice the time of the Densenet per query for the search (While it is on par for the training time and indexing time, it must be taken into account that it is the only model trained in parallel, which explains those differences. Had it been trained without parallelism as the other models, it would not have been the case[4]).

The three other models offer very similar results, with Densenet (2) being slightly better, but never by more than 2%. However, as shown in Table 6.3, Densenet is much slower than the other two models, in any part of the framework (training, indexing, search). Especially, it is twice slower when comparing the time per query, which for the end purpose, is the most important. While it stays really small (not even a second), only around 100 000 images were indexed in the database. For a higher number of images, such a difference would be much more significant.

Hence, the two best architectures seem to be the Resnet and the Efficientnet models. They have similar timings in all three parts of the framework and very similar results, alternating between which is better than the other.

Transformers

Nb	Prot.	top1	top5	top1_proj	top5_proj	t_s^m (s)
5	Rand.	58.29 ± 1.1	82.7 ± 0.75	91.8 ± 0.59	96.25 ± 0.45	2.65 ± 0.077
6	Rand.	61.65 ± 1.07	83.57 ± 0.82	91.73 ± 0.6	96.18 ± 0.39	3.06 ± 0,092
7	Rand.	55.29 ± 0.76	80.2 ± 0.87	90.77 ± 0.47	95.3 ± 0.42	5.5 ± 0.076
5	Weig.	59.28	81.44	92.98	96.5	248.98
6	Weig.	59.95	82.64	91.1	96.46	287.87
7	Weig.	57.04	79.37	91.12	95.5	533.44
5	Def.	72.83	91.19	92.67	96.66	249.42
6	Def.	73.69	90.82	92.13	96.66	289.55
7	Def.	70.7	91.47	93.37	96.27	529.45

Table 6.4: Results for models using a Transformer as backbone

Using Table 6.4, the same remarks as those made in the CNN sections (6.1) can be made again on the protocols. The *weighted* and *random* protocols have the same range of accuracies

[3]This will be discussed and shown in more depth in Section 6.6

[4]The parallelism was kept anyway to match as much as possible to the original KimiaNet

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
5	43	123.27	2.6
6	43	141	3.02

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
7	40	94.56	5.51

Table 6.5: Time results for the models using a Transformer as backbone

while the *default* protocol shows much better results. This means that the behavior was not only observed with CNNs but also with Transformers which ratifies the validity of the random protocol. Similarly, the indexing time is still smaller than what could be expected based on the query time, though the difference is less obvious.

The results in accuracies of the frameworks based on a Transformer architecture can be seen in the same Table 6.4. As can be observed, none of the three architectures is much better than another. All three lead to results in the same accuracy range, regardless of the protocol or the metric. Except for the top-5 accuracy at the class level, with the *default* protocol, where CvT (7) is slightly higher (but by a negligible amount), the two other models always provide results a little bit better, though again never more than 1-2% better. At first sight, DeiT (6) seems better at the class level, tending to discriminate better between classes of the same project whereas ViT (5) takes the first place on the project scale. Nevertheless, both of them are pretty much interchangeable.

In terms of computing times (see Table 6.5), ViT and DeiT exhibit again a similar behavior. However, while CvT is a bit better in terms of training time and indexing time, its performance crashes when looking at the query time, with a value almost twice higher as the ones of the others. It might indicate that CvT is optimized to process the data in batches rather than one by one.

Overall, both DeiT and ViT could be chosen as Transformer architectures, with ViT being slightly quicker and thus preferable.

Deep Metric Learning with different losses (DML)

Nb	Prot.	top1	top5	top1_proj	top5_proj	t_s^m (s)
1	Weig.	72.46	86.56	95.42	97.93	260.1
8	Weig.	57.54	80.32	91.4	95.11	259.91
9	Weig.	76.78	88.36	95.05	97.31	260.93
10	Weig.	74.79	86.75	94.58	96.63	262.13
1	Def.	75.89	92.93	94.83	98	260.72
8	Def.	72.99	91.13	92.87	96.22	260.17
9	Def.	82.68	94.07	95.28	97.58	268.63
10	Def.	81.83	93.77	95.34	97.56	261.39

Table 6.6: Results for models using the different DML losses

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
1	20	51.4	2.72
8	16	49.55	2.74

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
9	16	49.19	2.8
10	17	49.53	2.71

Table 6.7: Time results for the models using the DML learning and its different losses

The accuracy results are presented in Table 6.6. The first thing to notice is how the model using the proxy loss (8) seems to be the most impacted by class imbalance, with a drop of 15% and 11% in top-1 and top-5 accuracy at the class level between the *Default* and *Weighted* protocols.

When giving equal importance to all the classes, it gets far worse results than when attributing each image the same importance and hence more weight to dominant classes. Interestingly, this difference almost disappears when the level is set to project rather than class, suggesting that while the proxy loss fails at correctly recognizing between classes of small projects, it does well when having to recognize only the project. For the three other losses, the impact of the imbalance is barely noticeable in the Margin loss (1) while it is present but reduced (around 6-7 %) in the softmax and softTriple losses (9 and 10). The resilience of the margin loss to the class imbalance is probably linked to the fact that, unlike the three other losses, it is not a proxy-based loss but a ranking one. It thus relies less on the labels than the others. The proxy loss being more impacted than the two other losses does not have a directly visible cause. A possibility might be that the distance on which the proxy loss is based, the Euclidean distance, is more sensitive to imbalance than the cosine distance used in the others. s

Regarding the range of the results, the proxy loss, on top of being more affected by class imbalance, also leads to worse results than the other losses, always being below the others, for all metrics and all protocols. The three remaining losses have quite similar results for the *weighted* protocol, with only the softmax loss being very slightly above at the class level and the Margin loss above at the project level. For the *default* protocol, the results are similar in the project scale, with no loss being always the best and less than 1% difference between the scores. In the class scale, the softmax always comes on top, with a significant difference in score with the margin in top-1 accuracy (7%). The softTriple loss is right below the softmax, taking second place at the class level.

Regarding the times in Table 6.7, there is basically no difference between the four losses in the indexing and search time. It is to be expected as the loss is only used during training and has no use during the other phases. Hence, during training some differences can be seen that directly relate to the nature of the loss. As expected, the margin loss takes longer to train than the other losses due to it being a ranking loss rather than a proxy-based loss. The margin loss requires the formation of pairs in the batch of data, explaining the few additional minutes it takes per epoch.

Overall, the Normalized softmax loss seems to be the most accurate, although the margin has the advantage of being barely impacted by the class imbalance, making it a preferred choice in such a context where the dataset is highly imbalanced.

Deep Ranking

Nb	Prot.	top1	top5	top1_proj	top5_proj	t_s^m
11	Weig.	25.8	53.29	63.64	82.08	270.96
11	Def.	65.36	86.06	85.39	94.18	271.75

Table 6.8: Results for the model using deep ranking

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
11	43	49.77	2.83

Table 6.9: Time results for the model using deep ranking

The results are presented in Tables 6.8 and 6.9. A very big drop in accuracy can be seen between the results of the *default* and *weighted* protocols, especially for the top-1 accuracy at the class level with a drop of near 40%. A highly probable hypothesis to explain such a drop is the way the batch is formed. It is not made informatively unlike in the previous supervised

method, to fit the definition of the random sampling. Thus, the batches are highly imbalanced and this seems to lead to imbalanced results per class.

Contrastive Learning (CL) with different losses

Nb	Prot.	top1	top5	top1_proj	top5_proj	t_s^m (s)
12	Weig.	30.96	53.82	63.47	81.15	262.14
13	Weig.	37.83	72.06	92.26	97.71	261.07
14	Weig.	39.43	73.39	90.16	96.8	260.78
15	Weig.	59.1	81.33	93.82	96.9	261.14
12	Def.	61.21	84.92	80.65	93.44	262.47
13	Def.	64.5	91	91.21	96.96	260.15
14	Def.	60.78	90.47	89.94	96.69	261.15
15	Def.	83.14	94.11	95.23	97.81	261.55

Table 6.10: Results for models using the different CL losses

Nb	Prot.	top1	top5	top1_proj	top5_proj	t_s^m (s)
16	Weig.	5.92	15.57	12.6	35.88	260.96
17	Weig.	58.81	79.85	94.63	97.49	259.39
16	Def.	29.5	59.89	37.65	72.39	260.98
17	Def.	82.97	93.76	95.06	97.1	260.95

Table 6.11: Results of the models using only positive losses to investigate the impact of dimensional collapse, given the loss

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
12	33	49.5	2.73
13	32	49.56	2.71
14	31	49.57	2.72

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
15	51	49.48	2.72
16	33	49.52	2.72
17	30	49.65	2.72

Table 6.12: Time results for the models using the CL concept and its different losses

The results are presented in Table 6.10. The first thing to notice is that similarly to deep ranking, the contrastive methods are highly impacted by class imbalance, with a drop between 20 and 30% of top-1 accuracy at the class level, for each, between the *default* and *weighted* protocols. This issue with the class imbalance is probably related again to the absence of the informative sampling present in the other supervised methods.

Regarding the losses themselves, surprisingly, the contrastive loss (12) presents the worse results of all, with a significant difference to the second worse. The cosine (13) and BCE (14) both have similar results, with the cosine slightly better at the project level and the BCE better at the class level, regardless of the protocol. Surprisingly again, the NT-Xent loss (15) achieves much higher results than the other three losses, with a 20% of difference with the second best in top-1 class accuracy. That difference is less evident on top-5 class accuracy and top-1 at the project level, with around a 5% difference between the two best losses, and is completely absorbed on top-5 project accuracy. This is probably explained by the fact that contrarily to the other losses, where only half the samples of the batch form a negative pair, in the NT-Xent loss, every sample is used to form a negative pair with another sample of the batch unless it is a constructed positive pair. This brings much more contrast than in the other losses.

For the time results reported in Table 6.12, all models take the same time to train regardless of their loss, except for NT-Xent. This was expected due to the different sampling method of

NT-Xent that leads to a much higher number of pairs being taken into consideration in the computation. This takes more time to process and also reduces the maximum usable batch size as the used GPU memory is greater, hence the reduced batch size of models 15 and 17 of 64 instead of 128.

Finally, for the models using only positive pairs, the results are displayed in Table 6.11. There is not much surprise in the results, with the contrastive loss collapsing completely with a drop of accuracy between 25 and 45% compared to the 'correct' model. For NT-Xent, the results are barely affected due to the particularity of the loss of still forming negative pairs based on the other elements of the batch rather than simply relying on the furnished negative pairs as in the other three losses

Discussion on the supervised

The results per model have been discussed in the previous section, with a comparison of the different options of each concept/architecture style. This section compares the different types of architecture together and the different concepts.

For the architectures, the CNN-based models clearly surpass the Transformer based models, except for KimiaNet which achieves similar ranges of accuracies. On top of that, transformers take more time to train and index. Their query times vary between the different architectures of the two types but the quickest Transformers have similar query times as the quickest CNNs. Different elements could explain such differences in results such as a lack of training data (Transformers are known to need much more data than CNNs as they do not dispose of the inductive bias CNNs have that allow them to better/quicker understand image data) or an incompatibility of the training method chosen to that specific type of architecture (CNNs might be more adapted to deep metric learning than Transformers) or again a lack of training time. It might be interesting to note nevertheless that when looking at the results at the project level, Transformers and CNNs are mostly on par, with CNNs only having 1-2% more on average. The same happens for the results of the *default* protocols, with results for the two types of architectures being very close. This suggests that Transformers are more affected by class imbalance than CNNs and also have more difficulty at grasping details than CNNs. Qualitatively, all architectures lead to well-discriminating algorithms. T-SNE projections for the ResNet (1), EfficientNet (3), and ViT(5) architectures are presented in Figure 6.1. It can be seen that several well-separated clusters have

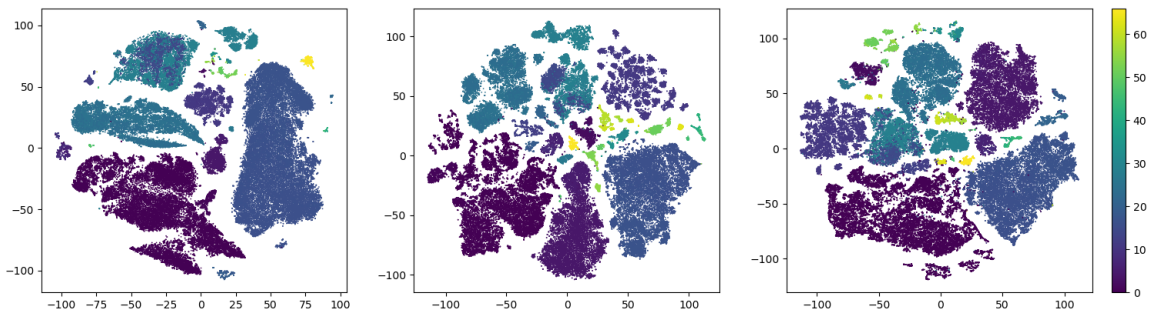

Figure 6.1: T-SNE for ResNet (1), EfficientNet (3), ViT (5) models

been formed, corresponding to different classes. The clusters of the ViT model seem a little bit more noisy than the two CNN models, which is coherent with the quantitative analysis. Indeed, several darker points can be noticed everywhere on the graph for ViT. It is especially noticeable

in the middle part of the graph, on the light blue and yellowish clusters. EfficientNet also has some 'floating' points, but far less numerous and less noticeable.s

For the concepts, DML is the best, by far if we exclude the NT-Xent loss of contrastive learning (15). The complexity of comparing the different protocols comes from the high variance of the results inside one single protocol. If we exclude the proxy loss in DML (8) and the NT-Xent loss in contrastive learning (15), then there is a high difference between the results of the *weighted* protocols of the three concepts, especially noticeable on the top-1 accuracy at the class level. That difference is also there on the *default* protocol though much less important (a 10-15% drop for the top-1 class accuracy for *default* against a 35-40% drop for *weighted*). The proxy loss and NT-Xent losses make the situation harder to 'classify' as the NT-Xent loss results in the *default* protocol are higher than the results of the DML losses, making CL win in that specific configuration. However, in the *weighted* protocol, NT-Xent loss is clearly below the DML loss at the class level, being impacted much more by the class imbalance. It is only above the proxy loss but this loss is the worst of DML and thus not representative of the concept.
The difference in results between the different protocols may be explained by the smaller dependency of CL and DR on the class labels as they only use them to form the pairs, and such that it would only concern themselves with the binary categorization of 'same class'/'different class'. Furthermore, as stated in Chapter 4, the diversity inside a single class can be quite important and sometimes, samples from another class might be closer to the anchor than a sample from the same class. As the pairs are formed by randomly selecting a sample from the same class and from a different class, it might have happened that the sample chosen for the positive pair be actually more different from the anchor that the sample selected for the negative pair, messing up the learning process. Qualitatively, the results are reflected by the quality of the embeddings. In Figure 6.2, the t-SNE maps of models 9 (DML - softmax loss), 11 (DR), and 15 (CL - NT-Xent loss) are displayed. Models 9 and 15 have well-separated clusters with those of model 15 being slightly more noisy while the clusters of model 11 are much less separated and most classes are superimposed (especially visible with the yellowish colors that are not as visible as in the other maps). In conclusion, the best supervised model seems to be model 9, hence a ResNet-based

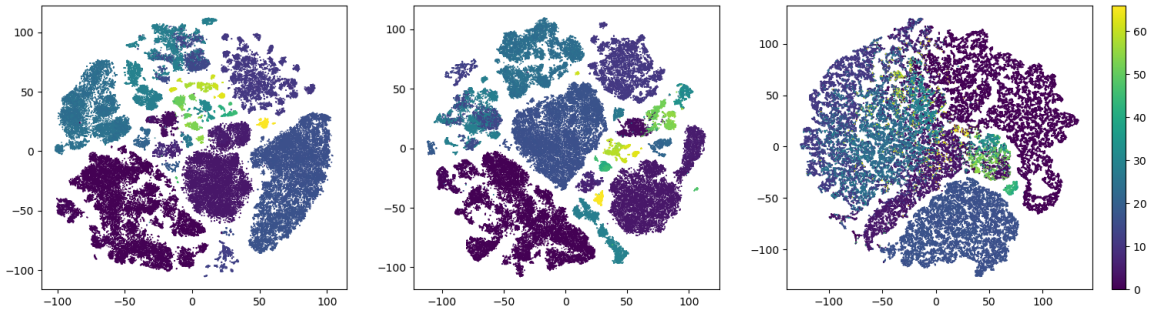

Figure 6.2: t-SNE for DML with Softmax loss (9), CL with NT-Xent loss (15), and DR with triplet loss (11)

feature extractor trained using the DML concept using a Normalized softmax loss. However, Model 1 (ResNet-based, trained using the DML concept with the margin loss) is more robust to class imbalance, with almost the same results as Model 9. Therefore, it is that model that is kept for the supervised learning part as reference.

6.2 Results per self-supervised model

The results are presented for the AE, separated per implementation then followed by a joint discussion. Then the K-means final results are provided and the analysis of the new labels is made. It finishes with the results of the A(N)CL and a joint discussion of the three self-supervised methods.

AutoEncoders

As previously stated, for efficiency purposes, the *random* protocol is the protocol by default used for the AEs with a high number of features due to prohibitive search times with the other protocols.

The tested models are presented in Table 6.13 while the description of the different experiments is made in Chapter 5, Section 5.1.2.

All those models are trained on the entire dataset, without being pre-trained, unless specified otherwise in the experiment description. The Scheduler is also always exponential except for the first implementation where the learning rate is modified based on the cosine function. Similarly to the supervised models, the varying number of epochs is due to both time constraints and different design choices given the time of launch. The number of features is directly dependent on the implementation and the experiment.

Auto-encoders

Nb	Model	# epochs	batch size	# features	other
18	Imp 1 - CNN based	50	256 + parallelism	128	Exp 1
19	Imp 1 - CNN based	5	256 + parallelism	25088	Exp 2 - vgg11
20	Imp 1 - CNN based	5	256 + parallelism	25 088	Exp 2 - vgg16
21	Imp 1 - CNN based	5	256 + parallelism	25 088	Exp 3a - resnet18
22	Imp 1 - CNN based	5	256 + parallelism	25 088	3b - resnet18
23	Imp 1 - CNN based	20	256 + parallelism	25 088	Exp 4 - pre-trained
24	Imp 1 - CNN based	5	256 + parallelism	25 088	Exp 5 - no jano
25	Imp 1 - CNN based	5	256 + parallelism	25 088	Exp 6 - class
26	Imp 1 - CNN based	5	256 + parallelism	25 088	Exp 6 - project
27	Imp 1 - CNN based	50	64 + parallelism	1000	Exp 7
28	Imp 2 - VAE	15	32 + parallelism	3840	Exp 1
29	Imp 2 - VAE	15	256 + parallelism	20	Exp 2a
30	Imp 2 - VAE	15	256 + parallelism	20	Exp 2b - one more layer
31	Imp 2 - VAE	15	256 + parallelism	128	Exp 2c - smaller reduction
32	Imp 2 - VAE	15	256 + parallelism	128	Exp 3
33	Imp 3 - Contrastive AE	15	128 + parallelism	3072	Exp 0 (fixed imp.s)
34	Imp 3 - Contrastive AE	15 et 100	128 et 256 + parallelism	3072	Exp 1
35	Imp 3 - Contrastive AE	100	256 + parallelism	3072	Exp 2
36	Imp 3 - Contrastive AE	100	256 + parallelism	9600	Exp 3
37	Imp 3 - Contrastive AE	15	32 + parallelism	16	Exp 4
38	Imp 3 - Contrastive AE	100	256 + parallelism	128	Exp 5
39	Imp 3 - Contrastive AE	10	256 + parallelism	750	Exp 6

Table 6.13: Models tested for the Autoencoder

Implementation 1

Nb	Exp.	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)	t_s^m (s)
18	1	Rand.	8.82 ± 0.8	19.17 ± 1	19.72 ± 0.98	37.04 ± 1.25	1.18 ± 0.09
19	2a	Rand.	6.04 ± 0.61	11.27 ± 0.72	13.94 ± 0.7	22.9 ± 0.73	1.97 ± 0.1
20	2b	Rand.	6.19 ± 0.52	11.19 ± 0.66	14.69 ± 0.71	23.15 ± 0.8	3.162 ± 0.09
21	3a	Rand.	2.4 ± 0.3	9.7 ± 0.75	8.17 ± 0.6	26.08 ± 0.97	2.45 ± 0.09
22	3b	Rand.	16.34 ± 1	31.43 ± 0.89	35.46 ± 0.74	52.54 ± 0.76	2.18 ± 0.1
23	4	Rand.	8.93 ± 0.72	16.4 ± 0.71	21.31 ± 0.73	33.02 ± 1.14	3.162 ± 0.09
18	1	Weig.	7.2	16.02	17.82	33.53	110.8
18	1	Def.	30.47	45.09	38.57	59.16	111.18

Table 6.14: Results for the model using the first implementation of the AE

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
18	20	23.47	1.16
19	20	56.56	1.93
20	36	110	3.1

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
21	9	18.36	2.4
22	9	17.04	2.13
23	62	109.75	3.1

Table 6.15: Time results for the first implementation of the AE

Nb features	128	25088	1000
t_s^{db}/query (ms)	3	500	20
t_i^{db} (s)	5-7	11	7-8

Table 6.16: Search time per query given the size of the feature vector

The results of the different experiments made on the first implementation of an Autoencoder architecture are presented in Table 6.14. It must be taken into consideration that the models are trained without using the labels while those labels are used to quantify their results, hence leading to some bias. Still, the results obtained with this implementation are disappointing. Furthermore, each additional experience was executed in the hope to get better results than the previous ones, without much success. This means that regardless of the setting or the parameters, this implementation of the AE did not seem to truly learn anything.

The first experience (18) consists in taking the vgg16-based autoencoder and changing the kernels' dimensions to obtain a feature vector of the same size as the usual one. Given its poor results, the second and third experiments are conceived, thinking that the results might be due to the change in the size of the kernels. They thus consist in testing the two original VGG-based AEs (19 - 20). Again, the results are not up to par and it is decided to test a Resnet18-based architecture (21), as the second hypothesis to explain those poor results would be a non-fitting backbone. This time, the results are even poorer than previously. Changes to the training strategy are then introduced, first by testing a pre-trained version (23), then by restricting the training set to see if the model manages to learn better when submitted to fewer data. For this, model 24 is trained on all images except the images belonging to the class *Janowczyk6_0*, the most important of the dataset. It was decided based on the t-SNE of the previous model that showed that class is present in all clusters, leading to believe that it was confusing the model. Model 25 is trained on one single class of the dataset (*cell_no_aug_0*) while Model 26 is trained on a single project (*cells_no_aug*). The pretraining (23) improves the results a bit but it is still below what is wanted. Models 24-25-26 did not get better results on the specific data they were trained on[5]. Indeed, the top_1 accuracy of the class *cells_no_aug_0* is 33% for model 26, 28%

[5]As nothing was learned from those results, they are not presented in the Table.

for model 25, and 39% for model 23, which was trained on the full dataset, with all the data indexed in the database. While it could be argued that it makes sense that Model 23 does better considering that all classes are used for the indexed data, looking at the results of Model 26 when only the project *cells_no_aug* is indexed shows that it does not manage to learn well, even on such a small dataset, with the top_1 accuracy of the second class of the project being only 10%. For Model 25, indexing only the class of interest and retrieving similar images to a query of that class was also disappointing, with the retrieved images not being the most similar available in the database. It shows either that the architecture is not good at learning or that the reduced training data was not enough to learn effectively. Finally, the last experience (27) was designed to reduce the dimension of the feature vector without modifying the parameters of the backbone architecture. Indeed, from looking at the reconstructed images, it was obvious that all the models were learning pretty well as the reconstructed images were really good (Figure 6.3). From this, it was hypothesized that the feature number of the models (except the first)

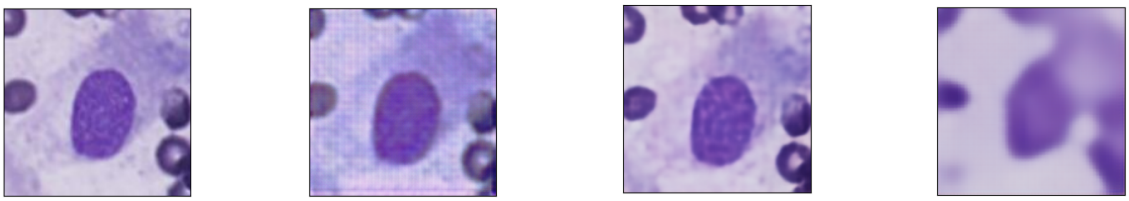

Figure 6.3: Original image and reconstructed images using the models 22, 23 and 27

was too big and the vector actually did not represent a summary of the image at all, and instead of retrieving the best features, it kept almost identically the original pixel values. This would explain why the reconstructions were good (basically no information loss between the original and the latent representations) while the search results are terrible (basically it consisted of a pixel-wise distance computation). Furthermore, as can be seen in Table 6.16, due to the number of features, the search time was becoming prohibitive. Unfortunately, the method to reduce the dimension did not seem to have been well designed and it lead to terrible results (less than 5% accuracy for all metrics, hence not even presented here), though the reconstructed image (Figure 6.3) shows that the model has learned from the data. Based on that it was decided to drop this implementation and to look for others.

Experience 3b (22) was actually the last tested one (and after the other two AE implementations), after thinking that maybe the scheduler was the issue, as in [Defraire, 2021], changing from the exponential scheduler to the step or no scheduler would dramatically affect the results. And this experiment gave better results, though still not the best but a clear improvement compared to the other experiences and in particular compared to Experience 3a (21) which was the same model except for the scheduler. In Table 6.15[6], it can also be seen that model 22 is one of the quickest, for all three time metrics presented. Unfortunately, due to lack of time, the other experiments were not remade using that scheduler.

Implementation 2

All experiments were realized sequentially, as a way to see if changing some parameters 'for the better' would improve the results. It is the case for the first one which has the worst results. This was expected as this first experiment was reusing the original implementation which was built for another dataset, making its input dimensions differ from the input dimensions of the

[6]The times for models 24-25-26 are not included because those models were only used to try and find out how the implementation learns, rather than in the objective of being used in the final framework. For model 27, it was not included due to the very poor results.

Nb	Exp.	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)	t_s^m (s)
28	1	Rand.	13.69 ± 0.53	23.37 ± 0.66	29.8 ± 0.8	41.99 ± 0.78	0.19 ± 0.05
29	2a	Rand.	27.82 ± 0.95	52.71 ± 1.18	58.3 ± 1.3	77.93 ± 1.19	0.31 ± 0.05
29	2a	Weig.	28.59	50.76	58.16	75.11	31.27
30	2b	Weig.	26.35	47.05	49.12	69.78	54.42
31	2c	Weig.	26.99	46.86	50.7	70.81	99.12
32	3	Weig.	23.37	37.62	41.9	56.26	193.64
29	2a	Def.	48.84	73.42	61.95	84.12	31.21
30	2b	Def.	45.53	71.84	56.68	82.38	54.44
31	2c	Def.	47.85	71.53	59.81	81.75	99.07
32	3	Def.	47.12	64.3	56.7	72.2	194.1

Table 6.17: Results for the model using the second implementation of the AE

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
28	11	2.12	1.9
29	9	2.12	3.3
30	30	2.6	5.7

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
31	30	3.97	1.03
32	50	34.91	2.02

Table 6.18: Time results for the second implementation of the AE

Nb features	20	128	3840
t_s^{db}/query (ms)	0.6	2.97	79.7
t_i^{db} (s)	5-6	5-7	7-8

Table 6.19: Search time per query given the size of the feature vector

dataset used in this work. While it could still be applied in a somewhat logical operation (splitting the input images into subparts of correct dimensions), it was not the most appropriate strategy.

The four other experiments however offer mostly the same range of accuracies, with the 2a (29) being slightly better. The two next, 2b and 2c (30-31), have almost identical results while surprisingly, the third experiment (32) is the second worst experiment of all. The slightly higher results of 2a compared to 2b and 2c seems to signify that trying to smooth the dimensional reduction, either by adding a layer (2b) or by ending up with a bigger feature vector (2c), does not bring anything and rather tends to over complicate the network for nothing. This is confirmed by the time results presented in Tables 6.18 and 6.19, where it can be seen that the experiment 2a (29) has a smaller total query time than 2b (30) and 2c (31) with a total of 3.3+0.6 = 3.9 against 5.7+0.6 = 6.3 and 1.03 + 2.97 = 4. Indeed, even if the model is three times faster when computing the query vector for 2c than for 2a due to a smaller reduction (and no layer addition), this speed gain is lost when looking at the search query time due to the bigger size of vector for 2c than for 2a. While the lack of difference between 2a and 2b in accuracy is understandable considering that in the end, the same number of features is used in both to represent the image, it was expected that 2c would get better results as its latent representation contains more features. As it is not the case, one hypothesis might be that the features learned in the 128-representation are related to one another, making the information less condensed than in the 20-representation. In the end, both representations would thus contain the same amount of information, explaining the similar results.

What is more unexpected however are the results of the last experiment. The addition of convolutional layers to the architecture was believed to be beneficial to the model granted the positive impact convolutions have shown to have in vision. Instead, it leads to a decrease in the model performance. What is even more surprising is the fact that this observed decrease is

not constant across the four metrics. The top-1 class accuracy is barely below the ones of the previous three experiments while the top-5 project accuracy is 20% below the best one. This variability is also present between protocols, with the difference in top-1 project accuracy for the *weighted* being 8% compared to the second worst but only 0.02% for the *default* protocol. Overall, the differences are smaller for the *default* than for the *weighted*. A hypothesis could be that the added convolutions make the model focus more on details, making the top-1 class accuracy on par with the other experiments but this also makes the model more sensitive to class imbalance.

Again, it can be seen that none of them handle well the class imbalance, with quite severe drops (between 10 to 20%) between the *weighted* and *default* protocols, on all measures.

Implementation 3

Nb	Exp.	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)	t_s^m (s)
33	0	Rand.	27.7 ± 1.1	50.31 ± 1.1	56.91 ± 1.18	76.6 ± 1.03	0.64 ± 0.05
34	1	Rand.	20.5 ± 0.9	36.49 ± 1.13	42.94 ± 0.8	59.35 ± 0.89	0.18 ± 0.046
35	2	Rand.	23.7 ± 1.2	43.89 ± 0.9	48.81 ± 1.23	67.92 ± 1.09	0.19 ± 0.049
36	3	Rand.	22.46 ± 0.76	40.99 ± 1.19	48.84 ± 1.25	66.42 ± 1.16	0.19 ± 0.047
37	4	Rand.	6.78 ± 0.44	19.34 ± 0.83	15.41 ± 0.94	38.1 ± 0.9	0.2 ± 0.1
38	5	Rand.	26.28 ± 0.88	49.92 ± 0.89	51.14 ± 0.94	72.72 ± 1.18	0.79 ± 0.1
39	6	Rand.	24.7 ± 0.83	46.48 ± 0.78	47.81 ± 1.14	67.96 ± 0.5	0.019 ± 0.1

Table 6.20: Results for the model using the third implementation of the AE

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
33	30	195.6	0.63
34	4	1.65	0.18
35	4	2.66	0.19
36	4	3.15	0.2

Nb	t_e (min)	t_i^m (s)	t_s^m/query (ms)
37	11	1.49	0.18
38	11	1.98	0.75
39	55	7.15	4.1

Table 6.21: Time results for the third implementation of the AE

Nb features	16	128	750	3072	9600
t_s^{db}/query (ms)	0.6	3	16.3	67	200
t_i^{db} (s)	5-6	6-7	6-7	7-8	9-10

Table 6.22: Search time per query given the size of the feature vector

The results of the different experiments are presented in Table 6.20. As expected, the best results are attributed to model 33, which corresponds to Experience 0, i.e. the implementation based on the adequate computation of the loss. The same model (i.e. same architecture) but based on the other loss formulation is model 34 (Experience 1). As can be seen, there is a drop between 7 and 15% between the metrics of the two models, showing the importance of the coherence between the architecture of the model and its loss function. However, as model 33 requires the multiplication of the weight matrices of each layer composing the architecture, the training time is extremely long as seen in Table 6.21 and the amount of memory is prohibitive. It effectively prevents using model 33 for the other experiments as the increase of the number of layers and/or dimensions of the layers makes the computation intractable.

Without taking into account the impact of the loss formulation, it is interesting to see the results the successive modifications of the base model (34 - Experience 1) have brought. The

second experience, represented by model 35, confirms that having smaller reductions in the dimensions of the layers is beneficial. It leads to a gain between 3.2 and 8.5% in accuracy depending on the metrics. However, combining the smaller reductions with a higher number of end features as done in Experience 3 (model 36) does not bring any advantages and even tends to decrease the accuracy on the class scale. This might mean that when using too many features, these features are less informative than what a smaller subset would be. Furthermore, due to the highest number of end features, the time taken by the model in indexing and search is higher, with 3.15s and 0.2ms against the 2.66 and 0.19 of model 35 (Table 6.21). The same is happening with the time taken by the rest of the framework, with a search time of around 200ms per query against a search time of around 67ms per query for model 35 (Table 6.22).

The following groups of experiences are all based on the modification of the dimension of the input layer to make it fit the histopathology dataset rather than the Mnist dataset for which the implementation was first conceived. Surprisingly, the first experience of this sub-group, Experience 4 and Model 37, sees its performances drop dramatically compared to the results of Model 34 (the corresponding model, with the original input size). It was not expected due to the opposite effect observed in Implementation 2. It may be due to the fact that the number of end features is really too small to represent the input image for this implementation. It seems to be the case as the next experience, Experience 5 with model 38, retrieves the range of results seen in the first experience. It is even the best model after model 33, with only around a 1% difference in results in the class scale. It indicates that adapting the dimensions of the layers to fit the dataset is beneficial, granted that the number of end features is also adapted to a reasonable number. This reasonable number must not however be too big. Experience 6, model 39, shows that using more features leads to a decrease in performance, in the same way as what happened with model 36, combined with increased indexing and search times, both for the model and for the rest of the framework (7.15s against 1.98s for indexing, 4.1ms vs 0.75ms for the model during search and 16.3ms vs 3ms for the database search itself).

Overall, the implementation benefits from an appropriately dimensioned architecture and a reasonable number of features. It would be even better if the loss was perfectly coherent with the architecture but it is not feasible due to memory constraint.

K-means

Two experiments are made with K-means, one with 67 clusters (model 41) and one with 10 (model 40), with a ResNet50 trained using DML and the Margin loss. Both models lead to vectors of 128 elements. They are taken pre-trained, without their weights frozen, and trained on the entire dataset for 15 epochs with a batch size of 256 in parallel.

Before going to the evaluations of the retrieval part, an evaluation of the new labels obtained through the clustering is presented.

For the evaluation of the quality of the clusters, different methods are used. The first one is the computation of the silhouette score which is the mean of the silhouette scores of each data point, which is given by $\frac{b-a}{\max(a,b)}$, with a the mean intra-cluster distance and b the mean nearest-cluster distance[7]. This silhouette score is 0.061 for model 40 and -0.08 for model 41, indicating that model 40 is leading to better cluster attribution (silhouette score is between -1 and 1, with 1 being the best).

The second method for analyzing the clusters is by observing the distribution of the images in them. In Figure 6.5, the image distribution in the clusters are displayed for model 40 and 41.

[7]Sklearn metrics, silhouette score

Even if neither of the two bar plots is perfectly balanced, model 40 leads to a better distribution than model 47, with the latest having some classes with less than 100 images.

Finally, the third method used to analyze the clustering is by analyzing the fragmentation of the classes and clusters. A boxplot of the number of classes contained per cluster is presented in Figure 6.4 for the two models alongside a boxplot of the number of clusters onto which each class has been mapped. For model 40, the mean of the number of classes per cluster is 48.3 and the standard deviation is 11.2. For model 41, the mean is 24.82 and the standard deviation is 15.13. Model 40 has a higher mean number, which is natural as it builds fewer clusters, and it shows less variability as its clusters contain similar numbers of classes. Model 41 has big variations, with some clusters containing only one class while another contains 53 classes (bottom right plot of Figure 6.4). It is confirmed by looking at the boxplots, with model 40 displaying a less wide one. The range of model 41 is 52 whereas the range of model 40 is 38. It is also partly due to the difference in the number of clusters but it leads to model 40 being more 'stable'.

Additional graphs are available in annexed files, 'cluster_analysis_model_40' and 'cluster_analysis_model_41'. Those files contain the graphs presented here as well as one barplot and one boxplot per class and per cluster showing the distribution respectively of the number of clusters and the number of classes. There is also a confusion matrix presenting how the images of each class have been split into the different clusters.

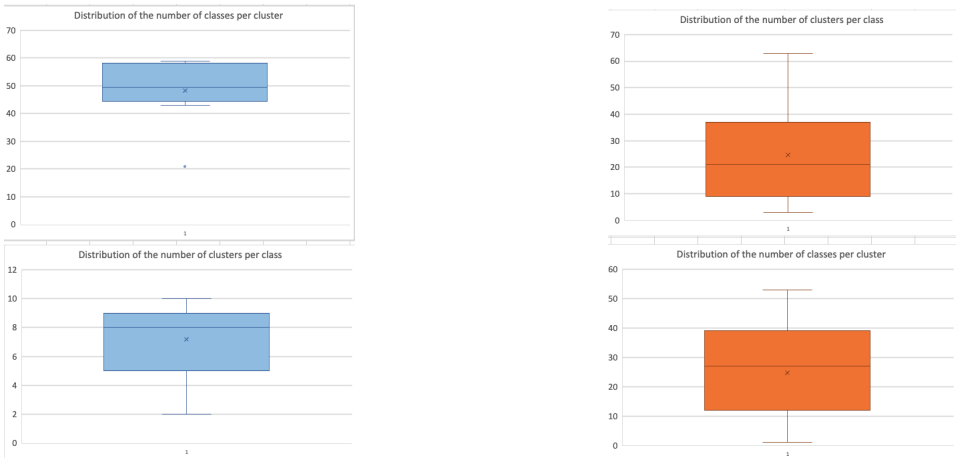

Figure 6.4: Class distribution per cluster (top row) and Cluster distribution per class (bottom row). Model 40 is in blue while model 41 is in orange.

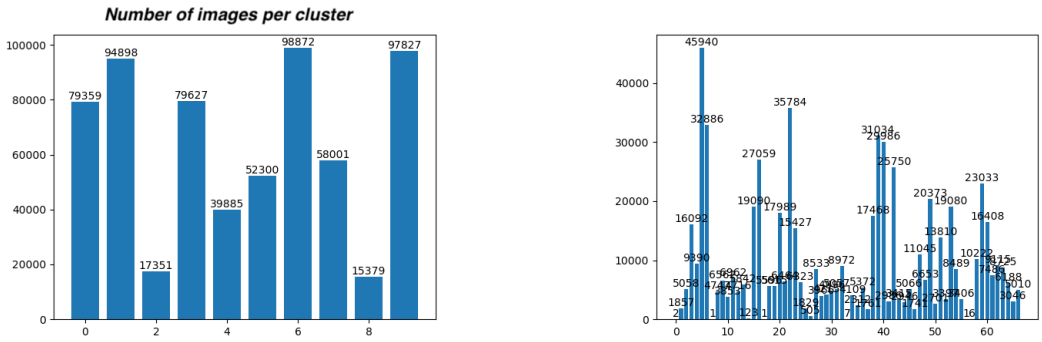

Figure 6.5: Image distribution in clusters for model 40 and 41.

The results are presented in Table 6.23. The time results of the models during the indexing and the search are not presented as they are the same as the ones of Model 1. Surprisingly, Model

Nb	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)	$\text{top1}_{newLabels}$	$\text{top5}_{newLabels}$
40	Weig.	24.86	44.68	51.76	73.5	/	/
41	Weig.	35.46	56.79	67.47	84.41	/	/
40	Def.	48.98	74.75	61.12	85.8	87.25	96.15
41	Def.	53.14	80.16	67.05	89.41	50.72	83.45

Table 6.23: Results for the model using K-means clustering

Nb	t_e (min)	$t_{K-means}$ (min)	t_{labels} (min)
40	13	35	33.5
41	13	41	37

Table 6.24: Time per epoch of the feature extraction model during training (after K-means), time of training of the K-means, time to retrieve the new labels of the training dataset.

41 reaches better results on the original metrics than Model 40, with around a 10% difference. However, its performance is falling when computing the top-1 and top-5 accuracies on the new clusters. This might be because Model 41 uses a number of clusters equal to the number of original classes. As such, its clustering might lead to the regrouping of images that originally belonged to the same class, explaining the better results on the original labels. However, it has not obtained the same distribution as the original, otherwise, the results would be the same as the results of model 1 which is far from being the case. Even though Model 41 is better on the original labels, its poor results on the new ones tend to indicate that it had difficulties learning the discriminating features of its own clusters, which brings a negative light to its capacity to learn. This is probably due to the high number of clusters making the learning difficult. Indeed, after all, the elbow plot recommended a number of clusters close to 10, which is far below 67. For this reason, Model 40 is the one that will be retained for the following. Indeed, its results show that its learning capacity is good and that it was able to discriminate well between the clusters it formed, unlike Model 41. Furthermore, the clusters' analysis deemed it slightly better in terms of stability. On top of that, Model 41's main goal was to see if the original classes were perfectly recovered using unsupervised clustering rather than being used as a feature extractor. It fulfilled its goal, showing that the original image distribution was not retrieved when using clustering, proof of the impact of the intra-class diversity discussed in Chapter 4.

ACL

Nb	Architecture	Transforms	app	batch size
42	Resnet50	custom	unique	256 + parallelism
43	Resnet50	custom	AMDIM	32
44	Resnet50	AMDIM	AMDIM	256 + parallelism
45	Resnet50	SimCRL	AMDIM	256 + parallelism
46	Resnet 50 (NC)	custom	AMDIM	32
47	BYOL network	BYOL	BYOL	64

Table 6.25: Models tested for the ACL

The models' description can be found in Table 6.25. All models are trained for 15 epochs with an Exponential scheduler using a contrastive loss. They lead to 128-element feature vectors except for BYOL (47) which leads to 256-element feature vectors. s

The results are presented in Table 6.26. A first remark is that these models were trained before all the variants of contrastive learning loss were tested. The contrastive loss was thus

Nb	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)	t_s^m (s)
42	Weig.	6.21	16.92	17.76	39.11	259.21
43	Weig.	29.58	51.83	57.46	77.55	258.17
44	Weig.	25.73	48.95	58.46	74.8	259.9
45	Weig.	19.42	37.59	32.93	58.25	259.06
46	Weig.	9.69	22.62	27.07	50.02	259.25
47	Weig.	41.24	63.17	77.52	88.08	274.13
42	Def.	39.73	69.09	54.76	79.84	257.89
43	Def.	63.63	83.86	81.11	91.01	260.01
44	Def.	55.64	79.18	73	89.09	258.16
45	Def.	47.15	68.23	57.51	78.83	260.03
46	Def.	36.22	66.6	46.93	78.98	259.98
47	Def.	68.08	85.87	86.45	92.37	274.06

Table 6.26: Results for the model trained using Augmented (Non) Contrastive Learning

selected in this case as it is the most renowned loss for contrastive learning. In retrospect, given the results in Table 6.10, the NT-Xent loss should have probably been chosen to train those models, which would have likely resulted in better performances.

Now, for the different techniques and transforms, a first observation is that the 'Unique' application (42) leads to terrible results. One hypothesis that could explain this is the fact that only the positive sample is obtained from the data augmentations. The anchor might then end up closer to the negative sample than to the positive sample as both the negative and the anchor have gone through the same process, unlike the positive sample. Secondly, the best-fitted transform for the histopathology dataset seems to be the custom one (43) as it reaches the highest results on three out of four accuracies. It is closely followed by the AMDIM transforms while the results drop with the SimCLR transforms, especially in terms of project accuracy. One potential explanation behind the failure of SimCLR is the absence of normalization, present in both other transforms.

Model 46 did unpredictably 'well'. As it was a model constructed only to see the impact of dimensional collapse, the results were expected to be much worse than they are. It is still much lower than its contrastive counterpart, model 43, as predicted, but it reaches better results than model 42 which is using negative pairs and should have arrived on top of it.

For model 47, it was the first complete non-contrastive model tested in this work and it did pretty well, getting first place by far on all metrics and all protocols. This shows the important impact that the choice of the negative samples can have on training as removing them improves the results in this case.

In terms of time results, there is no difference in the model times during indexing and searching between them because all models use a Resnet50 as backbone architecture, which makes all models execute the same operation when computing the feature vectors. The time taken for the search in the database is twice as high for Byol than for the others as its feature vectors are twice the size of the feature vectors of the other models. For the indexing, it has a smaller impact due to the batch insertion, with the indexing time of Byol being 17% higher than the indexing time of the others (4.5s against 3.86s).

Discussion on the self-supervised approaches

Three methods are tested for self-supervised learning: Auto-Encoders, K-means, and Augmented Contrastive Learning, each disposing of different configurations as was described earlier.

The results per implementation have been discussed before. This section first compares the three implementations of the AE, before comparing the best models of each method together.

The best models of each implementation, i.e. Model 22 for Implementation 1, Model 29 for Implementation 2, and Models 33 and 38 for Implementation 3, are used as representatives of their implementation (Table 6.27). Model 22 is taking third place, being far below the three other models. Model 29 is coming up first but by a very small margin. Those results are reflected in their embeddings representations (Figure 6.6). The groups in Model 29 are more distinct than in the others. Surprisingly, the representations of Model 22 and Model 38 both lack properly defined groups, while Model 38 reached better results than Model 22. One last way to compare the different models is by visualizing the images they retrieved given a single query (Figure 6.7). Visually, it seems like Model 29 retrieved the images the most similar to the query in the four that were retrieved. The returned image of Model 38 ranks second while Models 22 and 33 have returned nearly identical results, a little bit further away from the query, visually. Other tests with different queries have confirmed the tendency of Model 29 to rank first while the three other models are more difficult to rank. This confirms that Model 29 seems to be the best implementation of the AE presented in this work. However, this visual analysis also shows that using labels to evaluate self-supervised methods is tricky and may lead to false conclusions. Indeed, the quantitative results of Model 22 are lower than the results of Models 33 and 38 while their qualitative results are similar. Similarly, the other experiences for implementation 1 have shown pretty terrible quantitative results while in reality, their qualitative results, while being below the results of Model 22 (same implementation), are not completely without sense. In Figure 6.7, the last image is the top_1 result from Model 23. While it is further from the query compared to the results of the other models, the top left corner of the image displays structures identical to the structures present in the top left of the query (purple circles), which tends to indicate that the model has managed to learn and represent distinctive features of the images (Note that more examples of retrieved images for several models are available in Appendix E).

Nb	Exp.	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)	t_s^m (s)
22	3b	Rand.	16.34± 1	31.43±0.89	35.46±0.74	52.54±0.76	2.18
29	2a	Rand.	27.82 ± 0.95	52.71 ± 1.18	58.3 ± 1.3	77.93 ± 1.19	0.31 ± 0.05
33	0	Rand.	27.7 ± 1.1	50.31 ± 1.1	56.91 ± 1.18	76.6 ± 1.03	0.64 ± 0.05
38	5	Rand.	26.28 ± 0.88	49.92 ± 0.89	51.14 ± 0.94	72.72 ± 1.18	0.79 ± 0.1

Table 6.27: Best models of the three AE implementations

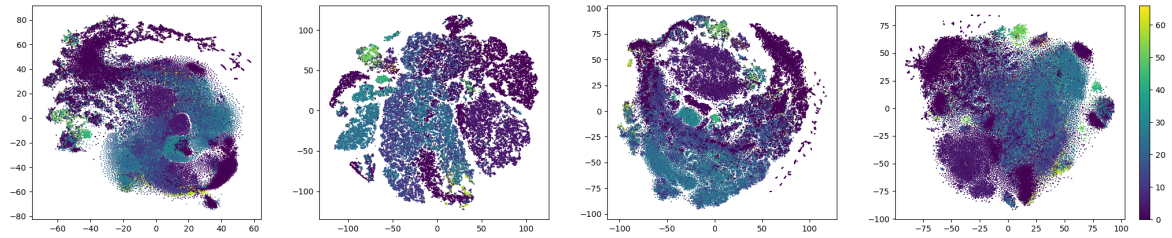

Figure 6.6: T-SNE maps for model 22, model 29, model 33 and model 38

Figure 6.7: Query image (leftmost) and the top_1 images for model 22 - 29 - 33 - 38 - 23

Interestingly, the quality of the reconstructed images is not a good indicator at all of the results that a model will get (Figure 6.8). Indeed, Model 23 had quite good reconstructed images as shown in Figure 6.3. The opposite can be said about the reconstructions of Models 29 and 33. Model 38 has a slightly better representation, but is still far from the quality of the representation of Model 23 (Note that the reconstruction was tested on several images, which all led to a similar ranking of the reconstructions). This might indicate that the choice of an AE

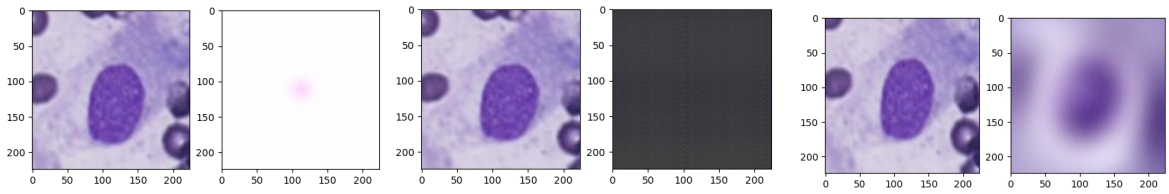

Figure 6.8: Original images and their reconstruction by the models 29 - 33 - 38

for CBIR is not the most appropriate as the best AE in the sense of the reconstructed images (which is the primary goal of an AE) is not the best when used for CBIR. However, it might also be due to other parameters such as the feature numbers as stated earlier.

To compare now the three methods, the best models are again taken as representatives: model 29 for AE, model 40 for K-means, and model 47 for ACL. Quantitatively (Table 6.28),

Nb	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)
29	Weig.	28.59	50.76	58.16	75.11
40	Weig.	24.86	44.68	51.76	73.5
47	Weig.	41.24	63.17	77.52	88.08

Table 6.28: Results for the best models of each self-supervised method

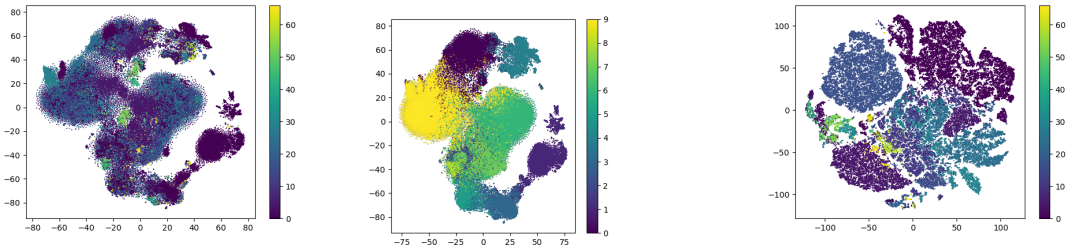

Figure 6.9: T-SNE maps for model 40 (original labels and K-means labels) and model 47

Model 47 is way above the two other models with a difference between 10 and 20 %. This is also reflected in the embeddings with Model 47 having very distinct groups for the original classes while Model 29 and 40 do not (Figure 6.6 and Figure 6.9). Note that Model 40 does lead to distinct groups but on the new labels generated by K-means. This, rather than showing the capacity of the model to differentiate the data in groups by itself, shows more the effectiveness of the supervised training concept, DML, in learning embeddings that discriminate well the given labels. It is nevertheless interesting to compare the two t-SNE maps of Model 40 as it shows how are split the original labels in the new labels. As can be seen, the distribution seems quite uniform, which is coherent with the cluster analysis done previously. Comparing their retrieved images for the same query, Figure 6.10, shows that all three models have done reasonably well. At first sight, 29 seems to be the closest but Model 47 is quite as close if the white element in its retrieved image is abstracted (the white element tends to draw the attention and makes the retrieved image look much lighter than the query when the background is actually in the

same tone). It seems like Model 47 paid more attention to the elements in the image than in its background (while still having a background coherent with the one of the query) while Model 29 did the opposite, with a closer background but elements less close. Model 40 however, is further both in terms of background and terms of elements, tending to confirm its last place in the quantitative results. Other tests with other queries were also conducted for this comparison and have shown Model 47 to be slightly better although Model 29 is quite close to it.

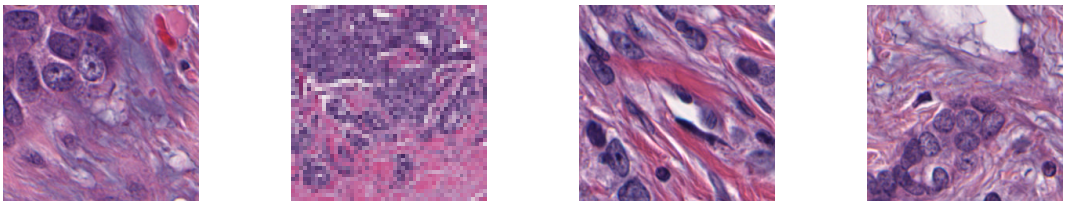

Figure 6.10: Query image (leftmost) and the top_1 images for model 29 - 40 - 47

In conclusion, regarding the self-supervised models, model 47, Byol (ACL), appears to be on top, followed closely (qualitatively, less closely quantitatively) by model 29.

6.3 Discussion on the difference between supervised and unsupervised models results

It is quite complicated to evaluate impartially the results of the supervised It is quite complicated to evaluate impartially the results of the supervised models and the results of the unsupervised models. Indeed, quantitative results will obviously tend to favor supervised models. Those models are optimized based on the labels used for the quantitative results while self-supervised models have never been in possession of them.

And it is indeed the case with the best model of the self-supervised methods, Model 47, only reaching 41.24 % in top-1 class accuracy whereas the best supervised model, Model 9, reaches almost twice that value, with 76.78% in top-1 class accuracy. However, it is important to notice that Model 47 still beats all models trained using contrastive learning (except Model 15) and the model trained using DR. This shows that self-supervised methods have some potential and may, with the correct architectures and training methods, reach similar ranges as the supervised methods. This unfortunately is only relevant for Model 47, with all the other self-supervised models having worse results than any of the supervised models.

This domination of the supervised models is confirmed by the different t-SNE graphs presented in the previous sections (Figures 6.1, 6.2, 6.6 and 6.9). It can be observed that while supervised models present different groups, most self-supervised models have all their data points grouped into one single ensemble. Regardless of the distribution of the classes in the embedding space, the data is simply not split into groups and all samples are packed together. The exceptions are Models 29 and 47, which show a more separated distribution of the data points, starting to look like the distributions that can be observed for supervised models. Note that using these graphs for contrasting the two types of methods is again positively biased towards the supervised models due to their use of labels. Furthermore, while it may seem more interesting to have a model able to form well-defined, separate groups, it may actually not be the case for CBIR. Indeed, CBIR is concerned with finding images that are similar to a query, but it does not mean that the images similar to the query must be necessarily dissimilar to other images. Even if all the data is forming one single ensemble, as long as there are different distances between the elements of that ensemble, then similarity can be computed. It would simply mean that all

images are in some way related rather than having hard separations between groups of images, as can be observed in labeled datasets. Hence, the 'single ensemble' of the self-supervised models may be more appropriate than the hard groups of the supervised models.

To remove completely the bias of evaluating with the labels, the next step is to compare the top_1 retrieved image of each model. For that, models 1 (one of the best CNN backbones), 5 (best transformer backbone), 3 (second of the best CNN backbones), 9 (best DML loss), 11 (DR), and 15 (best CL model) of the supervised ones are selected. Models 29 (AE), 40 (K-means), and 47 (ACL) of the self-supervised are taken. First the results of the supervised models for the query used in Figure 6.7 and 6.10 are shown in Figure 6.11. Using that query, the results are quite split.

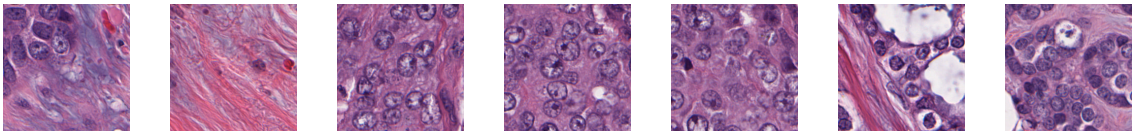

Figure 6.11: Query image (leftmost) and the top_1 images for model 1 - 3 - 5 - 9 - 11 - 15

It is difficult to say if the supervised models are better or not with respect to the self-supervised ones. The retrieved image of Model 1 is the furthest of all models. Model 11's image is not that close either; even when abstracting the white elements that give a lighter aspect to the image, it stays quite far compared to the other models. Models 3, 5, 9, and 15 however have retrieved quite good images which, if they are closer than the retrieved images of the self-supervised, is not by far. These four models seem to have mostly focused on the elements, like Model 47 did, with the most similar element between the query and their retrieved images being the purple structures. In Figures 6.12, 6.13, 6.14 and 6.15, the results for the same models using two other queries are presented. In Figures 6.12 and 6.13, Model 29 is disappointing, with the least similar images of all. Models 11 and 47 return images similar to one another though less similar to the query itself. However, especially for Model 47, some darker points can be noticed that appear to be identical to the darker points in the query, probably explaining why the images were considered similar. Models 1 and 5 are the most satisfactory, with models 9, 15, and 40 not so far. Model 3 is in the middle, with the theme of its image matching the query but several elements making it dissimilar. In Figures 6.14 and 6.15, the first observation is that almost all models do really well. This shows how much the results can be impacted by class imbalance as this query belongs to one of the two main classes of the dataset (*camelyon16_0*). Only Model 40's result is not that great with barely any resemblance with the query image, except for the color theme and the white element in the center that looks similar to the white element on the bottom of the query. It is quite hard to rank the other results because they are all really good. Models 1, 29, and 3 seem a little bit better, followed closely by Models 15, 5, and 47. Finally, Models 9 and 11 take the final places, right before Model 40. This shows that the self-supervised models, despite their not-so-great quantitative results can still lead to great qualitative results.

Overall, the supervised models as of now still lead to better results and are more consistent, with relevant retrieved images for most queries. This was expected because supervised models have benefited from more research and time spent on finding techniques that make them work well in a diverse range of situations, CBIR included. Compared to that, self-supervised techniques are relatively new and have not received yet the same amount of attention as supervised ones have, explaining why they do not reach yet the same level of performance. However, the quantitative results still show potential and the qualitative results demonstrate the capacity of the self-supervised models to learn directly from the content of the images, as hoped. Combining all results, model 1 and model 9 seem to be the best supervised models while model 47 seems to win on the self-supervised side.

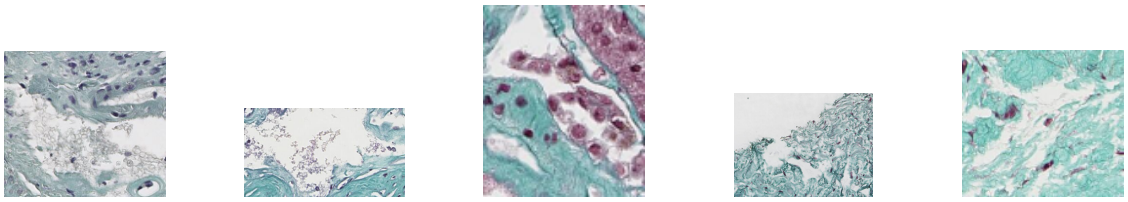

Figure 6.12: Query image (leftmost) and the top_1 images for model 1 - 3 - 5 - 9

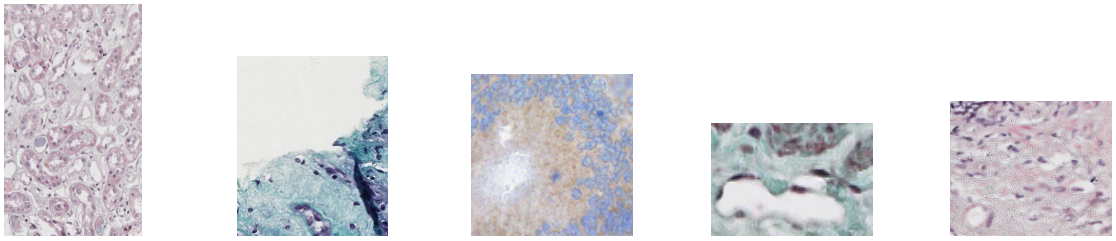

Figure 6.13: Top_1 images for model 11 - 15 - 29 - 40 - 47

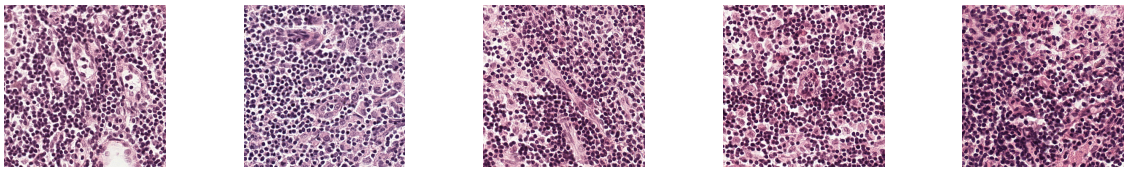

Figure 6.14: Query image (leftmost) and the top_1 images for model 1 - 3 - 5 - 9

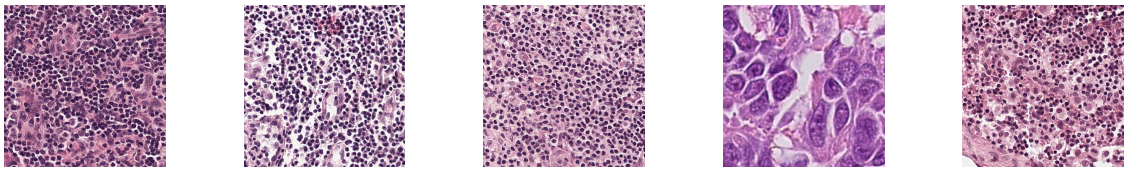

Figure 6.15: Top_1 images for model 11 - 15 - 29 - 40 - 47

6.4 Impact of training data

This section discussed the impact of the training data on the results. It is split in two. First, it discusses the effect of pretraining, transfer learning, and fine-tuning before discussing the generalization quality. The models tested are presented in Table 6.29. They are trained on 15 epochs with an Exponential scheduler and a batch size of 256 split on 2 GPUs. They all lead to vectors of 128 elements except for BYOL which gives a 256-element vector.

For the generalization models, they are trained on half the classes as listed in the description of the protocols (Chapter 5, Section 5.4). The images of the other classes are selected to be indexed and used as queries.

Type of training

The results are presented in Table 6.30. They are as expected, with the fine-tuned model (48) reaching the best results in all metrics and both protocols, followed by the Transfer learning model (49) and finishing with the model trained from scratch (50). The training time is also coherent, with the model whose weights were frozen (49) being the quickest as fewer parameters had to be updated at each epoch.

Nb	Architecture	Training	Pretraining
48.0	Resnet50	Entire set	Fine-tuning
49	Resnet50	Entire set	Transfer Learning
50	Resnet50	Entire set	From scratch
51	Resnet50	1^{st} generalisation	Fine-tuning
52	Resnet50	2^{nd} generalisation	Fine-tuning
53	Kmeans - Resnet50	2^{nd} generalisation	Fine-tuning
54	AE - Imp. 2 - Exp 2c	2^{nd} generalisation	Fine-tuning
55	Byol	2^{nd} generalization	Fine-tuning

Table 6.29: Models tested for generalization purpose

Nb	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)	t_e (s)
48	Weig.	78.21	87.14	94.65	97.02	14
49	Weig.	56.35	76.6	88.27	95.55	8
50	Weig.	34.75	62.29	67.21	85.16	13
48.0	Def.	81.04	94.14	94.32	98.12	14
49	Def.	72.41	90.75	90.98	95.96	8
50	Def.	59.29	83.58	76.16	92.63	13

Table 6.30: Results for the different types of training

Generalization

Nb	Nb_{eq}	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)
51	48	Weig.	52.1	71.9	90.07	96.4
52	48	Weig.	65.96	89.48	88.6	98.75
53	40	Weig.	49.95	69.24	75.2	84.77
54	29	Weig.	31.14	60	49.59	73.98
55	47	Weig.	61.12	81.06	89.37	94.84
51	48	Def.	75.64	93.33	94.08	98.99
52	48	Def.	74.8	94.06	94.44	98.63
53	40	Def.	69.49	91.22	90.86	97.31
54	29	Def.	54.65	78.59	69.73	87.91
55	47	Def.	69.33	91.5	93.29	97.67
53	40	new labels	88.95	96.69	/	/

Table 6.31: Results for generalization of the different architectures

The results are presented in Table 6.31[8]. Several observations can be made. First, despite the generalization, class imbalance still impacts the results, with the weighted protocols having worse accuracies than the default which means that even without being trained on them, the retrieval of major classes is better than the retrieval of the minor classes. Second, looking at Models 51 and 52, it can be seen that the first protocol leads to very different results compared to the second protocol. This means that the choice of the classes on which to train and on which to test is important, with some classes leading to better generalization than others. Third, the effects of generalization are opposed on the supervised models and self-supervised. The supervised model, 51 and 52, sees its accuracies decrease when tested on different classes than trained. Self-supervised models, 53, 54, and 55, see their accuracies actually increase when tested and trained on different classes. While it was expected that self-supervised models would suffer to a lesser

[8] Nb_{eq} represents the number of the model of same architecture but trained on the entirety of the dataset

extent than supervised models, as they do not rely on labels, it was not expected that the results would improve. To explain this, one hypothesis is that as half the indexing set and query set were removed and only half as many classes were present in the dataset, it is actually much likelier to retrieve an image of the same class as the query, leading to better results. In conclusion, self-supervised models cope quite well with predicting unknown data whereas supervised models see their performance reduced quite strongly.

6.5 Impact of Batch size and parallelism

Nb	batch size	Parallelism
1	32	No
56	128	No
57	32	Yes
48.0	256	Yes

Table 6.32: Models used for Parallelism and batch size effect

The models are presented in Table 6.32. They all use a supervised Resnet50, trained using DML and Margin loss. They are trained on the entire dataset for 15 epochs using exponential scheduling. They are also taken pre-trained without their weights frozen.

Nb	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)	t_e (s)
1	Def.	75.89	92.93	94.83	98	20
56	Def.	82.38	94.17	95	98.21	19
57	Def.	75.81	92.54	94.62	97.48	22
48.0	Def.	81.04	94.14	94.32	98.12	14
1	Weig.	72.46	86.56	95.42	97.93	20
56	Weig.	75.29	86.39	96.14	97.96	19
57	Weig.	73	87.37	95.46	97.25	22
48.0	Weig.	78.21	87.14	94.65	97.02	13

Table 6.33: Accuracy Results for different batch sizes and set-ups

The results are presented in Table 6.33. Globally, all accuracy results look the same, showing that batch size and parallelism do not affect the accuracy that much. It can still be seen that in top_1 at the class level, the bigger batch size models (56 and 48.0) have slightly better results. It is especially visible with the *default* protocol where the difference is up to 7% while the difference in the *weighted* protocol is barely around 2-3%. This may be caused by the imbalance of the dataset used for training. Indeed, the batches are composed using informative sampling. When the batch size is small, the batches will be more informative than when they are large because it will be easier to get an equal number of samples for each class. When batch sizes are too big, there might be classes whose number of samples is not big enough to contribute in an equal way to the other classes. This makes the smaller batch size models more resistant to the imbalance and their results are quite similar for the two protocols, also explaining why the discrepancy in the results is higher for the *default* protocol[9].

For the time results, it can be seen that parallelism greatly reduces the training time when associated with big enough batch sizes. When the batch size is too small (model 57), then the

[9]If small batch size models have the same range in both protocols and that this range is the same than the range of the high batch size models for the weighted protocols, it must mean that the ranges will be different in the default.

overhead operations due to the parallelism actually takes more time than the time saved by parallelizing the operations.

6.6 Impact of class imbalance

The class imbalance has already been discussed a bit when discussing each model's results. It was observed that the class imbalance could lead to biased results with the results obtained through the *default* protocol being always better, and sometimes by more than 20%, than the results of the *weighted* protocol (*weighted* has for effect to make each class have the same impact on the total results while *default* gives each image the same impact and hence, classes with more images have more impact).
To investigate if, as hypothesized, the largest classes have the best results and the smallest the worst, the results per class are generated using model 1 (best supervised), model 4 (has shown one of the biggest discrepancies between the two protocols) and model 47 (the best self-supervised). The Top_1 class accuracies are presented in Table 6.34[10].

Class	1	4	47	Class	1	4	47	Class	1	4	47
cam_0	97.01	97.01	95.11	mitos_0	8.44	9.7	7.38	chimio_0	100	95.65	56.52
cam_1	57.39	42.34	32.67	mitos_1	26.08	36.18	34.2	chimio_1	85.71	92.86	50
cells_0	91.74	84.3	62.81	mitos_2	75.44	65.55	59.05	ulg_lbtd_0	97.56	91.46	70.73
cells_1	80.77	44.23	28.85	patt_0	88.3	72.34	53.19	ulg_lbtd_1	68.75	12.5	6.25
glom_0	98.71	93.4	90.37	patt_1	49.32	46.58	43.84	ulg_lbtd_2	86.47	17.39	14.01
glom_1	92.5	83.63	61.74	tupac_0	70.11	65.5	50.97	ulg_lbtd_3	77.89	34.74	26.32
iciar_0	82.14	71.03	51.59	tupac_1	21.03	21.54	11.67	ulg_lbtd_4	96.93	76.07	65.64
iciar_1	48.33	39.17	22.5	ulb_0	91.37	71.22	10.7	ulg_lbtd_5	90.07	46.3	14.81
iciar_2	62.5	45.83	31.25	ulb_1	97.56	70.73	29.27	ulg_lbtd_6	0	33.33	0
iciar_3	43.52	33.33	13.89	ulb_2	100	23.08	7.69	ulg_lbtd_7	89.06	62.5	43.75
jano1_0	50.55	59.69	34.44	ulb_3	84.78	50	28.26	umcm_0	46.94	22.45	0
jano1_1	63.4	55.94	33.93	ulb_4	94.12	52.94	5.88	umcm_1	100	100	100
jano2_0	61.9	80.95	67.62	ulb_5	0	0	0	umcm_2	100	100	100
jano2_1	83.67	82.33	36	ulb_6	25	12.5	25	umcm_3	100	0	100
jano5_0	19.32	20.37	10	ulb_7	0	0	0	umcm_4	100	100	100
jano5_1	2.44	2.09	0.58	ulb_8	75	8.33	16.67	umcm_5	50	50	100
jano6_0	89	88.4	86.57	bone_0	87.5	75	0	umcm_6	100	100	100
jano6_1	65.63	68.88	54.89	bone_1	66.67	22.22	0	umcm_7	100	100	0
jano7_0	85.61	73.48	59.85	bone_2	100	25	12.5	warw_0	60.56	59.86	48.59
jano7_1	79.89	74.71	67.82	bone_3	75	25	12.5	warw_1	74.07	70.83	46.3
jano7_2	75.67	77.33	57.33	bone_4	88.89	66.67	22.22				
lbps_0	99.28	98.92	93.19	bone_5	77.78	75	38.89				
lbps_1	99.22	98.44	90.63	bone_6	85	90	50				

Table 6.34: Top_1 accuracy per class for the models 1, 4 and 47

It can be seen that the results are very different depending on the class. As expected, the two main classes have high results, above 95% for *camelyon16_0* for all three models and above 86% for *janowczyk6_0* for all three models. Given that by themselves they represent 50% of the query dataset, it is no surprise that in the *default* protocol, they tend to increase the results.

For the rest of the classes, the results are not necessarily linked with the size of the classes directly. It is especially visible in Figure 6.16 where no relationship can be established between the points, for any of the model. Looking at specific cases from Table 6.34, for example, *ulg_lbtd_4* has above 65% in all models and it only possesses 520 images while *mitos_1*'s highest result is 36.18% and it contains 15020 images, making it one of the biggest classes. What is interesting is that *mitos_2* has great results compared to *mitos_0* and *mitos_1*. It is also the class of the *mitos* project that has the most images. So while the size does not necessarily explain the

[10]The results for all the metrics can be found in the excel file 'results_per_class.xlsl' in the annexed files

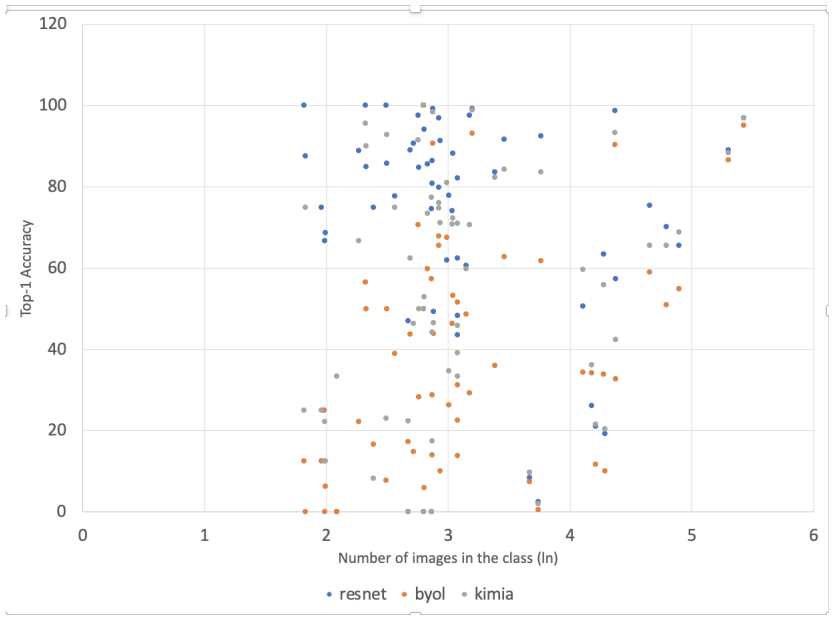

Figure 6.16: Top_1 accuracy per class given the number of images in the class, for the models 1 (ResNet), 4 (KimiaNet) and 47 (Byol)

results when comparing all sizes of all classes and all results of all classes, it becomes an important variable when we restrict the view to one single project. In a project, the class with more images tends to have better results than the classes of the same project with fewer images. From this, it can be inferred that while the models can distinguish quite well the projects, they have more trouble distinguishing the classes inside the same project, tending to learn better how to represent the major class of the project and as such, have better retrieval results for that class. This is confirmed by looking at the Top_1 project accuracy vs the Top_1 class accuracy for all models, with the first one being much better (around 20-30% higher).

Note that this is also a direct consequence of the intra-class diversity discussed in Chapter 4. As some classes have very diverse images, sometimes images from another class of the same project tend to be more similar than an image from the same class. For example, in Figure 6.17 is represented a query image from *mitos_ 0* and the top_1 retrieved image by the three models. All three models return images that are very similar visually to the query. Yet, none of them is of the same class, with all three retrieved images belonging to the major class of the project, *mitos_ 2*. The opposite happened in Figures 6.12 and 6.13. All retrieved images except for the one of Model 29 are of the same class, *glom_ 0* as the query. Yet, the retrieved images of Model 11 and Model 47 are quite different from it. Meaning that in this case, despite the diversity present in the class, the models still managed to retrieve images of the same class. But again, like *mitos_ 2*, *glom_ 0* is actually the dominant class of its project, which would explain why the models will by default retrieve images from that class instead of images of the *glom_ 1* class that might be more similar to the query. The models have preferably learned the representation of the dominant class of each project, leading to better results for these classes.

Note that as seen in the quantitative results for the CNN backbone architectures, KimiaNet leads to worse results than ResNet, with a much higher number of data points with low accuracy. It also presents a bigger range in the accuracy, explaining its important discrepancy between the results of the weighted and default protocols.

Note that while the *weighted* protocol allows avoiding the dominance of the two main classes, it is also biased. Indeed, in Table 6.34, it can be seen that some classes have very different results

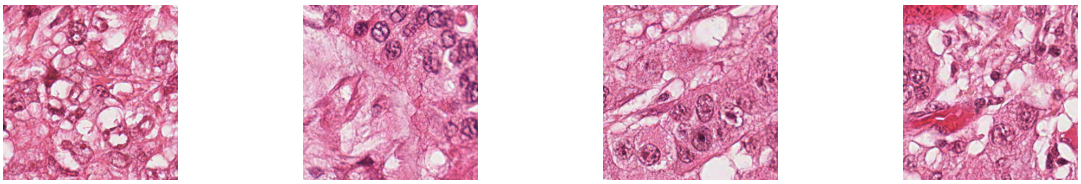

Figure 6.17: Query and Top_1 images for model 1 - 4 - 47

for the three models, the extreme being 100% with two models and 0 with the last. These classes are composed of only a few images, sometimes even only one. Hence, by miss retrieving this one image, the result for the class directly becomes zero while correctly retrieving it gives the class a 100% result. And in the *weighted* protocol, these classes have the same weights as classes with thousands of images whose results are much more stable. This also explains why the results of the weighted protocol are smaller than for the *default*, as a few classes with 0 as the result would not matter much in *default* due to containing barely a few images while they amount for 1/67th in *weighted*.

6.7 Search and Retrieval

Nb	Architecture	Feature number	Indexed Set	# images	FAISS Training
48	Resnet 50	128	Indexing	106 281	No
48.1	Resnet 50	128	Indexing	106 281	Yes
48.2	Resnet 50	128	Train + Indexing	739 780	No
48.3	Resnet 50	128	Train + Indexing	739 780	Yes
36	AE - imp. 2 - Exp 6	9600	Indexing	106 281	No
36.1	AE - imp. 2 - Exp 6	9600	Train + test	739 780	Yes

Table 6.35: Models tested for Search

The models are described in Table 6.35. The supervised Resnet50s are trained using DML and the margin loss. They are all trained using exponential scheduling on 15 epochs, with a batch size of 256 split onto 2 GPUs. Several observations can be drawn from the results in Table

Nb	Prot.	top1 (%)	top5 (%)	top1_proj (%)	top5_proj (%)	t_i^{db} (s)	t_s^{db}/query (ms)
48.0	Weig.	78.21	87.14	94.65	97.02	3.93	3.06
48.1	Weig.	78.22	87.14	94.67	97.01	3.93	0.39
48.1	Weig.	78.28	86.17	93.62	96.27	27.5	19.75
48.2	Weig.	78.3	86.16	93.64	96.27	27.5	2.066
36	Rand.	22.35	41.96	49.6	67.94	8.94	197
36.1	Rand.	23.13	44.7	47.5	67.94	56.08	1 324

Table 6.36: Accuracy Results for different batch sizes and set-ups

6.36. First, not really related to the search itself, it can be seen that the results of the models are quite stable with almost the same results with only the indexing set indexed and when both the train and indexing sets are indexed. Second, using the second type of FAISS index, the inverted index that requires clustering training barely changes the results. The difference is at most 2% between the brute-force index and the inverted one. However, the search is much faster, a factor 10 of difference **per query**, regardless of the amount of data inside (i.e. the gain of the training of the index is the same regardless of the amount indexed in the database (48 vs 48.1 and 48.2 vs 48.3)). This suggests that using the inverted index is purely beneficial and greatly improves

the efficiency of the system. Third, the last two models demonstrate the impact that the feature vector size has on the performance. For the same amount of data, the untrained index takes 64 times longer to treat vectors of 9600 features (36) than vectors of 128 features (48). For the trained index, it takes 640 times longer for the 9600 feature vectors (36.1) than for a 128 feature vector (48.2). Again, it demonstrates the utility of the inverted index, especially in the case of large feature vectors. If model 36.1 had been tested on the entire validation set (96 066), it would have taken 35h+ to compute the results for all queries, **with a trained index**.

6.8 Overall conclusion

This section marks the end of this Chapter. A lot of results have been presented in it. This section will quickly summarise the findings obtained from them.

superior to Transformer-based ones. DML and the softmax loss is the winning concept, though all the losses of DML are globally good. The NT-Xent loss of CL also gives good results, though worse than the results of all models of DML.

For the self-supervised models, the Augmented non-contrastive learning method wins with the Byol model, by far when looking quantitatively. It is followed by experience 2a of the second implementation of the autoencoder, itself followed closely by experiences 0 and 5 of the second implementation of the AE as well as the 10 clusters K-means model. Qualitatively, all models retrieve images that usually match well the query.

Comparing the supervised and self-supervised, the supervised models have much better results quantitatively. Qualitatively, they also tend to retrieve images that better match the query than the images retrieved by the self-supervised models but the gap is much smaller than what is shown by the quantitative results. Overall, the self-supervised models have proved to have plenty of potential and could lead to great results with a little bit more research. s

For the other parameters, fine-tuning has proven to be the best training strategy. In regards to generalization, the supervised models see their performance deteriorate whereas the self-supervised see their results improve. Globally though, all models still perform quite well on unseen data, even if it is obviously less well than on known data. The use of parallelism with big batch sizes helps speed up the training of the model without consequences on their accuracies. Same for the training of the FAISS index, it speeds up the search in the database without impacting the accuracy of the models. Finally, class imbalance stays an issue, with the dominant classes biasing positively the results obtained through the default protocol while the weighted protocol is negatively biased by the smallest classes. The size of the classes especially intervenes at the project level, with the dominant class of each project tending to be better represented/learned than the other classes of the project.

The two best models, one for each training fashion, are chosen to be model 1 (Supervised ResNet-based model trained using DML with the margin loss) and model 47 (BYOL, NACL model), though model 9 (same as model 1 but with the softmax loss) could have also been selected for supervised learning. The complete results for these two models, both quantitative and qualitative, can be found in Appendix C.

Chapter 7

Limitations and Conclusion

This chapter marks the end of this master thesis. It was developed in order to test different elements that can be used in the making of a CBIR framework. In particular, it had the purpose of looking into unsupervised and self-supervised methods to remove any need of labels and making it closer to its core concept which is to find the similarities between images based on the content alone of the images, rather than on their labels.

This study has effectively carried out its research and presentation of several different concepts and architectures that can be used for CBIR. It offers several options both in supervised and self-supervised learning. It displays their performances using numerous and varied metrics alongside an analysis providing plausible explanations to why some perform better than others.

While the self-supervised methods are still subpar compared to the supervised ones on the quantitative results, their qualitative results have confirmed their potential and the interest there is to try and develop them further. This outcome is actually even more noteworthy than obtaining great results for the self-supervised methods as part of this study as it shows that the intuition into which this work was approached was relevant and it opens the way to a new field and to new developments that could lead to great discoveries.

In the future, it could be interesting to develop further the self-supervised concepts and identify the elements that may limit their performances. Such research would still be important for supervised models as, even if they already reach great results, there is still place for improvements.

Even though this work has reached its objectives, it could be improved in different aspects. First, due to time constraints, the interest of specific elements have been discovered too late to be extended to more models/ integrated with the best other elements. Further experiences on them could have led to better results than those obtained. Such elements includes the Exponential scheduling for the second implementation of the AE and the infoNCE loss for augmented contrastive learning. On top of that, the hyperparameters and data augmentations were not tuned in any way (except the data augmentations in ACL). If they had been set to values more appropriate to the situation at hand, it might have led to better results. Especially regarding the data normalization step that uses values based on natural data rather than histopathological one. Finally, the biggest limitations of this work is the quantitative metrics used to evaluate the frameworks. The metrics all use the labels to evaluate the quality of the results of the frameworks. However, this fact makes the supervised methods more likely to have great results as they have been trained on those labels where the self-supervised methods have never seen them. It is then normal to see the supervised methods get better results per these metrics compared to the self-supervised, leading to some bias. As part of this work, it was not possible to find another way to get quantitative results without using the labels. The only other alternative found was

to display the retrieved images to compare them visually. Unfortunately, it does not allow to have full results as displaying the retrieved images for all queries would be intractable and the quality of the similarity was not evaluated by professionals which leads to incertitude as to how relevant the conclusions based on the visuals can be.

Overall, this work has managed to reach its objectives. It is not perfect but provides a great base to research even further into this field. In particular, a study of the most recurring errors made by the ensemble of models could provide great insights into what limits the performance of CBIR frameworks. It would also be relevant to try other combinations and identify other elements that could be added to the list of options made in this work. The implementation of this work can be found in the *cbir-tfe* repository on the Github page of the author of this thesis, at the following url: `https://github.com/AxelleSchyns/cbir-tfe`.

Bibliography

[An et al., 2021] An, X., Zhu, X., Xiao, Y., Wu, L., Zhang, M., Gao, Y., Qin, B., Zhang, D., and Fu, Y. (2021). Partial fc: Training 10 million identities on a single machine. In *Proceedings of IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), Montreal, BC, Canada*, pages 1445–1449.

[Bachman et al., 2019] Bachman, P., Hjelm, R. D., and Buchwalter, W. (2019). Learning representations by maximizing mutual information across views. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems, NIPS'19*, pages 15535–15545.

[Bahdanau et al., 2016] Bahdanau, D., Cho, K., and Bengio, Y. (2016). Neural machine translation by jointly learning to align and translate. *arXiv preprint: arXiv:1409.0473*.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision-ECCV 2006*, volume 3951, pages 404–417.

[Caron et al., 2021] Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. (2021). Emerging properties in self-supervised vision transformers. *arXiv preprint: arXiv:2104.14294*.

[Chen et al., 2022] Chen, C., Lu, M., Williamson, D., Chen, T., Schaumberg, A., and Mahmood, F. (2022). Fast and scalable search of whole-slide images via self-supervised deep learning. *Nature Biomedical Engineering*, 6:1–15.

[Chen et al., 2020] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1597–1607.

[Defraire, 2021] Defraire, S. (2020-2021). A distributed deep learning approach for histopathology image retrieval. Master's thesis, University of Liège.

[Dewan and Thepade, 2020] Dewan, J. and Thepade, S. (2020). Image retrieval using low level and local features contents: A comprehensive review. *Applied Computational Intelligence and Soft Computing*, 2020:1–20.

[Dosovitskiy et al., 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint: arXiv:2010.11929*.

[Dosovitskiy et al., 2016] Dosovitskiy, A., Fischer, P., Springenberg, J. T., Riedmiller, M., and Brox, T. (2016). Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:1734–1747.

[Falcon and Cho, 2020] Falcon, W. and Cho, K. (2020). A framework for contrastive self-supervised learning and designing a new approach. *arXiv preprint: arXiv:2009.00104*.

[Grill et al., 2020] Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. (2020). Bootstrap your own latent: A new approach to self-supervised learning. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NIPS 2020)*.

[Hadsell et al., 2006] Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint: arXiv:1512.03385*.

[Hegde et al., 2019] Hegde, N., Hipp, J., Liu, Y., Emmert-Buck, M., Reif, E., Smilkov, D., Terry, M., Cai, C., Amin, M., Mermel, C., Nelson, P., Peng, L., Corrado, G., and Stumpe, M. (2019). Similar image search for histopathology: Smily. *npj Digital Medicine*, 2:1–9.

[Huang et al., 2018] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2018). Densely connected convolutional networks. *arXiv preprint: arXiv:1608.06993*.

[Jain and Vailaya, 1996] Jain, A. K. and Vailaya, A. (1996). Image retrieval using color and shape. *Pattern Recognition*, 29(8):1233–1244.

[Johnson et al., 2017] Johnson, J., Douze, M., and Jégou, H. (2017). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

[Jégou et al., 2011] Jégou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128.

[Kalra et al., 2020] Kalra, S., Tizhoosh, H., Choi, C., Shah, S., Diamandis, P., Campbell, C. J., and Pantanowitz, L. (2020). Yottixel – an image search engine for large archives of histopathology whole slide images. *Medical Image Analysis*, 65:1–12.

[Kaya and Bilge, 2019] Kaya, M. and Bilge, H. (2019). Deep metric learning: A survey. *Symmetry*, 11:1066.

[Kingma and Welling, 2022] Kingma, D. P. and Welling, M. (2022). Auto-encoding variational bayes. *arXiv preprint: arXiv:1312.6114*.

[Kobs et al., 2021] Kobs, K., Steininger, M., Dulny, A., and Hotho, A. (2021). Do different deep metric learning losses lead to similar learned features? In *Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10624–10634, Los Alamitos, CA, USA. IEEE Computer Society.

[Krizhevsky et al., 2017] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

[Kumar et al., 2013] Kumar, A., Kim, J., Cai, W., Fulham, M., and Feng, D. D. F. (2013). Content-based medical image retrieval: A survey of applications to multidimensional and multimodality data. *Journal of Digital Imaging*, 26:1025–1039.

[Le-Khac et al., 2020] Le-Khac, P., Healy, G., and Smeaton, A. (2020). Contrastive representation learning: A framework and review. In *Proceedings of the 37th International Conference on Machine Learning*, volume 8, pages 193907–193934.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Louppe, 2022] Louppe, G. (2021-2022). Deep learning, info8010. ULiège.

[Lu et al., 2017] Lu, J., Hu, J., and Zhou, J. (2017). Deep metric learning for visual understanding: An overview of recent advances. *IEEE Signal Processing Magazine*, 34(6):76–84.

[Lu et al., 2015] Lu, J., Wang, G., Deng, W., Moulin, P., and Zhou, J. (2015). Multi-manifold deep metric learning for image set classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Maliki et al., 2019] Maliki, N. E., Silkan, H., and Maghri, M. E. (2019). Efficient indexing and similarity search using the geometric near-neighbor access tree (gnat) for face-images data. *Procedia Computer Science*, 148:600–609.

[Malkov and Yashunin, 2018] Malkov, Y. A. and Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836.

[Manjunath and Ma, 1996] Manjunath, B. and Ma, W. (1996). Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842.

[Minarno et al., 2021] Minarno, A. E., Ghufron, K. M., Sabrila, T. S., Husniah, L., and Sumadi, F. D. S. (2021). Cnn based autoencoder application in breast cancer image retrieval. In *Proceedings of the 2021 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pages 29–34.

[Mormont et al., 2021] Mormont, R., Geurts, P., and Maree, R. (2021). Multi-task pre-training of deep neural networks for digital pathology. *IEEE Journal of Biomedical and Health Informatics*, 25(2):412–421.

[Nguyen et al., 2020] Nguyen, B. X., Nguyen, B. D., Carneiro, G., Tjiputra, E., Tran, Q. D., and Do, T.-T. (2020). Deep metric learning meets deep clustering: An novel unsupervised approach for feature embedding. *arXiv preprint: arXiv:2009.04091*.

[Nguyen and Bai, 2010] Nguyen, H. and Bai, L. (2010). Cosine similarity metric learning for face verification. In *Proceedings of the Asian Conference on Computer Vision*, pages 709–720.

[Oztel et al., 2019] Oztel, I., Yolcu, G., and Oz, C. (2019). Performance comparison of transfer learning and training from scratch approaches for deep facial expression recognition. In *2019 4th International Conference on Computer Science and Engineering (UBMK)*, pages 1–6.

[Qian et al., 2019] Qian, Q., Shang, L., Sun, B., Hu, J., Li, H., and Jin, R. (2019). Softtriple loss: Deep metric learning without triplet sampling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6449– 6457.

[Radford et al., 2018] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training. *OpenAI*.

[Riasatian et al., 2021] Riasatian, A., Babaie, M., Maleki, D., Kalra, S., Valipour, M., Hemati, S., Zaveri, M., Safarpoor, A., Shafiei, S., Afshari, M., Rasoolijaberi, M., Sikaroudi, M., Adnan, M., Shah, S., Choi, C., Damaskinos, S., Campbell, C. J., Diamandis, P., Pantanowitz, L., Kashani, H., Ghodsi, A., and Tizhoosh, H. R. (2021). Fine-tuning and training of densenet for histopathology image representation using tcga diagnostic slides. *Medical Image Analysis*, 70:1–11.

[Rifai et al., 2011] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 833–840, Madison, WI, USA. Omnipress.

[Roth et al., 2020] Roth, K., Milbich, T., Sinha, S., Gupta, P., Ommer, B., and Cohen, J. P. (2020). Revisiting training strategies and generalization performance in deep metric learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119.

[Sabatelli et al., 2019] Sabatelli, M., Kestemont, M., Daelemans, W., and Geurts, P. (2019). Deep transfer learning for art classification problems. In Leal-Taixé, L. and Roth, S., editors, *Computer Vision – ECCV 2018 Workshops*, pages 631–646, Cham. Springer International Publishing.

[Siddiqui et al., 2017] Siddiqui, S., Salman, A., Malik, I., Shafait, F., Mian, A., Shortis, M., and Harvey, E. (2017). Automatic fish species classification in underwater videos: Exploiting pretrained deep neural network models to compensate for limited labelled data. *ICES Journal of Marine Science*, 75.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Singh and Srivastava, 2017] Singh, V. P. and Srivastava, R. (2017). Improved image retrieval using color-invariant moments. In *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*, pages 1–6.

[Sivic and Zisserman, 2003] Sivic and Zisserman (2003). Video google: a text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, pages 1470–1477 vol.2.

[Tan et al., 2019] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2815–2823.

[Tan and Le, 2020] Tan, M. and Le, Q. V. (2020). Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint: arXiv:1905.11946*.

[Teh et al., 2020] Teh, E. W., DeVries, T., and Taylor, G. W. (2020). Proxynca++: Revisiting and revitalizing proxy neighborhood component analysis. In *Proceedings of the 16th European Conference on Computer Vision – ECCV 2020*, pages 448–464.

[Tian et al., 2021] Tian, Y., Chen, X., and Ganguli, S. (2021). Understanding self-supervised learning dynamics without contrastive pairs. In *Proceedings of the 38th International Conference on Machine Learning*.

[Touvron et al., 2021] Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. (2021). Training data-efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning Research*, volume 139, pages 10347–10357.

[van den Oord et al., 2019] van den Oord, A., Li, Y., and Vinyals, O. (2019). Representation learning with contrastive predictive coding. *arXiv preprint: arXiv:1807.03748*.

[van der Maaten and Hinton, 2008] van der Maaten, L. and Hinton, G. (2008). Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems, NIPS 2017*, pages 5998–6008.

[Wang et al., 2014] Wang, J., song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., and Wu, Y. (2014). Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[Wang et al., 2023] Wang, X., Du, Y., Yang, S., Zhang, J., Wang, M., Zhang, J., Yang, W., Huang, J., and Han, X. (2023). Retccl: Clustering-guided contrastive learning for whole-slide image retrieval. *Medical Image Analysis*, 83:102645.

[Wiggers et al., 2019] Wiggers, K. L., Britto, A. S., Heutte, L., Koerich, A. L., and Oliveira, L. S. (2019). Image retrieval and pattern spotting using siamese neural network. In *Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

[Wu et al., 2017] Wu, C.-Y., Manmatha, R., Smola, A. J., and Krähenbühl, P. (2017). Sampling matters in deep embedding learning. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2859–2867.

[Wu et al., 2021] Wu, H., Xiao, B., Codella, N., Liu, M., Dai, X., Yuan, L., and Zhang, L. (2021). Cvt: Introducing convolutions to vision transformers. In *Proceedings on the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[Zhai and Wu, 2019] Zhai, A. and Wu, H.-Y. (2019). Classification is a strong baseline for deep metric learning. *arXiv preprint: arXiv:1811.12649*.

[Zheng et al., 2021] Zheng, W., Zhang, B., Lu, J., and Zhou, J. (2021). Deep relational metric learning. In *Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV)*.

[Zhong et al., 2021] Zhong, A., Li, X., Wu, D., Ren, H., Kim, K., Kim, Y., Buch, V., Neumark, N., Bizzo, B., Tak, W. Y., Park, S. Y., Lee, Y. R., Kang, M. K., Park, J. G., Kim, B. S., Chung, W. J., Guo, N., Dayan, I., Kalra, M. K., and Li, Q. (2021). Deep metric learning-based image retrieval system for chest radiograph and its clinical applications in covid-19. *Medical Image Analysis*, 70:101993.

Appendix

The following elements are presented in this Appendix:

- A. The Classes and projects in the histopathology dataset
- B. Demonstration of the loss formula for the third implementation of the AE
- C. Complete results for the model 1 and model 47
- D. Visualization of the data
- E. Retrieved images for several models and queries

Note that in addition of these elements, several annex files are available on MatheO: implementation.pdf, results_per_class.xlsx, model40_clusters_analysis.pdf and model41_clusters_analysis.pdf.

A. Classes and projects in the histopathology dataset

The table 7.1 shows all the projects composing the dataset. C1 is the original number of the class while C2 is the new number that were attributed to them for simplicity purpose.

Project	C1	C2
Camelyon16	0	0
	1	1
cells no aug	0	0
	1	1
patterns no aug	0	0
	1	1
glomeruli no aug	0	0
	1	1
iciar18 micro	113351562	0
	113351588	1
	113351608	2
	113351628	3
janowczyk1	1	0
	1	1
janowczky2	2	0
	1	1
janowczyk5	0	0
	1	1
janowczyk6	0	0
	1	1
janowczyk7	0	0
	1	1
	2	2
ulb anapth lba	4711	0
	4712	1
	4713	2
	4714	3
	4715	4
	4720	5
	68567	6
	485565	7
	672444	8
lbpstroma	113349434	0
	113349448	1

Project	C1	C2
ulg lbtd2 chimio necrose	36362022	0
	36362044	1
ulg lbtd lba	4762	0
	4763	1
	4764	2
	4765	3
	4766	4
	4767	5
	4768	6
	406558	7
ulg bonemarrow	0	0
	1	1
	2	2
	3	3
	4	4
	5	5
	6	6
	7	7
tupac mitosis	0	0
	1	1
warwicj crc	0	0
	1	1
umcm colorectal	01_TUMOR	0
	02_STROMA	1
	03_COMPLEX	2
	04_LYMPHO	3
	05_DEBRIS	4
	06_MUCOSA	5
	07_ADIPOSE	6
	08_EMPTY	7
mitos2014	0	0
	1	1
	2	2

Table 7.1: Division of the histopathology dataset
C1 = original class number, C2 = simplified class number

B. Demonstration of the loss formula for the third implementation of the AE

The function of interest is given by

$$h_j(x) = \sigma\left([w_3\sigma(w_2\sigma(w_1x))]_j\right) \tag{7.1}$$

where σ is the element-wise sigmoid activation function and w_i $(i = 1, 2, 3)$ are the weight matrices of the different layers of the network. x is the input vector while h is the output of the bottleneck layer. As a remainder, the layers have the following dimensions:

- Layer 1: input size = 784, output size = 64
- Layer 2: input size = 64, output size = 32
- Layer 3: input size = 32, output size = 16

In order to differentiate $h_j(x)$ with respect to x_i, let us write h_j more explicitly using matrix calculations and relying on the dimensions detailed above. First, the jth element inside the brackets in (7.1) is given by

$$[w_3\sigma(w_2\sigma(w_1x))]_j = \sum_{l=1}^{32}(w_3)_{jl}\sigma\left([w_2\sigma(w_1x)]_l\right) \tag{7.2}$$

and this element will be referred to as $(h_3)_j$ in the derivation below.

Similarly, the lth element inside the brackets in (7.2) can be written as

$$[w_2\sigma(w_1x)]_l = \sum_{m=1}^{64}(w_2)_{lm}\sigma\left([w_1x]_m\right) \tag{7.3}$$

and it will be denoted by $(h_2)_l$ in the computations.

Finally, the mth element in (7.3) is given by

$$[w_1x]_m = \sum_{k=1}^{784}(w_1)_{mk}x_k \tag{7.4}$$

and it will be denoted by $(h_1)_m$.

Putting (7.2), (7.3) and (7.4) together in (7.1) yields

$$h_j(x) = \sigma\left(\sum_{l=1}^{32}(w_3)_{jl}\ \sigma\left(\sum_{m=1}^{64}(w_2)_{lm}\ \sigma\left(\sum_{k=1}^{784}(w_1)_{mk}x_k\right)\right)\right)$$

and following the same notations as before, this whole expression will be referred to as $(h_4)_j$.

For the differentation step, it might be useful to add the intermediate notations to the full expression of $(h_4)_j$:

$$(h_4)_j = \sigma\left(\overbrace{\sum_{l=1}^{32}(w_3)_{jl}\ \sigma\left(\underbrace{\sum_{m=1}^{64}(w_2)_{lm}\ \sigma\left(\overbrace{\sum_{k=1}^{784}(w_1)_{mk}x_k}^{(h_1)_m}\right)}_{(h_2)_l}\right)}^{(h_3)_j}\right)$$

By the chain-rule of differentiation, we get

$$\begin{aligned}
\frac{\partial(h_4)_j}{\partial x_i} &= \sigma'_{|(h_3)_j}\ \frac{\partial(h_3)_j}{\partial x_i} \\
&= \sigma'_{|(h_3)_j}\left(\sum_{l=1}^{32}(w_3)_{jl}\ \sigma'_{|(h_2)_l}\ \frac{\partial(h_2)_l}{\partial x_i}\right) \\
&= \sigma'_{|(h_3)_j}\left(\sum_{l=1}^{32}(w_3)_{jl}\ \sigma'_{|(h_2)_l}\ \left(\sum_{m=1}^{64}(w_2)_{lm}\ \sigma'_{|(h_1)_m}\frac{\partial(h_1)_m}{\partial x_i}\right)\right) \\
&= \sigma'_{|(h_3)_j}\left(\sum_{l=1}^{32}(w_3)_{jl}\ \sigma'_{|(h_2)_l}\ \left(\sum_{m=1}^{64}(w_2)_{lm}\ \sigma'_{|(h_1)_m}(w_1)_{mi}\right)\right) \qquad (7.5)
\end{aligned}$$

Now, due to its analytical expression, we know that

$$\sigma'(h) = \sigma(h)(1-\sigma(h))$$

Therefore,

$$\begin{aligned}
\sigma'_{|(h_3)_j} = \sigma'((h_3)_j) &= \sigma\left((h_3)_j\right)\left(1-\sigma\left((h_3)_j\right)\right) \\
&= (h_4)_j(1-(h_4)_j) \text{ by definition of } h_3 \text{ and } h_4 \\
&= [\mathrm{diag}(h_4(1-h_4))]_{jj}
\end{aligned}$$

Using the same type of expressions for the other derivatives of σ appearing in (7.5) and by the properties of matrix products, we end up with

$$\frac{\partial(h_4)_j}{\partial x_i} = \left(\mathrm{diag}(h_4(1-h_4))\ w_3\ \mathrm{diag}(h_3(1-h_3))\ w_2\ \mathrm{diag}(h_2(1-h_2))\ w_1\right)_{ji}$$

C. Complete results for the model 1 and model 47

Quantitative results

Metric	Model 1 with the different protocols				Model 47 with the different protocols			
	Random	All	Weighted	Remove	Random	All	Weighted	Remove
Top-1	72.06 ± 0.9	75.89	72.46	59.69	39.95 ± 0.79	68.08	41.24	44.63
Top-5	87.7 ± 0.9	92.93	86.57	85.22	64.06 ± 0.87	85.87	63.17	72.71
Top-1 proj	95.35 ± 0.4	94.8	95.42	89.29	74.3 ± 0.65	86.45	77.52	73.92
Top-5 proj	97.92 ± 0.28	97.97	97.93	94.69	86.6 ± 0.84	92.37	88.08	84.92
Top-1 sim	95.36 ± 0.4	94.82	95.44	89.33	76.24 ± 0.71	89.92	79.13	80.93
Top-5 sim	97.92 ± 0.28	97.99	97.93	94.73	88.11 ± 0.82	94.75	89.31	89.72
Maj	71.72 ± 0.9	78.03	72.15	61.81	33.14 ± 0.98	68.8	32.5	43.37
Maj proj	95.14 ± 0.4	94.64	94.91	89.45	71.87 ± 0.83	86.17	72.75	73.16
Maj sim	95.14 ± 0.4	94.66	94.914	89.52	74.05 ± 0.93	90.09	74.1	81.06

Qualitative results

For each class, one image is selected at random as query. The top_1 retrieved image for each model and class is presented, with its class and distance to the query.

Figure 7.1: Query: Cam_0; 1: Cam_0, d = 0.04 ; 47: Cam_0, d = 4.92

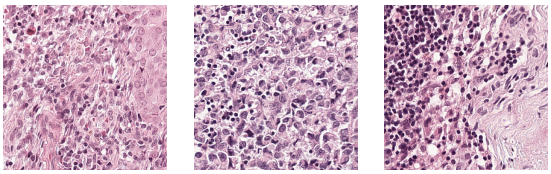

Figure 7.2: Query: Cam_1; 1: Cam_1, d = 0.06 ; 47: Cam_0, d = 19.84

Figure 7.3: Query: Cell_0; 1: Cell_0, d = 0.13 ; 47: Cell_0, d = 1.5

Figure 7.4: Query: Cells_1; 1: Cells_0, d = 0.25 ; 47: ulg_lbtd_0, d = 1.95

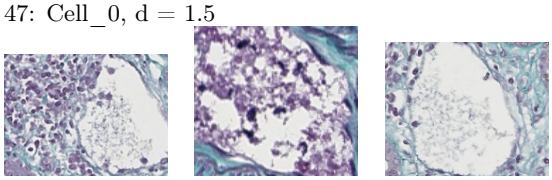

Figure 7.5: Query: Glom_0; 1: Glom_0, d = 0.12 ; 47: Glom_0, d = 63.98

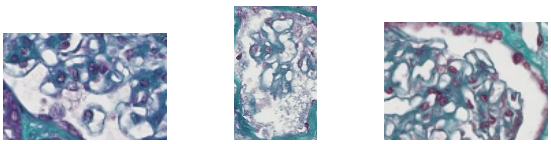

Figure 7.6: Query: Glom_1; 1: Glom_1, d = 0.07 ; 47: Glom_1, d = 16.76

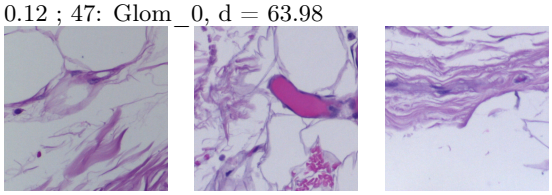

Figure 7.7: Query: iciar_0; 1: iciar_0, d = 0.06 ; 47: iciar_0, d = 31.45

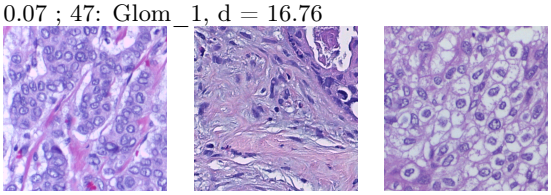

Figure 7.8: Query: iciar_1; 1: iciar_2, d = 0.18 ; 47: iciar_2, d = 41.36

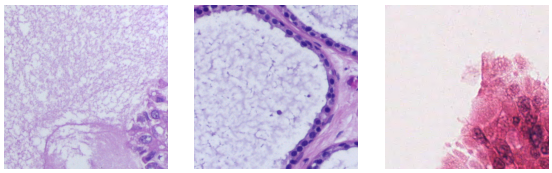

Figure 7.9: Query: iciar_2; 1: iciar_3, d = 0.2 ; 47: mitos_2, d = 48.49

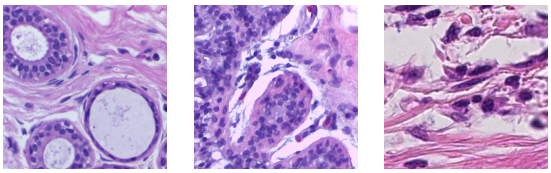

Figure 7.10: Query: iciar_3; 1: iciar_0, d = 0.15 ; 47: tupac_0, d = 24.85

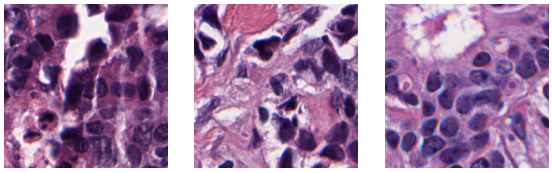

Figure 7.11: Query: jano1_0; 1: jano1_1, d = 0.09 ; 47: jano1_1, d = 16.97

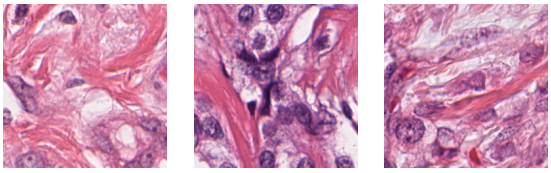

Figure 7.12: Query: jano1_1; 1: jano1_0, d = 0.11 ; 47: tupac_0, d = 9.2

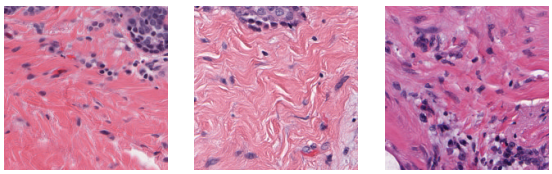

Figure 7.13: Query: jano2_0; 1: jano2_0, d = 0.05 ; 47: jano2_0, d = 21.98

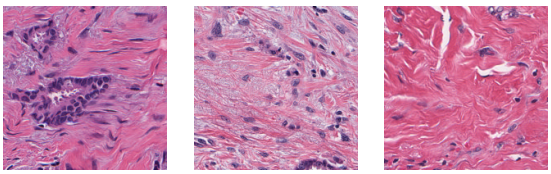

Figure 7.14: Query: jano2_1; 1: jano2_0, d = 0.08 ; 47: jano2_0, d = 8.92

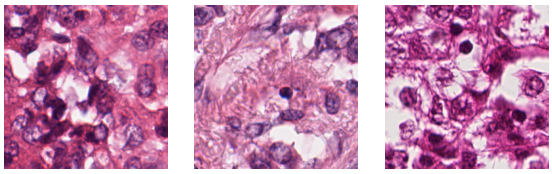

Figure 7.15: Query: jano5_0; 1: tupac_0, d = 0.01 ; 47: mitos_2, d = 17.18

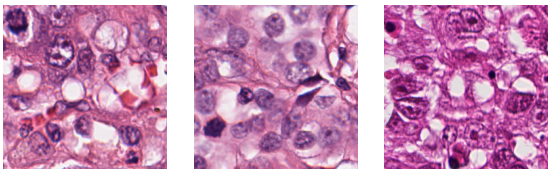

Figure 7.16: Query: jano5_1; 1: jano5_1, d = 0.01 ; 47: mitos_0, d = 5.04

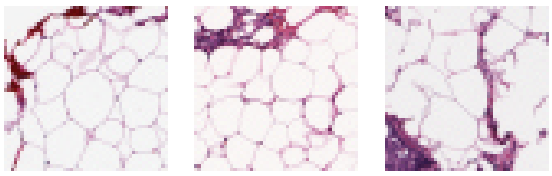

Figure 7.17: Query: jano6_0; 1: jano6_0, d = 0.02 ; 47: jano6_0, d = 9.86

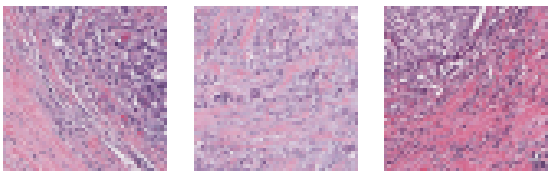

Figure 7.18: Query: jano6_1; 1: jano6_1, d = 0.005 ; 47: jano6_1, d = 5

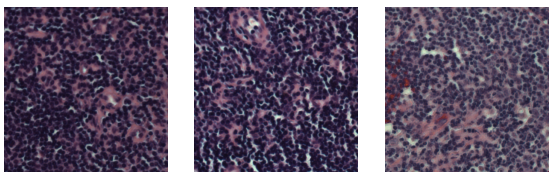

Figure 7.19: Query: jano7_0; 1: jano7_0, d = 0.03 ; 47: jano7_1, d = 23.64

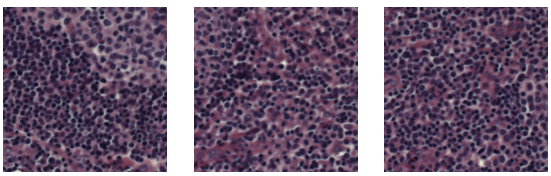

Figure 7.20: Query: jano7_1; 1: jano7_1, d = 0.04 ; 47: jano7_1, d = 6.98

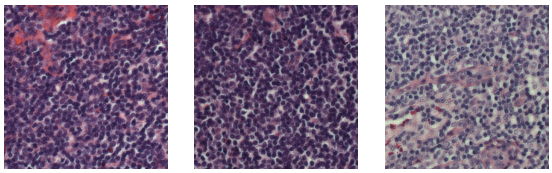

Figure 7.21: Query: jano7_2; 1: jano7_2, d = 0.02 ; 47: jano7_2, d = 12.04

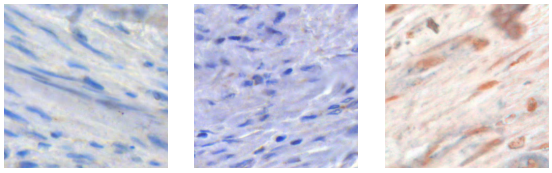

Figure 7.23: Query: lbps_1; 1: lbps_1, d = 0.02 ; 47: lbps_1, d = 16.05

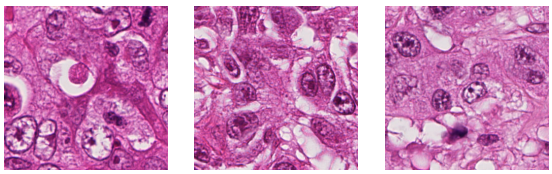

Figure 7.25: Query: mitos_1; 1: mitos_2, d = 0.006 ; 47: mitos_0, d = 5.11

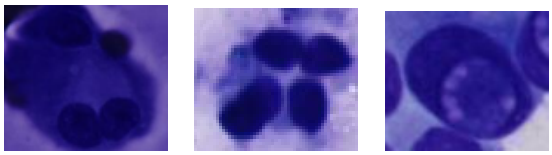

Figure 7.27: Query: patt_0; 1: patt_0, d = 0.3 ; 47: cells_0, d = 7.07

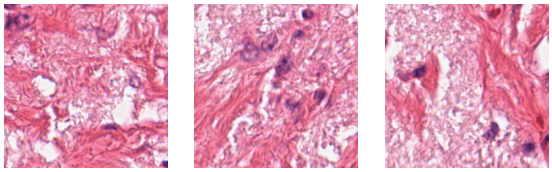

Figure 7.29: Query: tupac_0; 1: tupac_0, d = 0.007 ; 47: tupac_0, d = 12.89

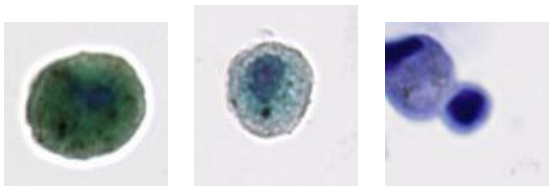

Figure 7.31: Query: ulb_0; 1: ulb_0, d = 0.1 ; 47: ulb_2, d = 8.05

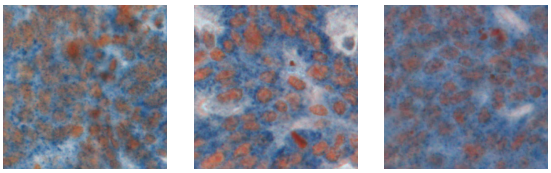

Figure 7.22: Query: lbps_0; 1: lbps_0, d = 0.07 ; 47: lbps_0, d = 3.6

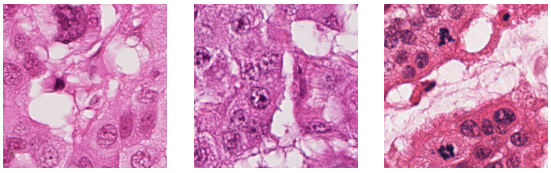

Figure 7.24: Query: mitos_0; 1: mitos_2, d = 0.01 ; 47: mitos_1, d = 8.36

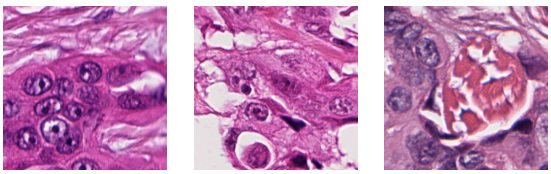

Figure 7.26: Query: mitos_2; 1: mitos_2, d = 0.08 ; 47: jano1_1, d = 15.97

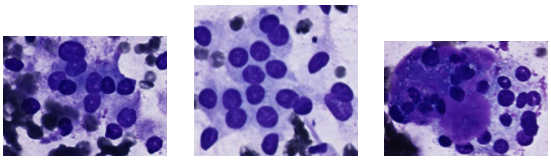

Figure 7.28: Query: patt_1; 1: patt_0, d = 0.12 ; 47: patt_0, d = 20.71

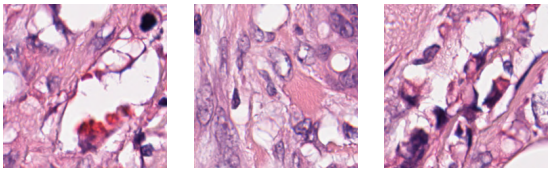

Figure 7.30: Query: tupac_1; 1: tupac_0, d = 0.01 ; 47: tupac_0, d = 14.13

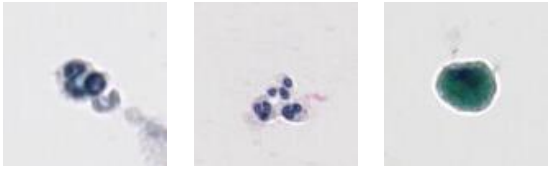

Figure 7.32: Query: ulb_1; 1: ulb_1, d = 0.02 ; 47: ulb_0, d = 5.99

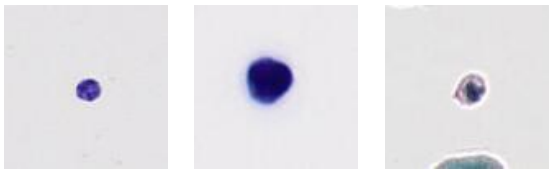

Figure 7.33: Query: ulb_2; 1: ulb_2, d = 0.24 ; 47: ulb_1, d = 7.08

Figure 7.34: Query: ulb_3; 1: ulb_3, d = 0.05 ; 47: ulb_3, d = 16.88

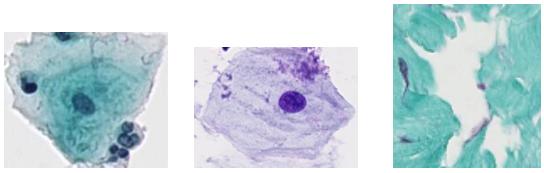

Figure 7.35: Query: ulb_5; 1: ulb_5, d = 0.15 ; 47: glom_0, d = 27.05

Figure 7.36: Query: ulb_6; 1: ulb_3, d = 0.29 ; 47: jano6_0, d = 19.92

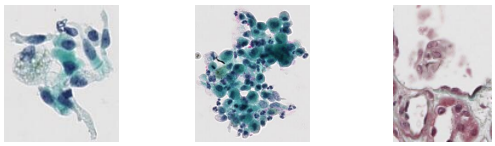

Figure 7.37: Query: ulb_7; 1: ulb_7, d = 0.37 ; 47: glom_0, d = 32.44

Figure 7.38: Query: ulb_8; 1: ulb_7, d = 0.34 ; 47: ulb_0, d = 6.78

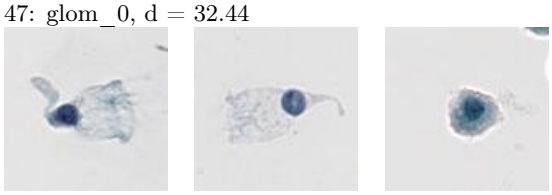

Figure 7.39: Query: ulb_9; 1: ulb_9, d = 0.08 ; 47: ulb_0, d = 6.8

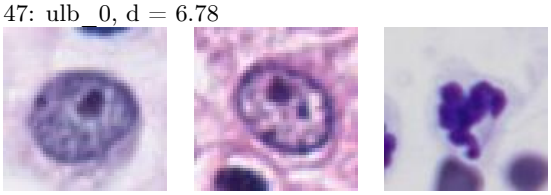

Figure 7.40: Query: bone_0; 1: bone_0, d = 0.25 ; 47: ulg_lbtd_2, d = 5.85

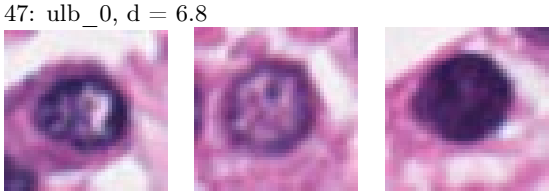

Figure 7.41: Query: bone_1; 1: bone_1, d = 0.1 ; 47: bone_5, d = 1.7

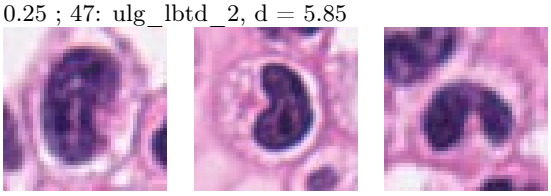

Figure 7.42: Query: bone_2; 1: bone_2, d = 0.09 ; 47: bone_3, d = 2.13

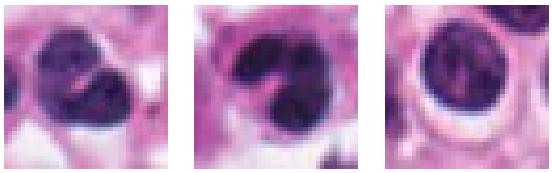

Figure 7.43: Query: bone_3; 1: bone_4, d = 0.13 ; 47: bone_5, d = 1.11

Figure 7.44: Query: bone_4; 1: bone_4, d = 0.16 ; 47: bone_3, d = 2.84

Figure 7.45: Query: bone_5; 1: bone_5, d = 0.15 ; 47: cells_0, d = 1.04

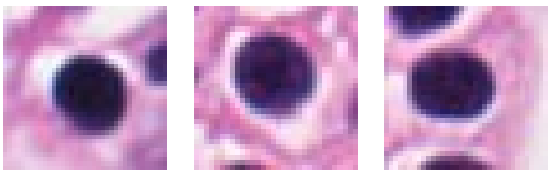

Figure 7.46: Query: bone_6; 1: bone_6, d = 0.19 ; 47: bone_6, d = 2.31

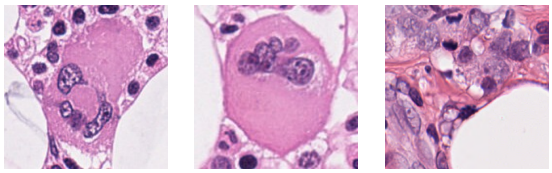

Figure 7.47: Query: bone_7; 1: bone_7, d = 0.13 ; 47: jano5_1, d = 38.16

Figure 7.48: Query: ulg_lbtd_0; 1: ulg_lbtd_0, d = 0.38 ; 47: ulg_lbtd_1, d = 2.14

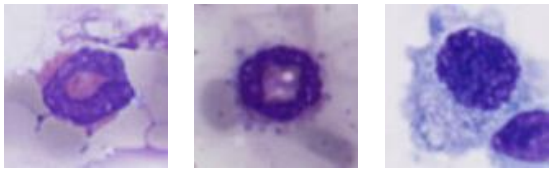

Figure 7.49: Query: ulg_lbtd_1; 1: ulg_lbtd_1, d = 0.1 ; 47: ulg_lbtd_0, d = 4.84

Figure 7.50: Query: ulg_lbtd_2; 1: ulg_lbtd_2, d = 0.04 ; 47: ulg_lbtd_2, d = 2.05

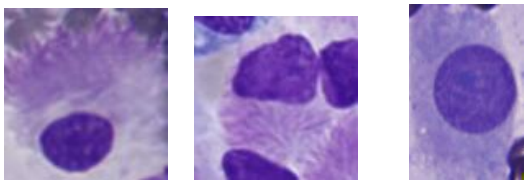

Figure 7.51: Query: ulg_lbtd_3; 1: ulg_lbtd_3, d = 0.33 ; 47: cells_0, d = 4.33

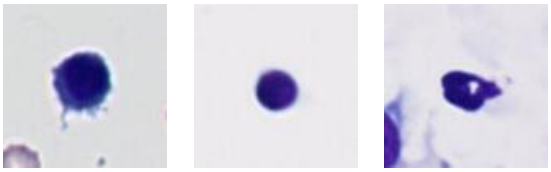

Figure 7.52: Query: ulg_lbtd_4; 1: ulg_ana_2, d = 0.16 ; 47: ulg_lbtd_2, d = 3.69

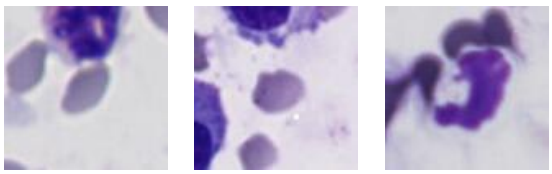

Figure 7.53: Query: ulg_lbtd_5; 1: ulg_lbtd_5, d = 0.33 ; 47: ulg_lbtd_2, d = 2.68

Figure 7.54: Query: ulg_lbtd_6; 1: ulg_lbtd_5, d = 0.16 ; 47: ulg_lbtd_6, d = 7.48

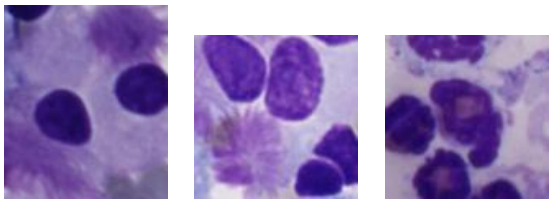

Figure 7.55: Query: ulg_lbtd_7; 1: ulg_lbtd_3, d = 0.21 ; 47: ulg_lbtd_1, d = 8.93

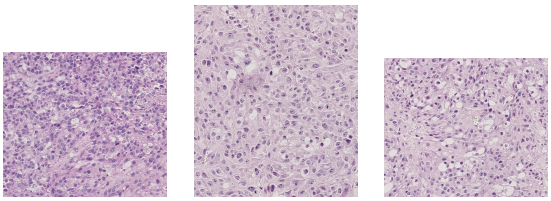

Figure 7.56: Query: ulg_lbtd2_0; 1: ulg_lbtd2_0, d = 0.11 ; 47: ulg_lbtd2_0, d = 16.11

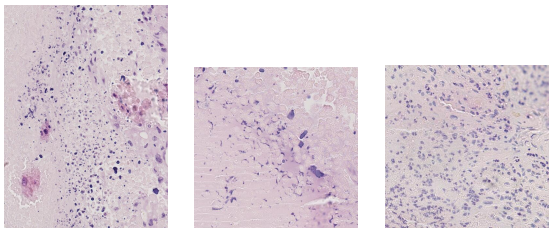

Figure 7.57: Query: ulg_lbtd2_1; 1: ulg_lbtd2_1, d = 0.07 ; 47: ulg_lbtd2_1, d = 32

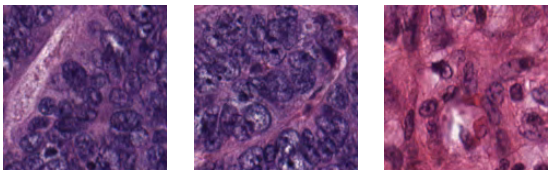

Figure 7.58: Query: umcm_0; 1: umcm_0, d = 0.09 ; 47: umcm_1, d = 16.48

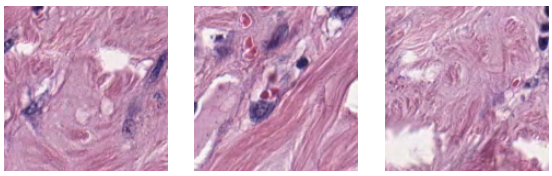

Figure 7.59: Query: umcm_1; 1: umcm_1, d = 0.38 ; 47: umcm_1, d = 6.77

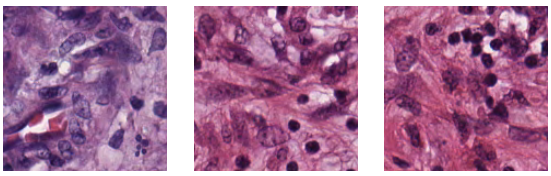

Figure 7.60: Query: umcm_2; 1: umcm_2, d = 0.2 ; 47: umcm_2, d = 14.07

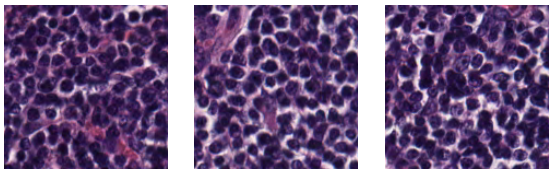

Figure 7.61: Query: umcm_3; 1: umcm_3, d = 0.006 ; 47: umcm_3, d = 6.69

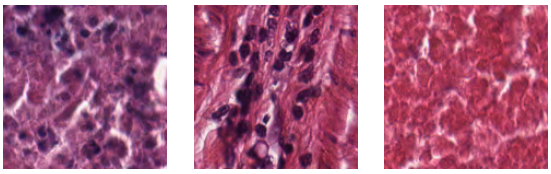

Figure 7.62: Query: umcm_4; 1: umcm_2, d = 0.15 ; 47: umcm_4, d = 6.58

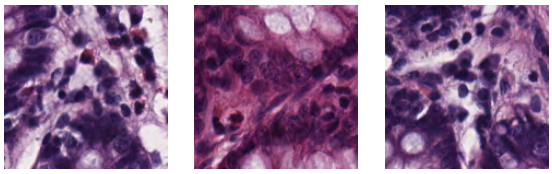

Figure 7.63: Query: umcm_5; 1: umcm_5, d = 0.06 ; 47: umcm_5, d =5.5

Figure 7.64: Query: umcm_6; 1: umcm_6, d = 0.11 ; 47: umcm_4, d = 16.97

Figure 7.65: Query: umcm_7; 1: umcm_7, d = 0.06 ; 47: umcm_7, d = 1.98

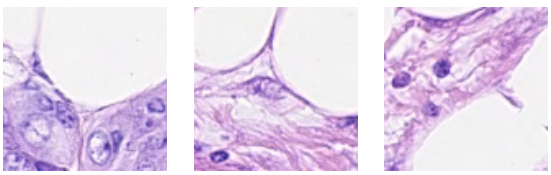

Figure 7.66: Query: warw_0; 1: warw_1, d = 0.05 ; 47: warw_1, d = 24.96

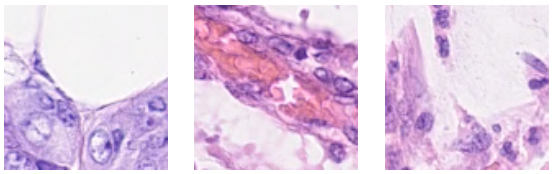

Figure 7.67: Query: warw_1; 1: warw_0, d = 0.14 ; 47: warw_1, d = 14.85

D. Visualization of the data

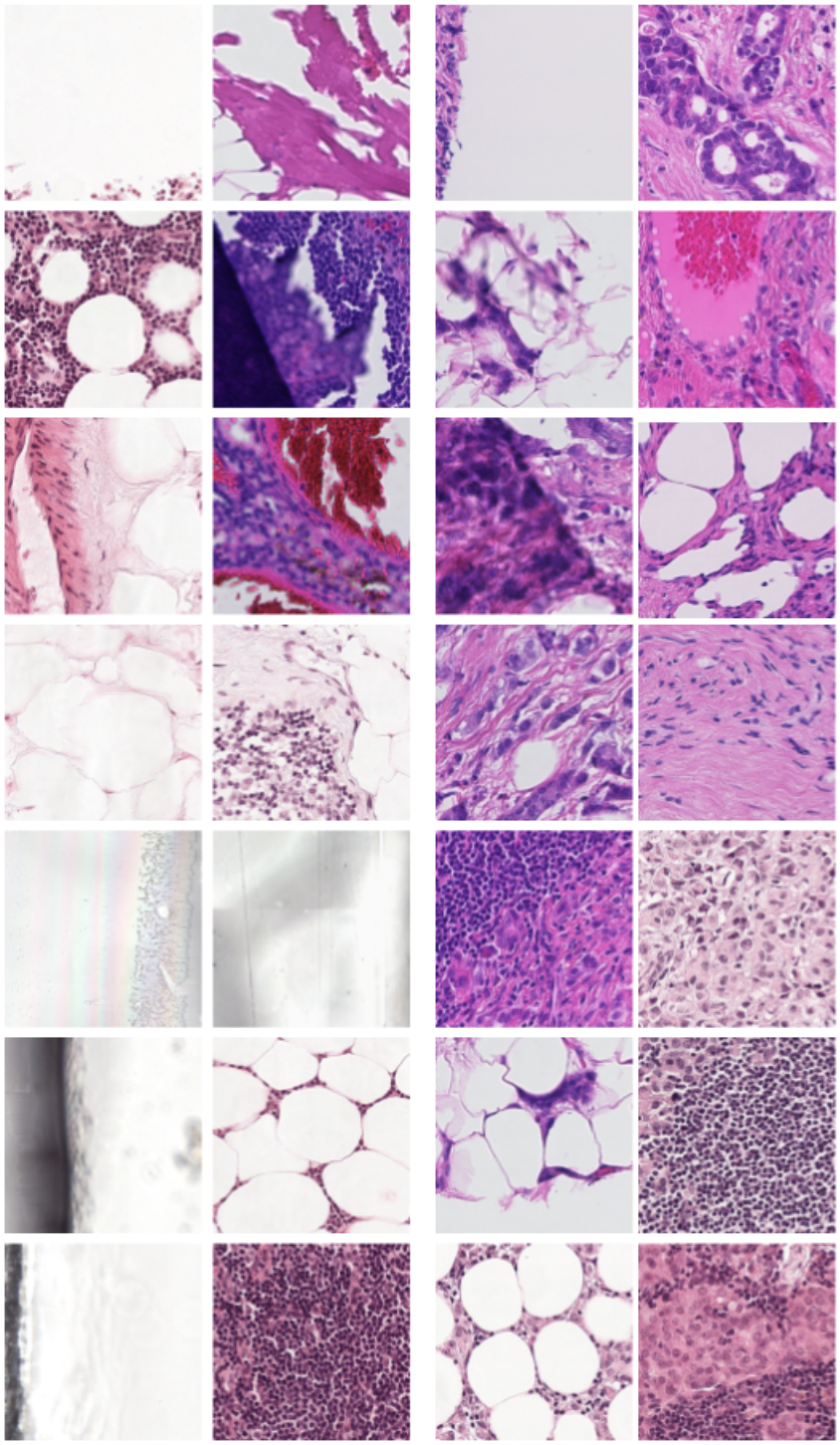

Figure 7.68: Camelyon16: class 0 (left) and 1 (right)

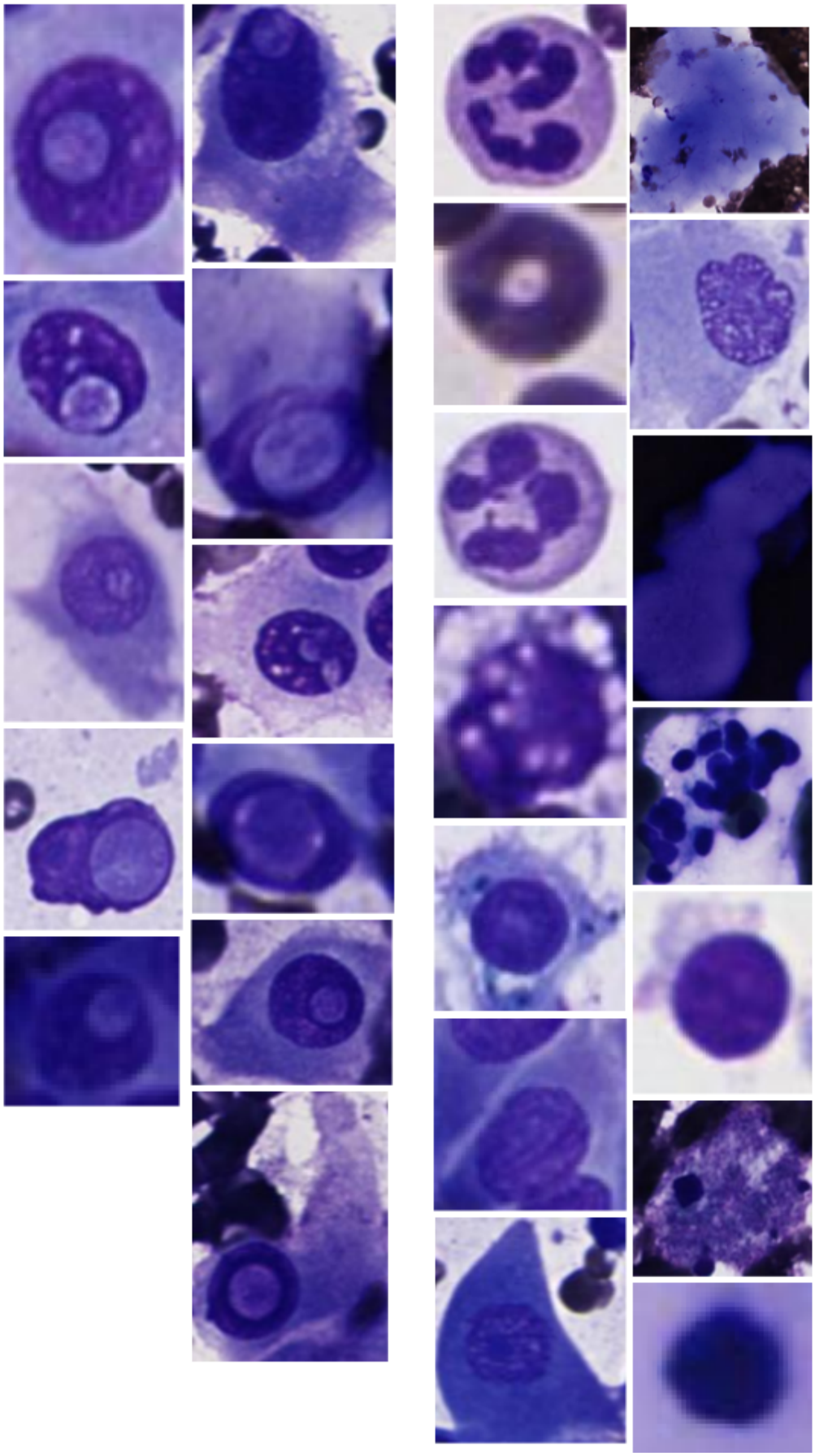

Figure 7.69: Cells no aug: class 0 (left) and 1 (right)

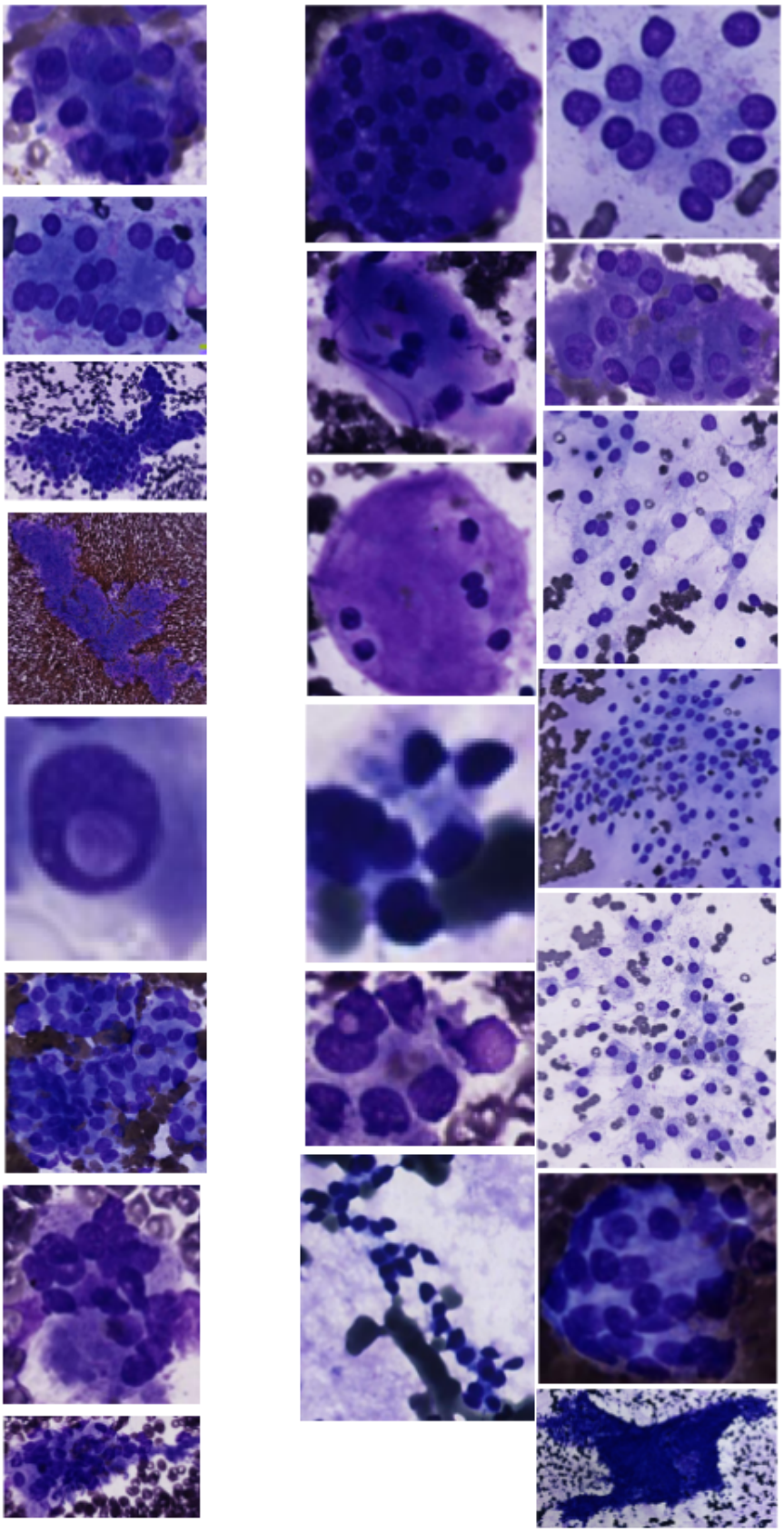

Figure 7.70: Patterns no aug: class 0 (left) and 1 (right)

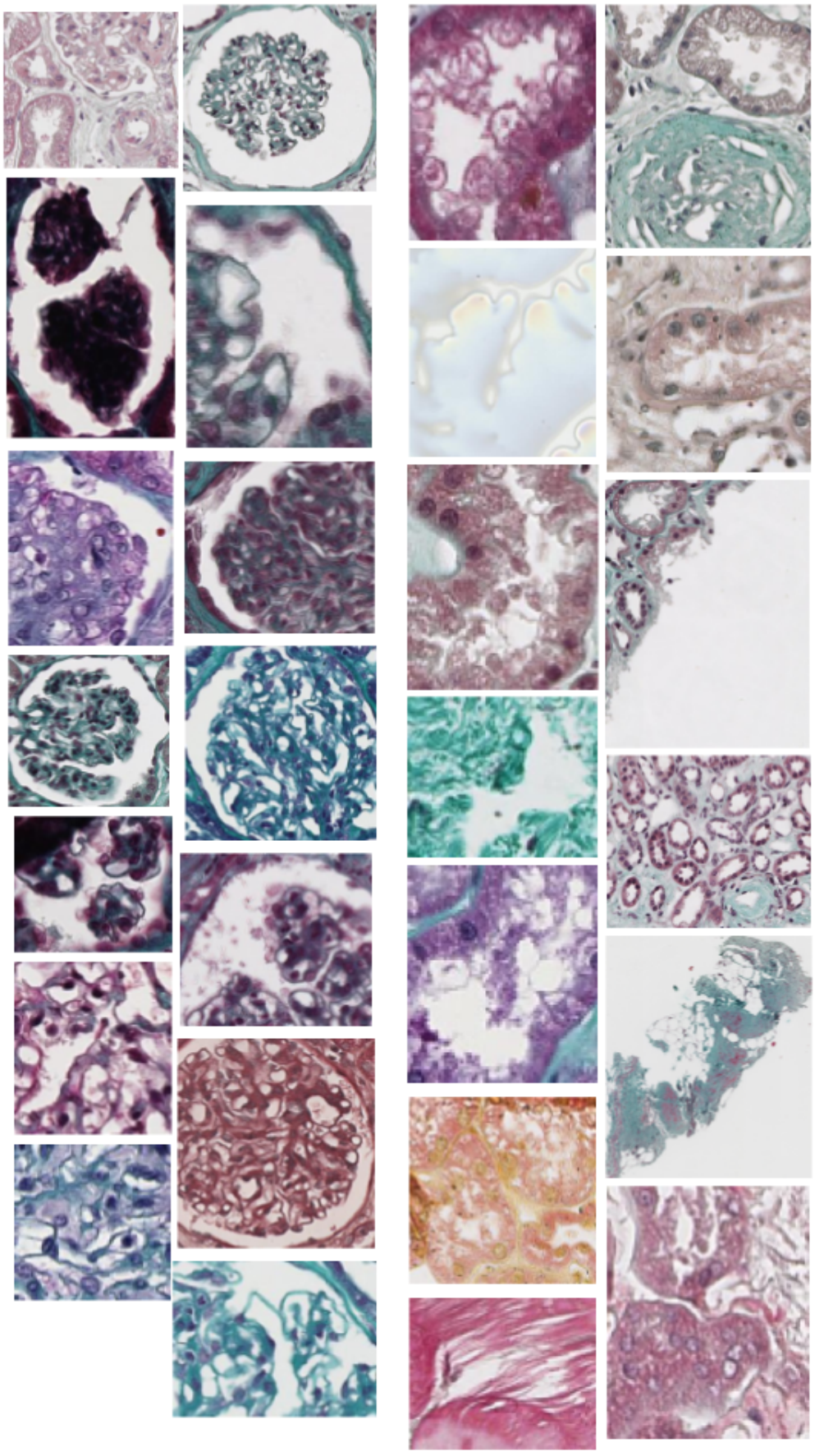

Figure 7.71: Glomeruli: class 0 (left) and 1 (right)

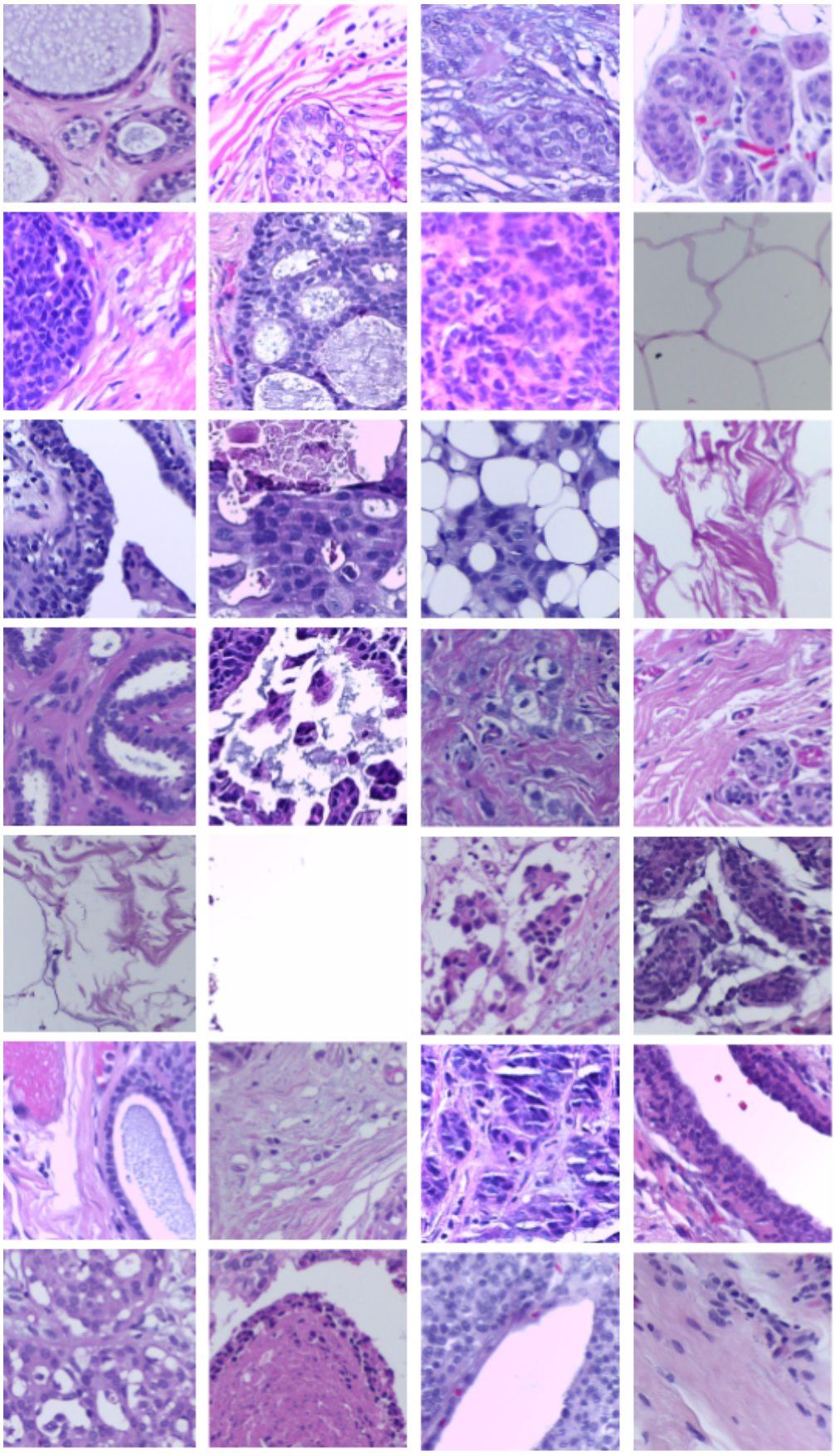

Figure 7.72: Iciar18 micro: classes in order from left to right (columns)

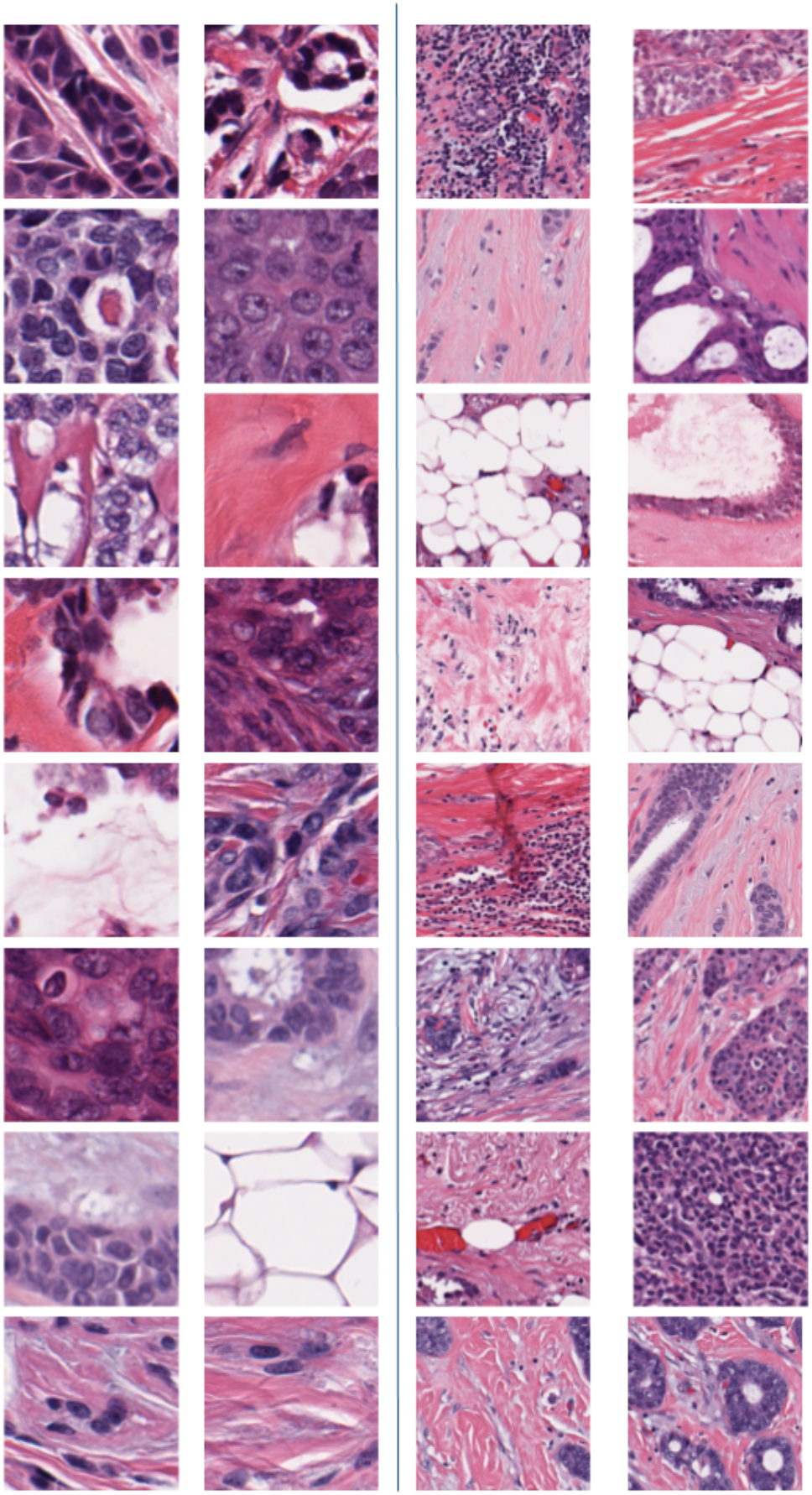

Figure 7.73: Janowczyk1: class 0 (1^{st}) and 1 (2^{nd}) - janowczyk2: class 0 (3^{rd}) and 1 (4^{th})

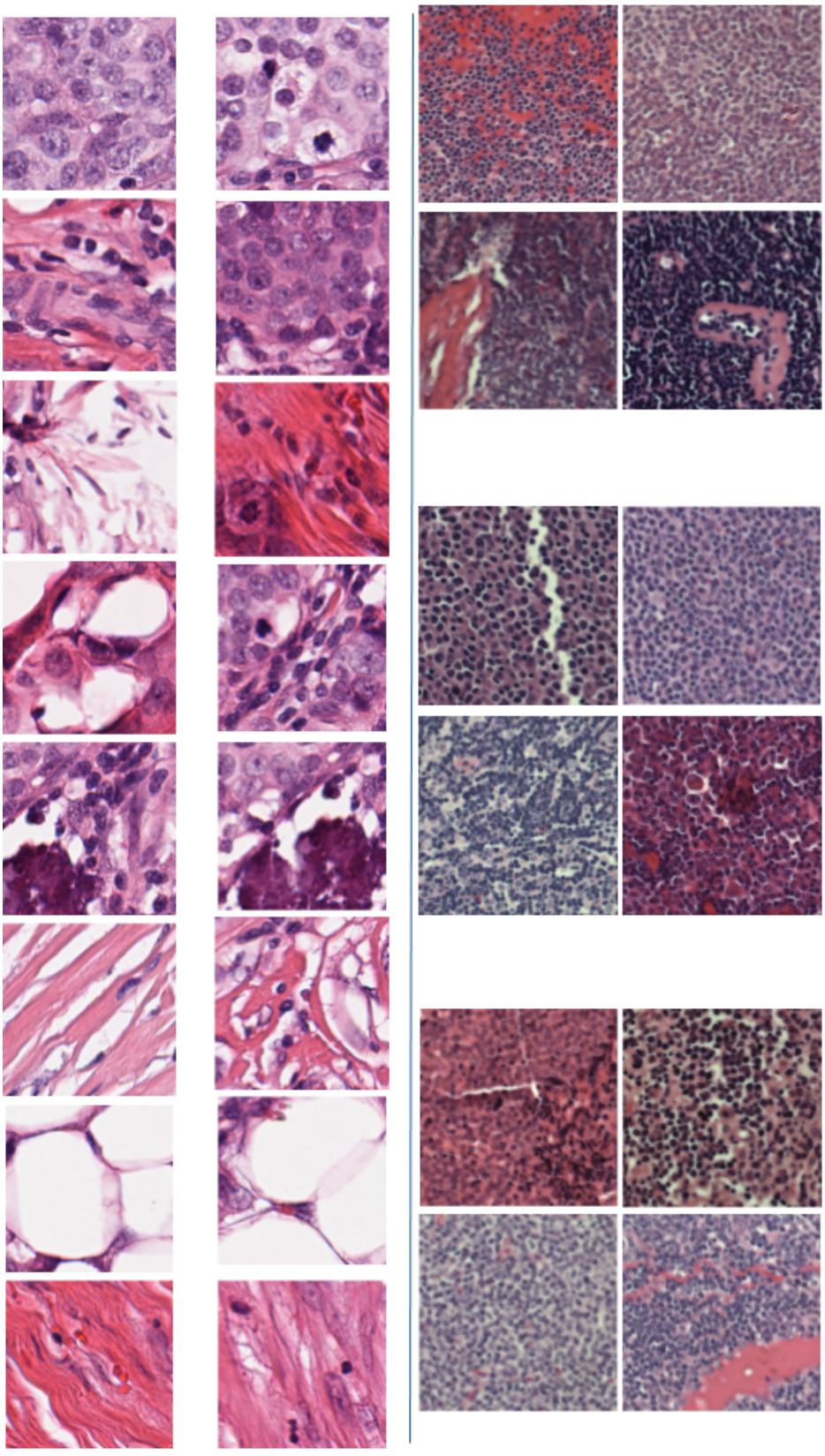

Figure 7.74: Janowczyk5: class 0 (1^{st}) and 1 (2^{nd}) - janowczyk7: class 0 (3^{rd} - top) and 1 (3^{rd} - middle) and 2 (3^{rd} - bottom)

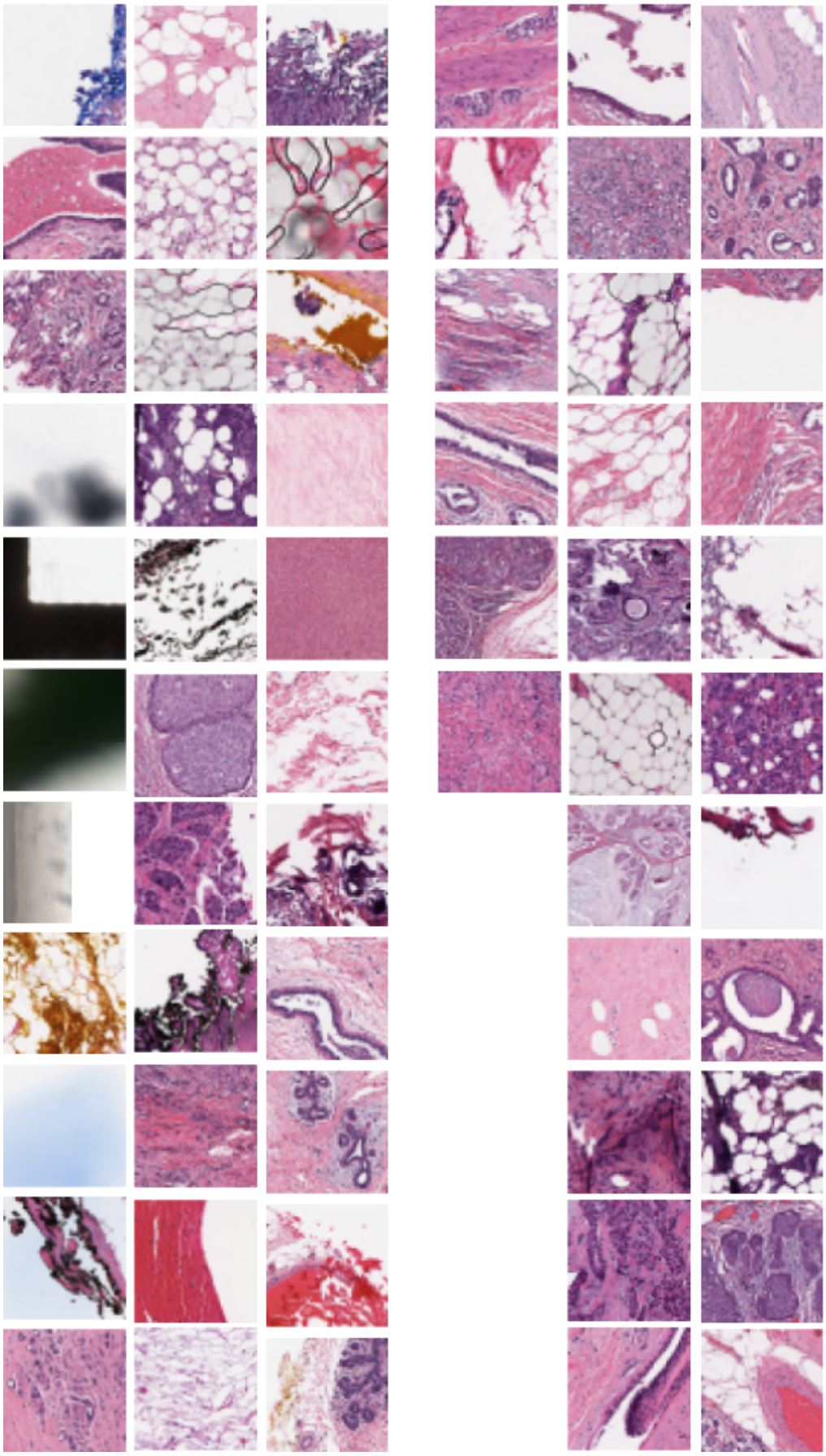

Figure 7.75: Janowczyk6: class 0 (left) and 1 (right)

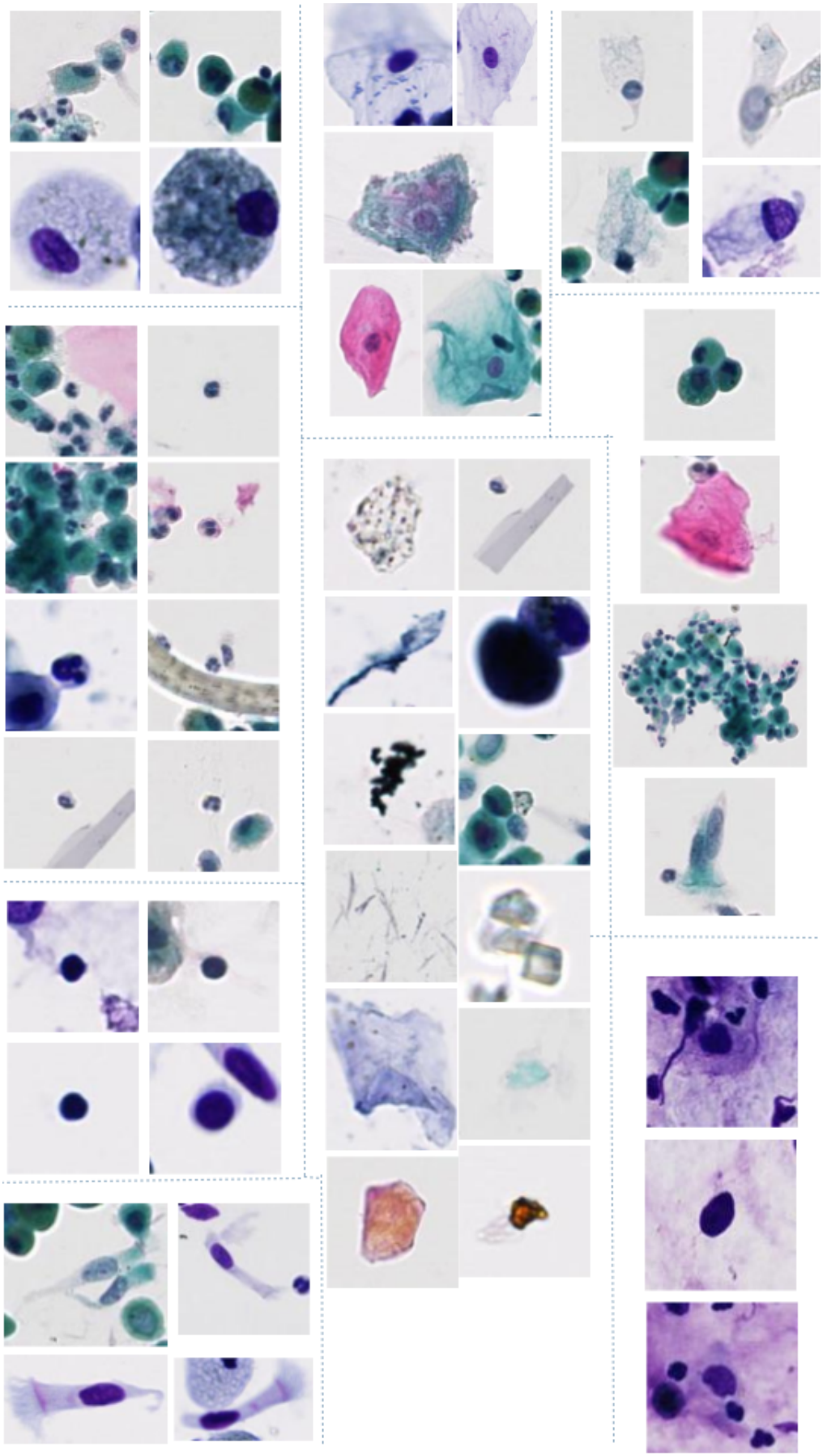

Figure 7.76: ulb anapath: classes in order from top to bottom then left to right

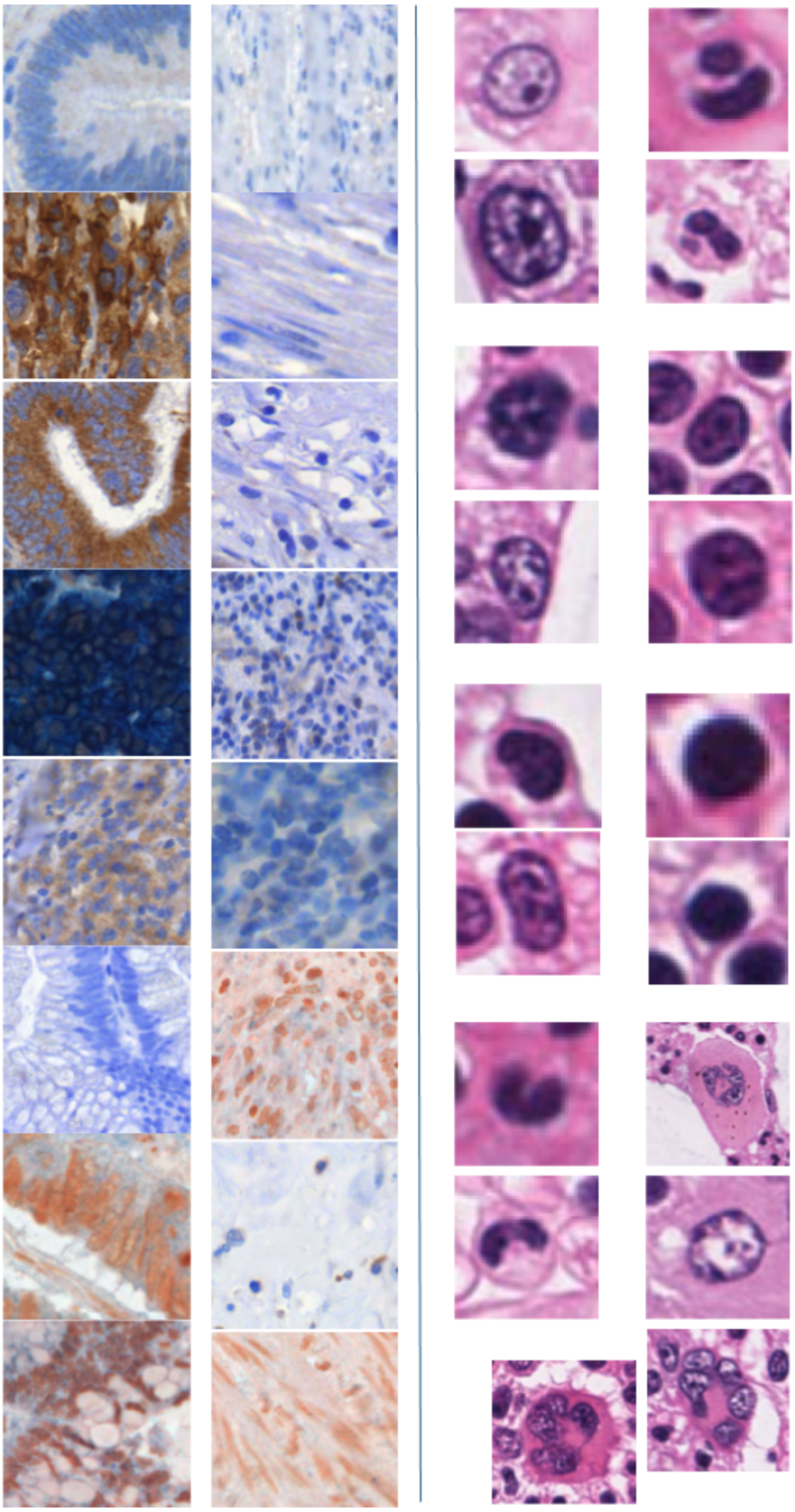

Figure 7.77: lbpstroma: class 113349434 (1^{st}) and 113349448 (2^{nd}) - ulg bonemarrow: classes in order from left to right then top to bottom

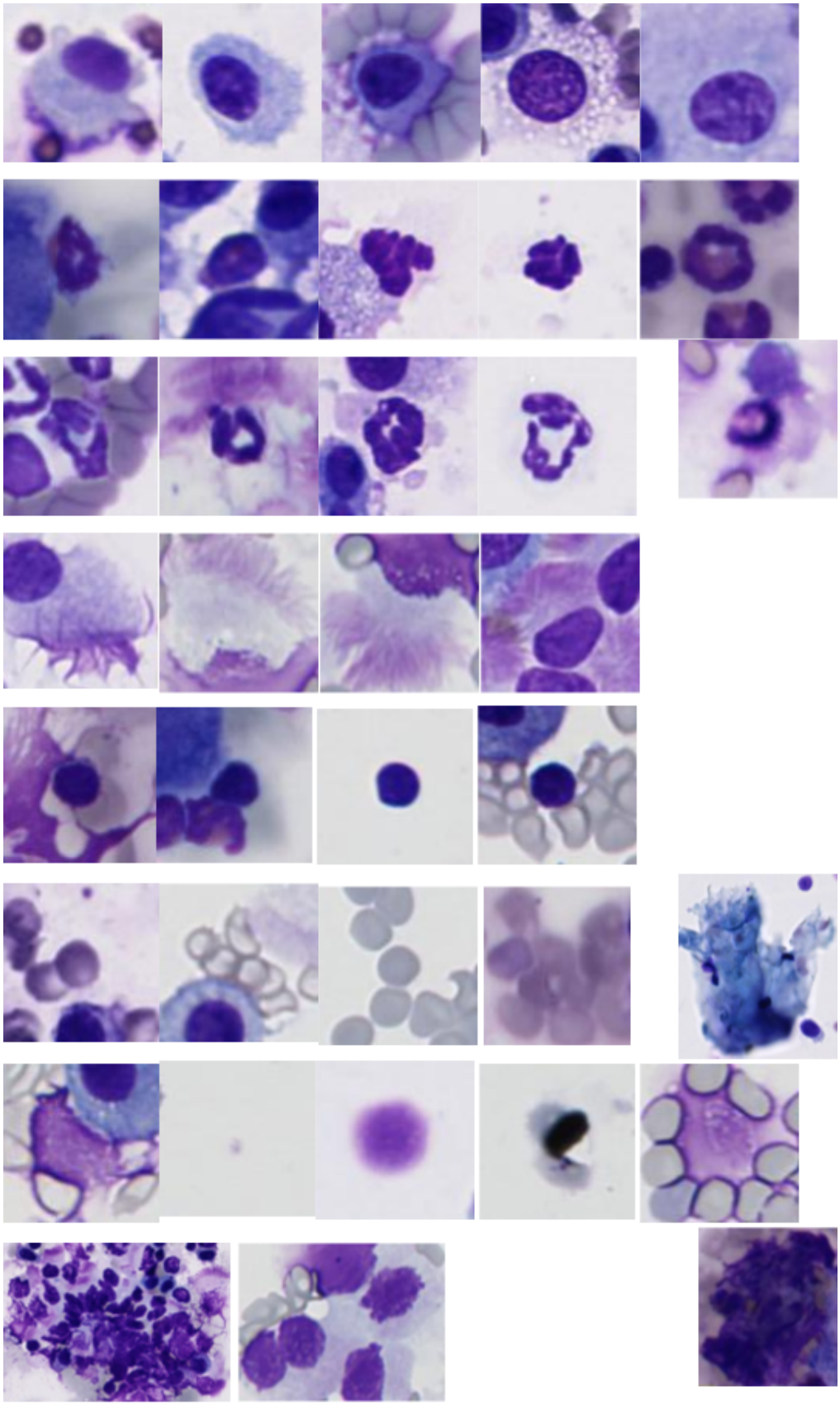

Figure 7.78: ulg lbtd lba: classes in order from top to bottom

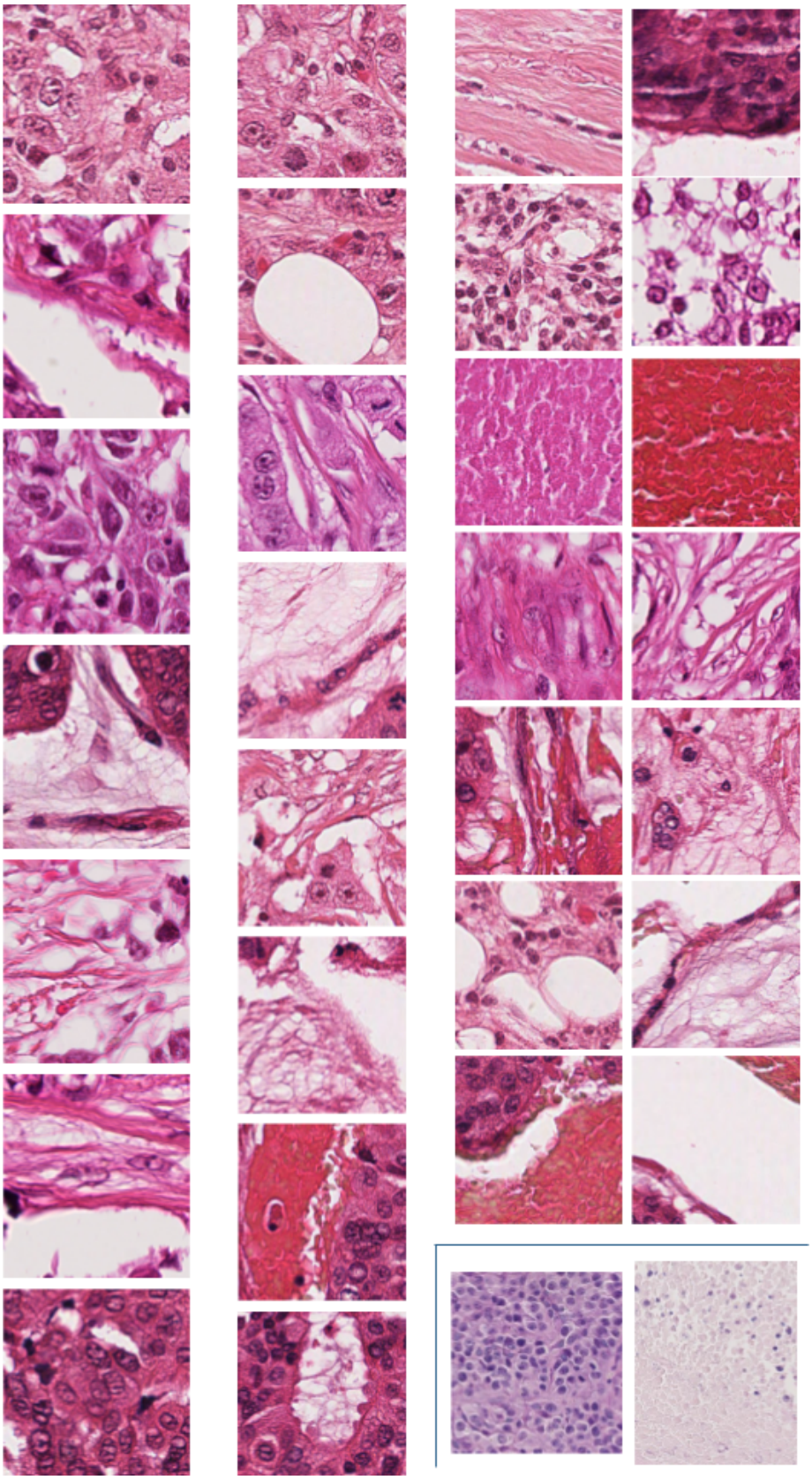

Figure 7.79: Mitos2014: class 0 (1^{st}), 1 (2^{nd}) and 2 (3^{rd}) - ulg lbtd2 chimio necrose: bottom right

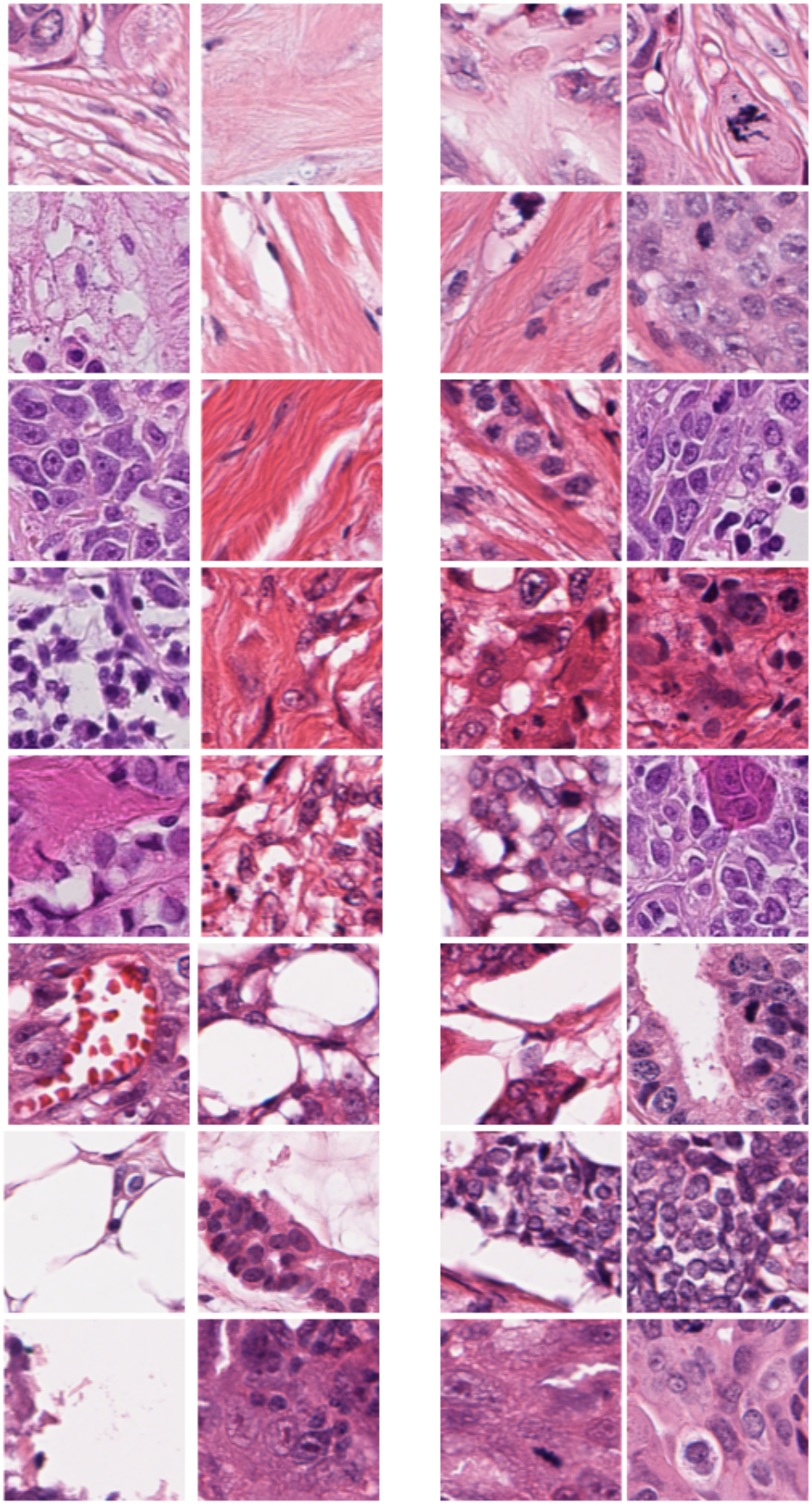

Figure 7.80: Tupac mitosis: O (left) and 1 (right)

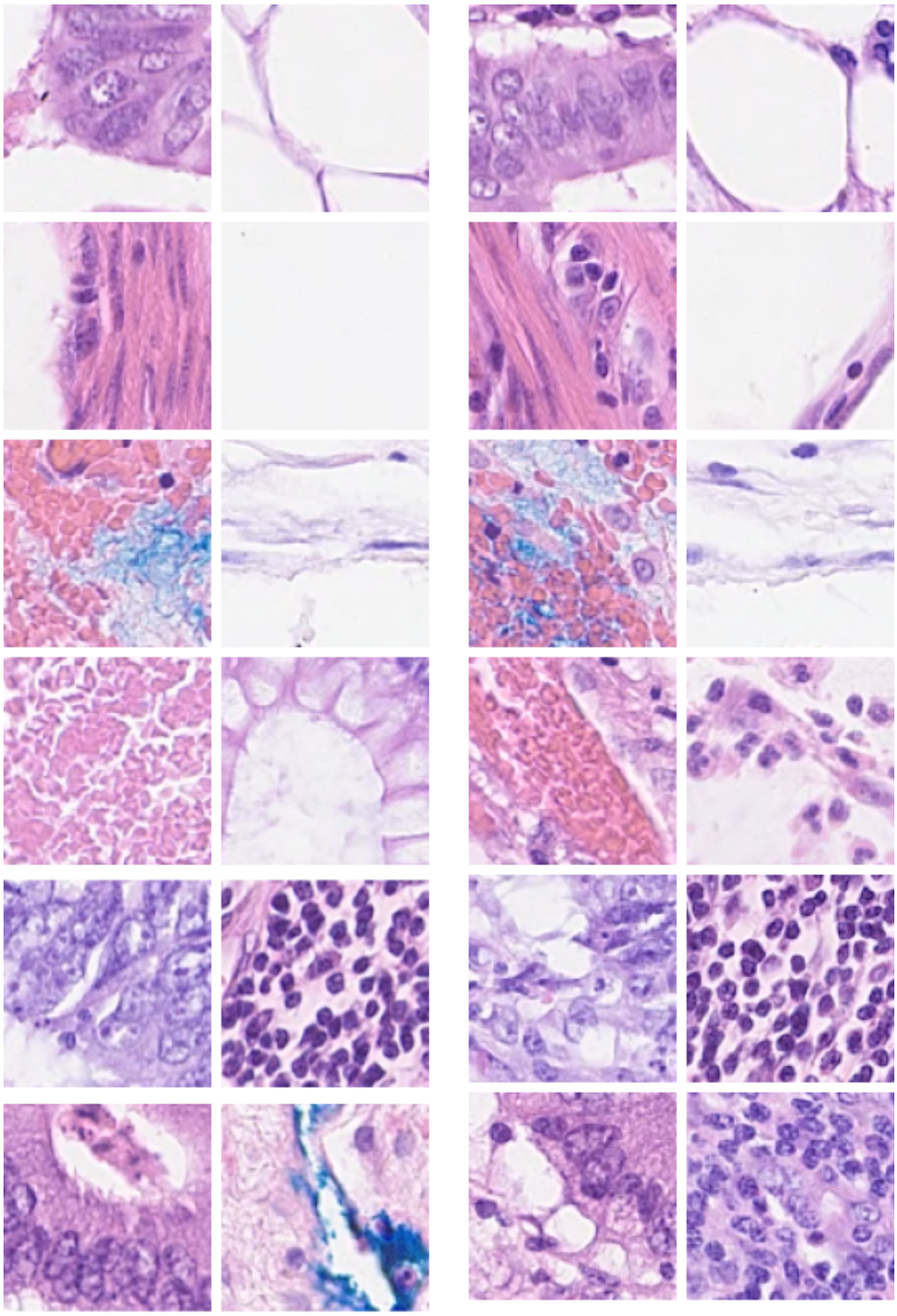

Figure 7.81: warwickj crc: 0 (left) and 1 (right)

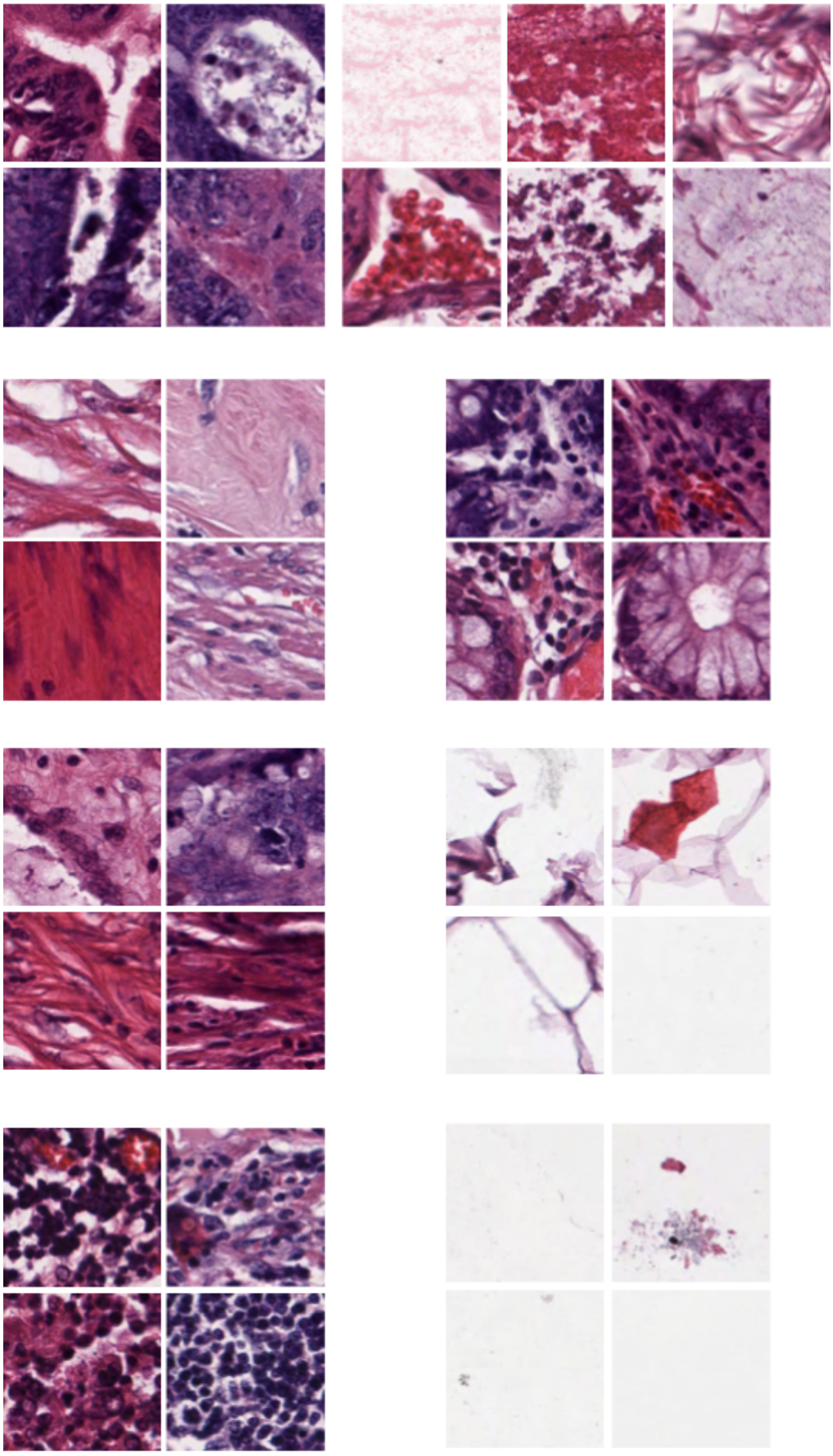

Figure 7.82: Umcm colorectal: classes in order

E. Retrieved images for models 3, 5, 9, 11, 15 and 22, 23, 29, 33, 38

Supervised models (3, 5, 9, 11, 15)

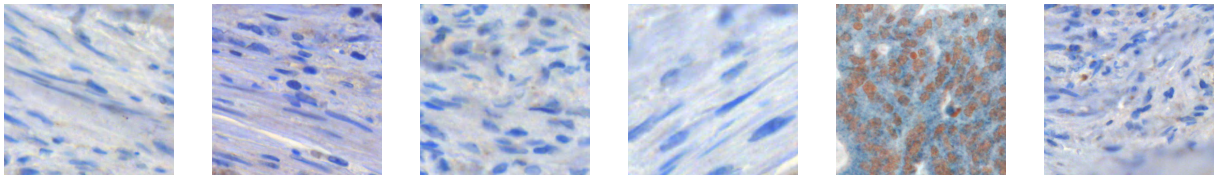

Figure 7.83: Query from lbpstroma_1, retrieved images from Model 3, 5, 9, 11 and 15

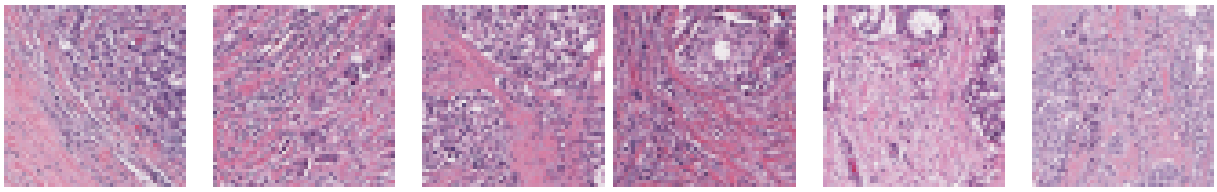

Figure 7.84: Query from jano6_1, retrieved images from Model 3, 5, 9, 11 and 15

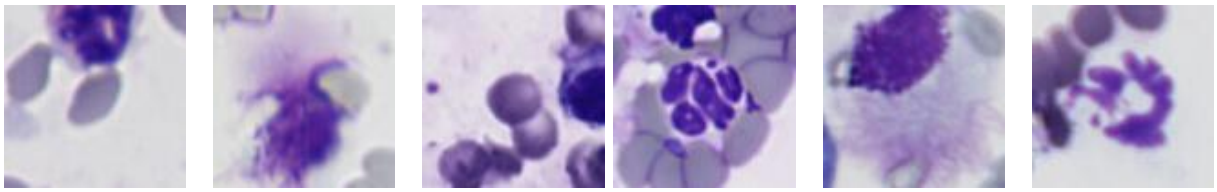

Figure 7.85: Query from ulg_lbtd_lba_5, retrieved images from Model 3, 5, 9, 11 and 15

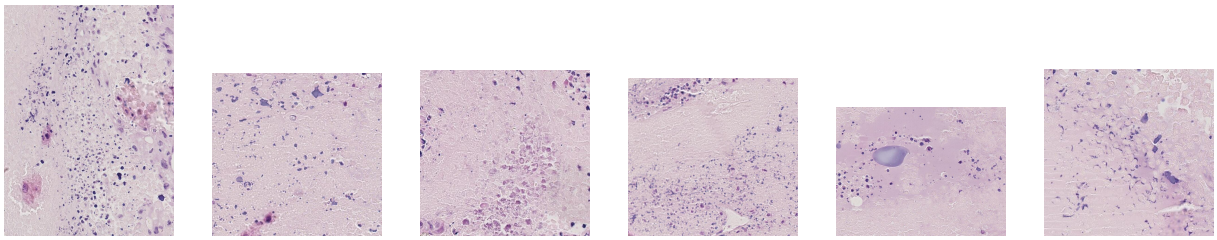

Figure 7.86: Query from ulg_lbtd2_chimio_necrose_1, retrieved images from Model 3, 5, 9, 11 and 15

Autoencoder models (22, 23, 29, 33, 28)

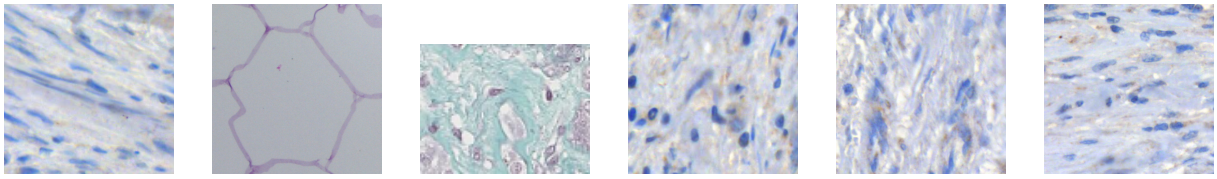

Figure 7.87: Query from lbpstroma_1, retrieved images from Model 22, 23, 29, 33 and 38

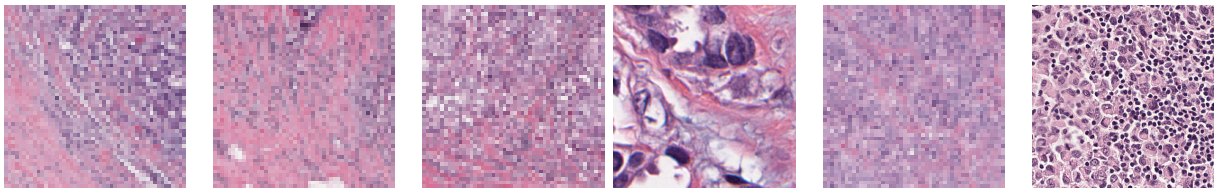

Figure 7.88: Query from jano6_1, retrieved images from Model 22, 23, 29, 33 and 38

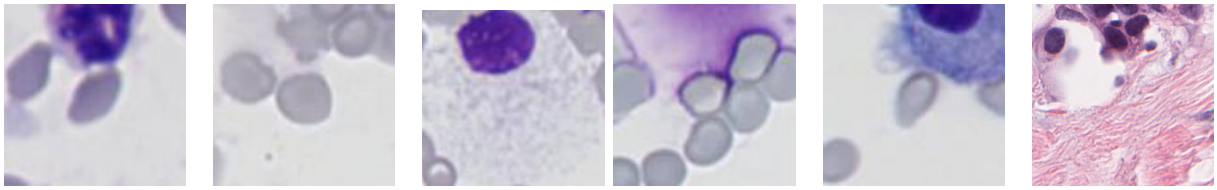

Figure 7.89: Query from ulg_lbtd_lba_5, retrieved images from Model 22, 23, 29, 33 and 38

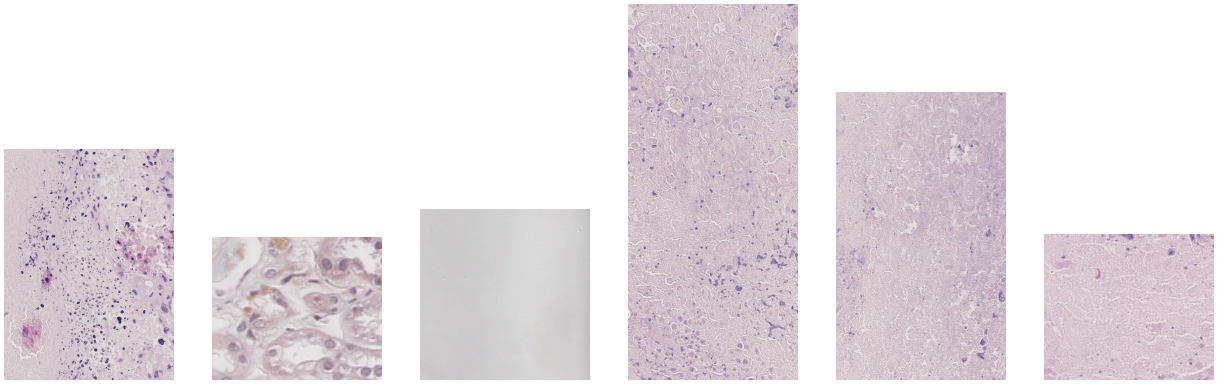

Figure 7.90: Query from ulg_lbtd2_chimio_necrose_1, retrieved images from Model 22, 23, 29, 33 and 38