A distributed deep learning approach for histopathology image retrieval

Auteur : Defraire, Stephan
Promoteur(s) : Maree, Raphael
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/13016

University of Liège

Faculty of Applied Science

A distributed deep learning approach for histopathology image retrieval

Master thesis realised to obtain a master degree in engineering in computer science

Author:
Stephan Defraire

Supervisor:
Raphaël Marée

Academic year 2020-2021

Abstract

Digital microscopy and radiology generate growing amounts of imagery data. To help practitioners find the information crucial to establish the most accurate possible diagnoses, Artificial Intelligence tools need to be developed.

This master thesis, based on the study of existing literature and open-source code, proposes a distributed deep learning architecture that allows a user, by using a fast approximate nearest neighbour search, to retrieve similar histopathology images to a query image.

The retained Deep Learning architecture, ResNet50 with some modifications, was distributed on different servers in order to allow the handling of up to million or billion images.

It was trained on a large-scale dataset of 67 classes of annotated medical images and the obtained results are quite promising, as well for the visual similarity of the retrieved images as for the search time. This research also analyses the generalisation to classes on which the system was not trained, and the impact of the approximated search on the accuracy and the retrieval time.

Nevertheless, even though the results are positive, this system might present some limitations as it was tested on only one dataset and was not reviewed by medical practitioners.

Acknowledgements

This thesis marks the end of my five years as a student at Uliège and I would like to express my gratitude to my professors who have passed on part of their knowledge to me and, of course, to my family and friends who have kept me motivated and engaged through their encouragements, and long and unfailing support.

I want to especially thank both Raphaël Marée, my supervisor, who initiated the Cytomine research project, and Romain Mormont, PhD student in machine learning, for their guidance and valuable feedback during our numerous and interesting meetings. I really appreciated the countless hours they spent reading my drafts and for providing valuable information despite their busy schedule. Without their insight and contributions, this work would simply not have been possible.

I am only too happy to have been pointed towards such a great altruistic project. I simply hope this modest contribution is added to the Cytomine platform, and most of all, that it will help pathologists establish more accurate diagnoses.

Contents

List of Abbreviations

AI Artificial Intelligence. 1

BoB Bunch of Barcode. 14, 15, 71

CBIR Content Based Image Retrieval. 1, 3–5, 10, 71

CNN Convolutional Neural Network. 6–8, 17, 29

DML Deep Metric Learning. 19, 20, 29, 30, 32, 42

DR Deep Ranking. 17, 19, 25, 29, 32, 42, 43, 73

MLP Multilayer Perceptron. 5, 6

SGD Stochastic Gradient Descent. 7

TCGA The Cancer Genome Atlas. 12, 13

WSI Whole Slide Image. 1, 2, 13–17, 23, 25, 64, 71

Chapter 1

Introduction

This chapter explains the motivation and the context of this research. It also states the objectives to be attained. And finally, it details the structure of this work.

1.1 Motivation

The expanding use of digital microscopy and radiography in pathology departments contributes to large amounts of imagery data. These fast-growing collections of images can convey a lot of useful and relevant information for medical practitioners. To ensure easy retrieval of the details that will help establish diagnoses in the most accurate and rapid way, efficient Artificial Intelligence (AI) tools need to be further developed.

At the University of Liège, Cytomine ULiège Research and Development, a group of computer science researchers, is involved in such projects. In the context of BigPicture European project, one of their goals is to create the biggest database of pathology images to increase the development of AI in medicine [1] [2].

In 2010, they initiated the Cytomine open-source project to facilitate the work of multidisciplinary teams dealing with very large imaging data. Thanks to their initiatives, remote collaboration is made possible through sharing of images, algorithms, and quantitative results over the web. Cytomine is used worldwide in a wide range of areas: biomedicine, digital collections, industrial quality control, ... [3]

1.2 Objectives

Whole-slide images (WSI) refer to the digital scan of a tissue section. Thanks to the Cytomine platform, those WSIs can be inspected on a web browser (Figure 1.1). Manual annotations can be added by selecting parts of a WSI (Figure 1.2). To help practitioners or researchers diagnose unknown features to them, they could select the unknown feature, and retrieve similar annotated images from a database based on a similarity measure (Figure 1.3). This is called Content-Based Image Retrieval (CBIR).

[1] `https://uliege.cytomine.org/`
[2] `https://cytomine.be/`
[3] Ib. 1 and 2

Figure 1.1: Whole-slide image(WSI) of a lymph node section image captured from the Cytomine platform.

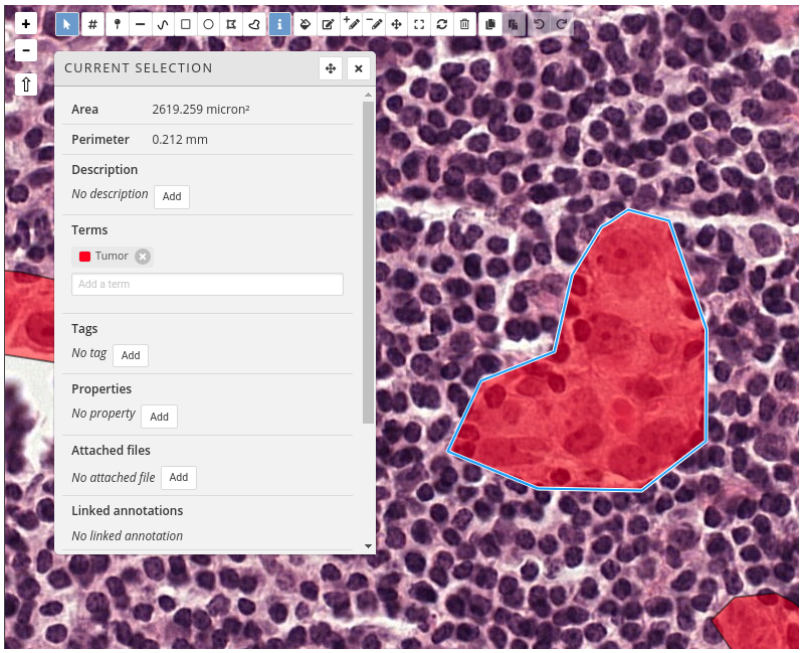

Figure 1.2: Manual annotation of a whole-slide image.

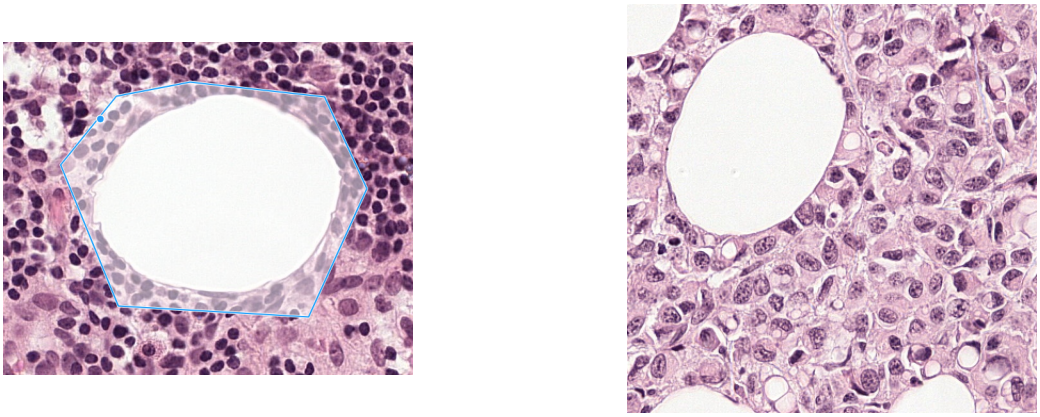

Figure 1.3: Illustration of CBIR. Left: feature selected by a practitioner. Right: one of the returned annotated images.

A Content-Based Image Retrieval (CBIR) system was previously developed by the Cytomine R&D team. This method was built on a randomised tree-based method and applied on small scale datasets. But the goal here is to apply the method to potentially million or billion images.

To do so, this master thesis will focus on the literature on deep learning-based methods, implement some of these and compare their performances on a large dataset of digital pathology image annotations acquired over several years using the Cytomine platform. This study would enable to select the best method to integrate into the Cytomine web plateform in the future, e.g. in the context of the BigPicture project where millions of images will be collected.

Another major goal of this thesis is to store the dataset on different servers, and if possible, integrate the new application into Cytomine.

1.3 Structure

This thesis is organised as follows:

- Chapter 2 gives insight into the background necessary to understand Deep Learning. It then summarises the key publications that contributed to the creation of the new code and the implementation of the training method;
- Chapter 3 is dedicated to the methodology adopted to fulfil the first key objective;
- Chapter 4 analyses and compares the obtained results;
- Chapter 5 is devoted to the distribution of the retained method on different servers, i.e. the second main objective;
- Chapter 6 presents the conclusions and makes suggestions for the future work of this research.

Chapter 2

Theoretical basis and state-of-the art content-based image retrieval methods

First of all, for the sake of clarity, it is important to emphasise that, in this work, the retrieval of similar images is carried out with Content-Based Image Retrieval (CBIR). It is different from other, most commonly used, types of image retrieval systems, such as Text-Based Image Retrieval where textual queries are used to describe desired images. In contrast, in CBIR systems, queries only rely on the visual content of the image, and not on text entered by a user. An example of a popular CBIR system is Google Reverse Image Search Engine.

With such a system, the user must enter an image for which he/she wants to find similar images. This is the `query` image. In turn, Google Reverse Image Search Engine returns a list of similar images it found. Those images come from a specific database designed for the image retrieval problem. An image that is stored in this database and ready to be retrieved is said to be `indexed` in this database. Examples of retrieved images with Google Reverse Image Search Engine are shown in Figure 2.1.

To facilitate understanding, this chapter first presents the concept of Deep Learning as it is used by most of the recent approaches for CBIR. Then, it summarises key publications that served as a basis to achieve the first main objective of this

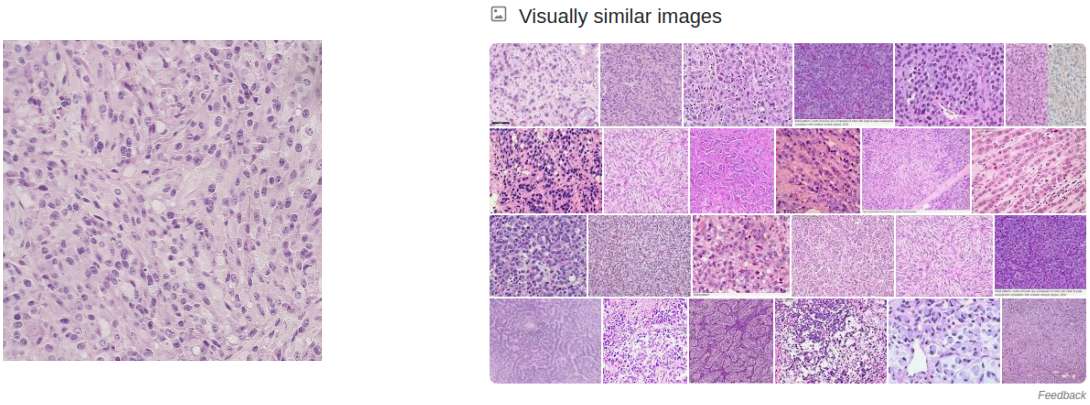

Figure 2.1: Examples of retrieved images with Google Reverse Image Search Engine (right), based on a histopathology query image (left).

thesis, namely create an open-source implementation to enable content-based image retrieval on histopathology images using Deep Learning. These recent and/or most popular publications in the field of digital image retrieval, can, for the purpose of this thesis, be divided in two categories.

The first category describes specific CBIR systems that were previously developed for histopathology image retrieval. These methods are compared and their key advantages are discussed to identify which features could be reused for our purpose. In a second step, we analyse further deep learning methods for image similarity computation as well as methods for approximate, fast indexing/retrieval, as our purpose is to apply such methods on large datasets. These different approaches will be evaluated on our large dataset in the next chapter.

2.1 Deep learning

This section was inspired by both [Zhang et al., 2021] and [Louppe, 2021], and it is intended to, first, throw light on the necessary background to understand the following sections of this chapter, and, second, present the image feature extractors that will be used for this thesis.

2.1.1 Neuron

The building block of deep learning is the neuron. It is a unit with n inputs, each of which is associated a weight w_i, $0 \leq i \leq n$, and a bias b. If $\mathbf{x}$ is an input of size n, the output of a neuron is:

$$y = \sigma(\mathbf{x}\mathbf{w} + b) \tag{2.1}$$

where σ is the activation function of the neuron. This function must be differentiable, and decides whether a neuron should be activated or not, while adding non-linearities to the network.

2.1.2 Multilayer perceptron

By combining several neurons, it is possible to create very complex networks. The simplest one is the Multilayer Perceptron (MLP).

Putting several neurons side-by-side creates a linear layer, and the multilayer perceptron is made of a succession of those. The last layer is called the output layer and each intermediate layer is called a hidden layer (Figure 2.2).

If an arbitrary layer $i \geq 1$ of h neurons has an input $\mathbf{x}_{i-1} \in \mathbb{R}^n$, this layer has a weight matrix $\mathbf{W}_i \in \mathbb{R}^{n \times h}$ and a bias vector $\mathbf{b}_i \in \mathbb{R}^h$. Therefore, the ouput of the layer $\mathbf{x}_i \in \mathbb{R}^h$ is given by:

$$\mathbf{x}_i = \sigma(\mathbf{x}_{i-1}\mathbf{W}_i + \mathbf{b}_i) \tag{2.2}$$

The output $\mathbf{y}$ of a multilayer perceptron, given an input $\mathbf{x}$, of l layers is computed by using equation 2.2 in series, where $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{y} = \mathbf{x}_l$.

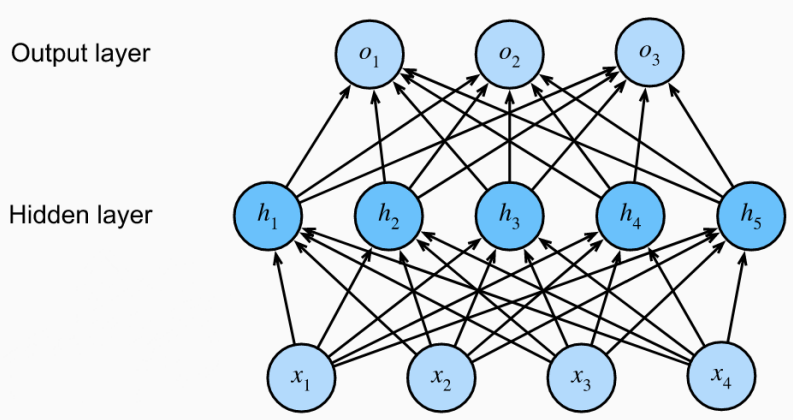

Figure 2.2: Visualisation of a multilayer perceptron [Zhang et al., 2021].

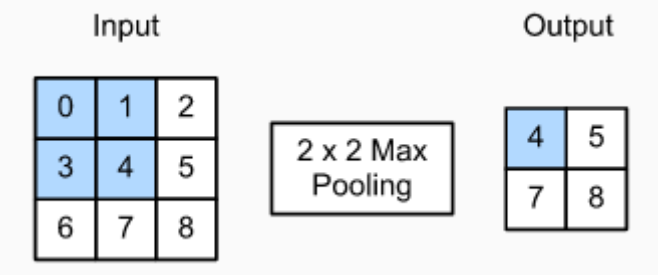

Figure 2.3: Illustration of the `max` pooling [Zhang et al., 2021].

2.1.3 Convolutional neural network

MLPs are not suited for processing images as they cannot capture spatial information about the pixels. One application of the Convolutional Neural Networks (CNN) aims to compensate the incapability of the MLPs to understand images well.

The main operation of a CNN is the convolution: for a 3D tensor $\mathbf{x} \in \mathbb{R}^{C\times H\times W}$ and a convolution kernel $\mathbf{u} \in \mathbb{R}^{C\times h\times w}$, the output is a 2D object $\mathbf{y} \in \mathbb{R}^{(H-h-1)\times(W-w-1)}$:

$$\mathbf{y}_{j,i} = \mathbf{b}_{j,i} + \sum_{c=0}^{C-1} (\mathbf{x}_c \circledast \mathbf{u}_c) = \mathbf{b}_{j,i} + \sum_{c=0}^{C-1} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{c,n+j,m+i} \mathbf{u}_{c,n,m} \tag{2.3}$$

More informally, $\mathbf{u}$ acts as a filter sliding on the image.

By stacking several convolutional layers, in the same way as for the MLP, and using pooling operations to reduce the size of the input of a layer, one builds a convolutional neural network, that is capable of processing images well. There are two kinds of pooling operations used in CNN: the `max` pooling, and the `average` pooling. The `max` pooling takes the maximum value in the window of the image, and reports this value for the output (Figure 2.3). The `average` pooling does almost the same, but reports the average value in the window instead.

2.1.4 Training a neural network

For a neural network to provide decent results, a particular set of weights and biases need to be found. For the CNN, the weights to be found are the convolution kernels weights.

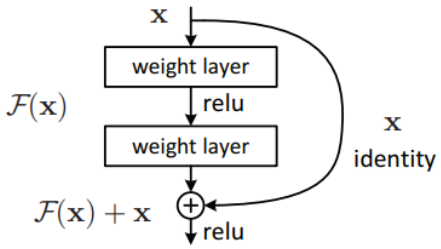

Figure 2.4: Shortcut connection [He et al., 2016]

The main problem is to find the right set of weights and biases that will provide the best results possible. This set is found during the operation called training.

In this master thesis, only supervised learning algorithms are used: this section will therefore only cover supervised learning.

The main idea of supervised training is to provide the network elements of a labelled dataset to make predictions. Those predictions can be compared to the ground truth, i.e. the true prediction to make. This comparison with the ground truth is made possible by using a `loss` function, or loss for short, that results in a training error to minimise.

In order to minimise the training error, an optimisation algorithm called Stochastic Gradient Descent (SGD), or one of its derivative such as the Adam optimiser, is used. It is an iterative algorithm, that slowly converges towards a local minimum of the loss function by updating the weights and biases of the network accordingly. Each iteration is called an `epoch`, and it consists of having forwarded each sample from the training dataset in the network. Usually, the samples are provided to the network by batches of a given size. This allows to speed up the training and to reduce the variance of the gradient estimation. The weights and biases of the network are updated after each batch.

2.1.5 ResNet50

The paper "Deep Residual Learning for Image Recognition" [He et al., 2016] solves a problem happening during the training of a deep CNN known as degradation: when the network depth increases, the accuracy gets worse. This is unexpected because the network should be able to model more complex functions when it has more layers.

This degradation actually might indicate that the optimiser is having difficulty approximating identity mappings by multiple non-linear layers. For this reason, the authors introduce the shortcut connection:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{W}_i) + \mathbf{x} \tag{2.4}$$

where $\mathcal{F}$ represents a mapping to be learned, with weights $\mathbf{W}_i$. Figure 2.4 shows a graph of the operations.

With this formulation, the optimiser can simply put the weights to 0 to approximate a linear mapping. The shortcut connection also has the auxialiary benefit of preventing gradient vanishing during training. ResNet50 is a deep CNN that uses those shortcut connections and that will be used as a feature extractor for this master thesis.

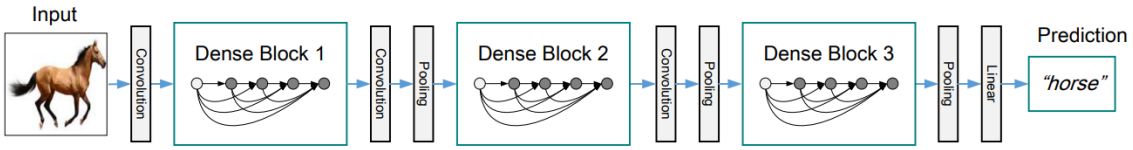

Figure 2.5: DenseNet [Huang et al., 2017]

2.1.6 DenseNet-121

The paper "Densely Connected Convolutional Networks" [Huang et al., 2017] introduced the concept of DenseNets, that solves a problem related to ResNets: the identity function and the output of the function $\mathcal{F}$ are combined by summation, and this may hurt the information flow in the network.

As a solution, the authors propose the dense connectivity: the output of a layer is connected to all the subsequent layers. More formally, layer i receives the outputs of all its preceding layers $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$:

$$\mathbf{x}_i = H_i([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}]) \tag{2.5}$$

where H is a non linear transformation.

DenseNet-121 is a deep CNN that uses this dense connectivity. It is made of several densely connected dense blocks, between which there are transition blocks. Figure 2.5 shows a diagram of the DenseNet architecture. It will also be used as a feature extractor in this master thesis.

2.1.7 DeiT

The paper "Training data-efficient image transformers & distillation through attention" introduces the network DeiT. It is an image classifier that does not make use of convolution, but that instead uses the Transformer architecture [Vaswani et al., 2017]. It can be used as a feature extractor in the context of similar image retrieval, just as ResNet50 or DenseNet-121. This Transformer and the Vision Transformer are described here below.

2.1.7.1 Transformer

The Transformer makes use of the attention mechanism. In the context of the attention mechanism, a query vector $\mathbf{q} \in \mathbb{R}^d$ interacts with a matrix of keys $\mathbf{K} \in \mathbb{R}^{k \times d}$ to obtain the output. This output is the weighted sum of the k vectors of the value matrix $\mathbf{V} \in \mathbb{R}^{k \times d}$. In the case of the Transformer, the weights of the sum are obtained with the following rule:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^\top / \sqrt{d})\mathbf{V} \tag{2.6}$$

The Transformer takes the attention mechanism two steps further: it uses self-attention, and a Multi-Head attention layer.

Given a sequence of input tokens $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, where $\mathbf{x}_i \in \mathbb{R}^d, 1 \leq i \leq n$, the self-attention outputs a sequence $\mathbf{y}_i$ of the same length [Zhang et al., 2021]:

$$\mathbf{y}_i = f(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), (\mathbf{x}_2, \mathbf{x}_2), \ldots, (\mathbf{x}_n, \mathbf{x}_n)) \tag{2.7}$$

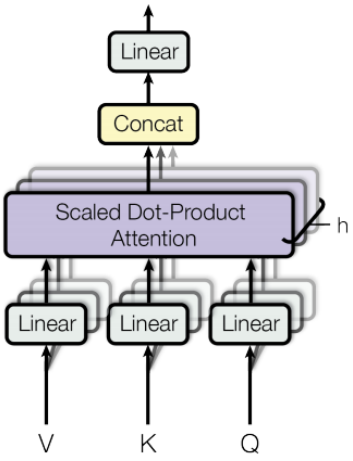

Figure 2.6: Multi-Head attention [Vaswani et al., 2017]

The output of the self-attention shows how relevant vector $\mathbf{x}_i$ is to the other vectors in the sequence.

Last but not least, the Multi-Head attention uses h attention transformations with different weights. The results of each transformation are concatenated and given to a linear layer (Figure 2.6).

Put all together, this describes the Multi-Head Attention block in Figure 2.7. Given the sequence is processed in parallel, the network needs positional information about each element in the sequence: this is the role of the Positional Encoding block. As can be seen in Figure 2.7, the Transformer architecture also uses the shortcut connections that are used in ResNet50.

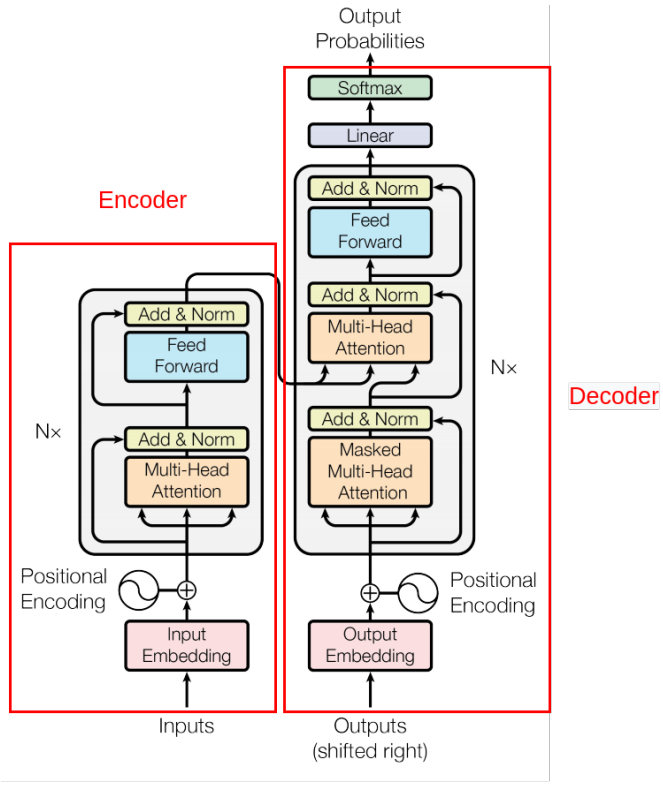

Figure 2.7: Transformer architecture [Vaswani et al., 2017]

DeiT only uses the encoder Transformer, so the decoder does not need to be explained.

2.1.7.2 Vision Transformer

DeiT is an enhancement of ViT [Dosovitskiy et al., 2020], but the main workflow remains globally the same. The image is first divided into several patches, that are unrolled into a sequence. Those patches are flattened and transformed with a trainable linear projection.

In the Vision Transformer, the Positional Encoding is learned during training, which is not the case in the original Tranformer architecture.

The [`class`] token is an extra learnable parameter prepended to the sequence given to the encoder. Only the output of the encoder corresponding to this [`class`] token is fed to the classification layer.

Figure 2.8 summarises those operations.

DeiT is used in this work instead of ViT as it is more data-efficient. Indeed, ViT requires to be trained on too large datasets to be effective.

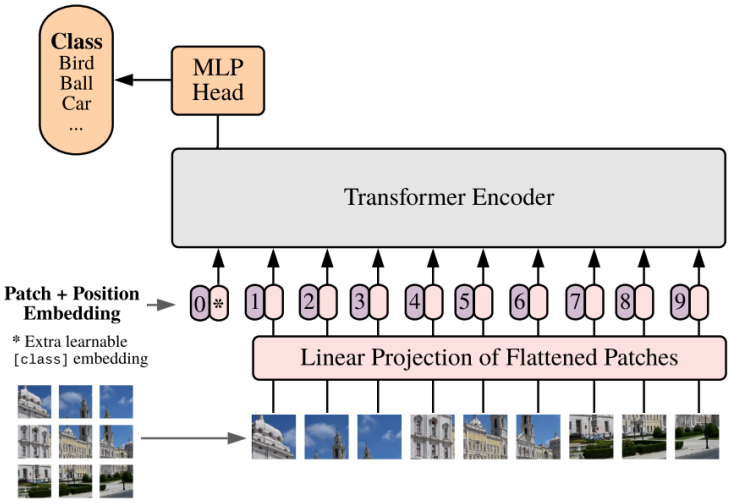

Figure 2.8: Transformer architecture [Dosovitskiy et al., 2020]

2.2 Publications describing specific CBIR systems

This section first summarises the methods used by the previous CBIR system that was implemented on the Cytomine platform. It then summarises either popular or recent publications based on deep learning in the field of histopathology content-based image retrieval.

2.2.1 Incremental Indexing and Distributed Image Search using Shared Randomized Vocabularies

The paper "Incremental Indexing and Distributed Image Search using Shared Randomized Vocabularies" [Marée et al., 2010] (later referred to as randomised vectors system) proposes a distributed CBIR technique based on randomised vectors. This image retrieval technique was the one implemented in Cytomine in 2010. It

stands out from the ones summarised hereafter, as it is the only one that presents a fully unsupervised approach. The code is available at the following address : `https://github.com/cytomine/CBIRetrieval`

2.2.1.1 Methods

A centralised architecture for the distributed approach is developed: a central server, that might be the client, is aware of a network of cooperating image servers; each of these image servers stores and indexes a part of the complete image dataset.

How is the image similarity measure derived? A common mapping structure is deployed on each image server and on the central server, and it assigns multiple visual words to multiple patches of an image. A visual word is a tuple describing a patch that will be defined later. This mapping structure is made of an ensemble of T vectors V_t $(t = 1, t = 2, \dots t = T)$. Each of these vectors has a fixed size and is constructed in the following way: V_t is composed of the m binary tests $(test_1(t), \dots test_m(t))$. Each test is chosen this way: $test_i(t) \equiv 1(x_{j_i} < th_i)$, where the attribute x_{j_i} and the threshold value th_i are randomly chosen.

The threshold values are the same for the central server and each image server. Each image server manages its own index. For each new reference image I_R stored, the image server extracts patches of random sizes at random locations. These are resized to 16×16. Each of these patches is then mapped by each V_t to a binary code $B = b_1 \dots b_m,\ b_i = 1$ if $test_i(t) = 1,\ 0$ otherwise. The tuple (B, t) is a visual word describing a patch. Therefore, each patch is mapped by each V_t to a visual word, which is indexed through a hash table. For each t and each B, a list of pairs is maintained and composed of image identifiers I_R and the number $N_{I_R,B,t}$ of patches of I_R mapped by V_t to that visual word, as well as the total count $N_{Blocal,t}$ of patches of the local image set mapped to this visual word.

An estimation of the similarity measure between a query image I_Q and a reference image I_R is given by :

$$k(I_Q, I_R) = \sum_{t=1}^{T} \frac{1}{T} \sum_{B \in \nu_{I_Q,t}} \frac{1}{N_{B,t}} \frac{N_{I_Q,B,t}}{N_{I_Q}} \frac{N_{I_R,B,t}}{N_{I_R}} \tag{2.8}$$

where $\nu_{I_Q,t}$ is the set of non-empty visual words induced by the vector V_t for the query image I_Q, $N_{B,t}$ is the number of patches from all reference images that are mapped to word B by V_t, $N_{I_Q,B,t}$ (resp. $N_{I_R,B,t}$) is the number of patches from I_Q (resp. I_R) that are mapped to B by V_t and N_{I_R} and N_{I_Q} are respectively the number of patches extracted from the reference image and the query image.

A communication protocol is implemented between the central server and the image servers to retrieve similar images, as each image server must known the value $N_{B,t}$ to compute the similarity. When a client makes a request, he/she sends a list $\mathcal{B}$ of triplets $(B, t, \frac{N_{I_Q,B,t}}{N_{I_Q}})$ to the central server (the central server can compute it, if the client does not know the mapping structure), that describes his/her query image. The central server sends each (B, t) to the central server, to request their value $N_{Blocal,t}$. The central server can then compute the global value $N_{B,t}$, and send the four-tuples $(B, t, \frac{1}{N_{B,t}}, \frac{N_{I_Q,B,t}}{N_{I_Q}})$ to each image server. The servers can then compute the global similarity between the query image and its local index images, and return the top list to the central server, that forwards the list to the client.

2.2.1.2 Datasets and results

The solution proposed in the paper has been tested on three different datasets : IRMA-2005, SPORTS and a dataset composed of whole-slide histopathology images.

IRMA-2005 is composed of 10,000 X-ray images : 9000 are used for the database, 1000 for the query. About 40 methods were evaluated on this dataset with results ranging from 26.7% to 87.4%. The proposed method scored 81.6%. This inferior score is due to the unsupervised approach of the method.

SPORTS consists of 2449 images of sport grouped into five classes. 75% of images are used as the reference dataset, 25% for the query. A score of 71.02% is obtained (average accuracy per class). The results obtained are better than in a reference that uses supervised approaches.

Histopathology is made of 8 whole-slide images of about 20,000×20,000 pixels. These images are divided into tiles of 256×256. Because ground-truth is not available for such amount of data, only qualitative results are shown.

2.2.2 Similar image search for histopathology: SMILY

The paper "Similar image search for histopathology: SMILY" [Hegde et al., 2019] proposes a deep-learning based reverse image search tool for histopathology images: Similar Images Like Yours.

2.2.2.1 Methods

Smily is based on a convolutional neural network called a deep ranking network (see Section 2.3.1). Unfortunately, there seems to be no model publicly available, neither is the dataset used for learning.

The network is trained on about 500,000,000 natural images from 18,000 distinct classes (it is never trained on histopathology images). In this way, the network learned to distinguish similar images from dissimilar ones by computing and comparing the embeddings of input images. This network was successfully leveraged to generate embeddings that discriminated between cellular phenotypes in high-content screening.

To test the database, slides from The Cancer Genome Atlas (TCGA, public dataset) were used. The slides are divided into patches of 300×300 and compressed into embedding vectors of size 128. Moreover, the four 90° rotations and the mirrored versions for the patch are generated as well as their corresponding embeddings (Figure 2.9). In the absence of any compression, storing the embeddings only requires a 0.4% storage overhead.

To query the database, Smily computes the embedding of the query patch, and then compares this embedding with those stored into the database (with Euclidian distance). Only the most similar orientation is returned, and no results within 1000 pixels from each other is returned. For efficient lookups, k-d trees are used, with leaf size of 40 and depth of 6. The authors provide no explanation to justify these choices. Furthermore, the search is parallelised. To perform a search, the user has

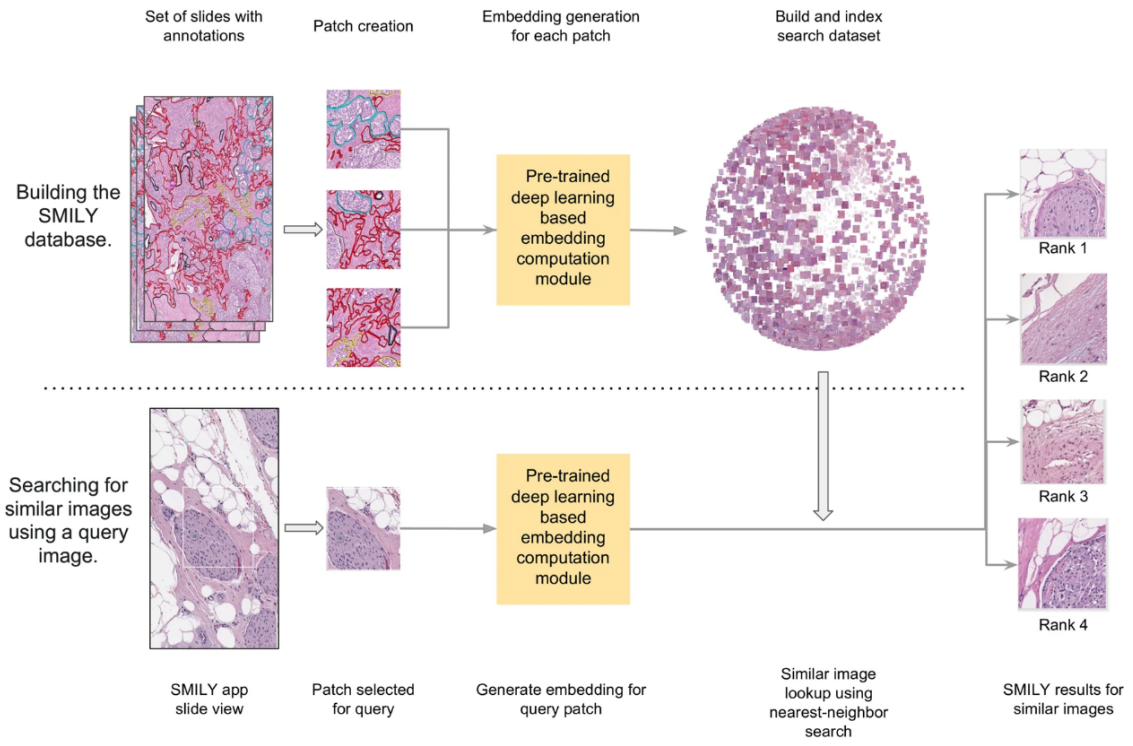

Figure 2.9: The two major steps of the Smily architecture. Top: indexation of annotated WSIs in the database. Bottom: retrieval of similar patches. [Hegde et al., 2019]

to provide a patch of height and width between 200 and 400 pixels (Figure 2.9). The patch is resized to 224×224.

2.2.2.2 Results

The evaluation used 127,000 patches of 45 slides from TCGA, and 22,500 query patches from another 15 slides, which are not identified in the paper. The patches were annotated with various non-exhaustive histologic features, however, no mention is made of how many features were used. Their annotations are not made public, but some annotations of TCGA are available on GDC website. A top-5 score was used to assess the performance of retrieving patches with the same histologic features.

On prostate specimens, Smily achieved a top-5 score of 62%, which is significantly higher than SIFT, a traditional feature extractor (44.2%). When Smily retrieved results that did not exactly match the histologic feature, it commonly returned a similar feature. When the search was expanded to multiple organs, the histologic feature match score was at 65.3%, but the combined histologic feature and organ match was lower at 40.0%.

Smily was also evaluated by pathologists. This is a crucial step because if a query image contains only fat, a retrieved image search result that contains both fat and an artery (but is only annotated as "artery") will be considered as an error during the prior evaluation. As a control to ensure that graders were not artificially scoring Smily results highly, some search results were from a random search instead of Smily. On prostate specimens, a top-5 score of 62.1% to find similar histologic features was achieved. On multiple organs, it achieved 57.8%.

The parallel computation for the lookup is indeed needed : using 400 computers

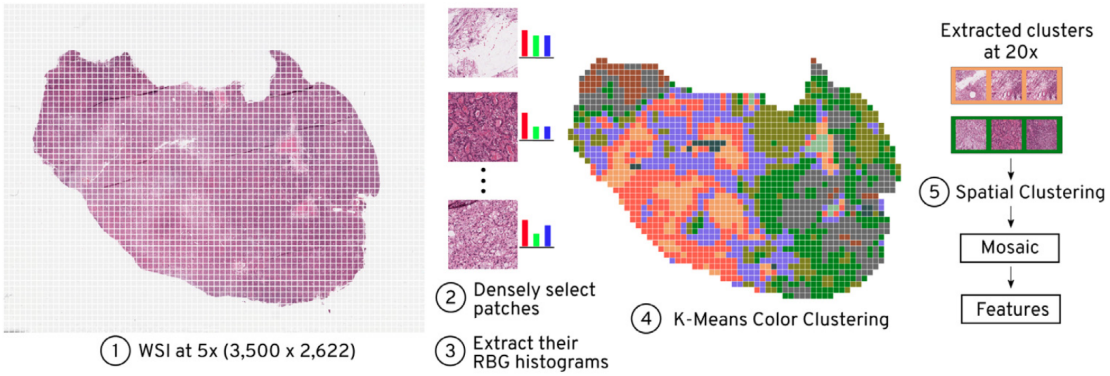

Figure 2.10: Construction of the mosaic for Yottixel. The WSI is divided into patches, that are clustered based on their colour and their location. Only one patch from each cluster is selected to be part of the mosaic. [Kalra et al., 2020]

with ten compute threads each, queries had a median query time of 1.3s. With 100 less patches in the database while using a naive loopkup, the query time was 25 seconds.

2.2.3 Yottixel – An Image Search Engine for Large Archives of Histopathology Whole Slide Images

The paper "Yottixel" [Kalra et al., 2020] proposes an unsupervised technique to efficiently create a representative set of patches of images.

2.2.3.1 Methods

There are two major phases in the execution of Yottixel : (i) the offline indexing phase and (ii) the run-time search.

Offline indexing phase Yottixel indexes a WSI in two steps: by creating its mosaic (a set of representative patches) and by converting the mosaic to a Bunch of Barcodes (BoB). To create the mosaic of an image, a WSI is first segmented into K_{ch} regions based on their colour, using k-means (Figure 2.10). This segmentation frequently results in the segmentation of different tissue types. A small percentage of patches (5%) is then randomly selected by preserving the spatial diversity of the patches. This is done by using k-means for grouping the patches based on their location. The reason k-means is used a second time instead of using random sampling is that random sampling will provide inconsistent results.

After the mosaic is created, the patches are converted into a set of barcodes (Figure 2.11). First a patch is converted to a feature vector using a DenseNet, which is pretrained on ImageNet. DensetNet seems to capture more compound/-complex patterns within histopathology images than VGG19, Inception or in-house solutions. After the extraction, a discrete differentiation or the MinMax algorithm [Tizhoosh et al., 2016] is used to enable a fast Hamming distance search. The MinMax algorithm takes a matrix as an input. The matrix is projected into several vectors, and these vectors are smoothed in order to remove small peaks and valleys. Then all the maxima and minima of the vectors are found : the elements that are

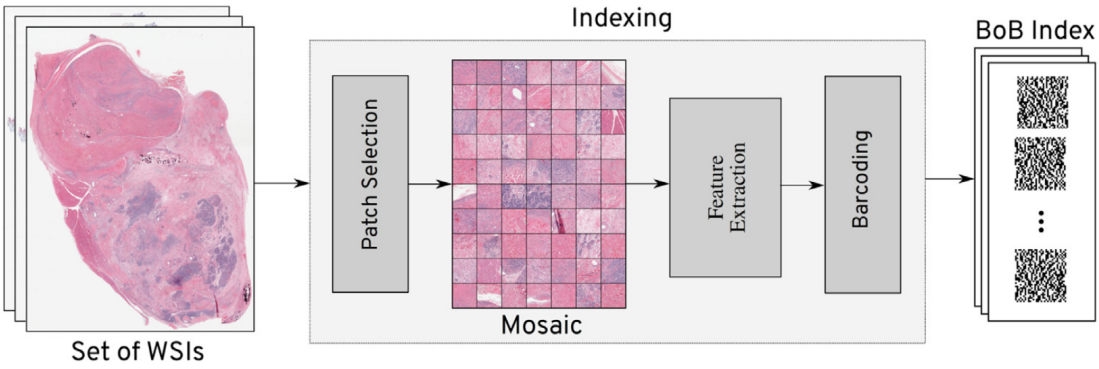

Figure 2.11: Derivation of the BoB index for Yottixel. The mosaic of the slide is first computed, and each feature vector of each patch is binarised. [Kalra et al., 2020]

between a maximum and a minimum are set to 1, 0 otherwise. Finally, all the vectors are appended to each other, and the result is returned. For an average 700MB WSI, the BoB index is as small as 10KB.

Run-time search There are two modes of searching : a vertical and a horizontal search. The vertical search is confined to the same organ as the query patch ; for the horizontal one, the entire index is searched.

2.2.3.2 Datasets and results

Two datasets are used to validate the efficiency of Yottixel. The first one is private and composed of 300 WSIs across more than 80 diagnoses on multiple organs (104GB). The second one consists of 2,020 WSIs from The Cancer Genome Atlas (2TB). Only WSIs that contain Formalin-Fixed Paraffin-Embedded tissue are kept. Each slide is labelled with the diagnoses and the organ. While the exact list of slides the authors have used is not available, it is possible to know which slide contains FFPE tissue, because it is mentioned in the name of the file.

The researchers observed that their quantitative results were three to four times better than a random approach. They also noticed that satisfactory results can be obtained with a smaller mosaic (except for rare organs or diagnoses). However, they did not compare their results with other works.

The most interesting result is the one provided by users' feedbacks. These users were three pathologists and users with knowledge in computer vision. They were presented a query image and the corresponding top three search results. The results were however presented in a random order, and they were asked to label the results with the very poor, poor, fair, good or very good label. The top results were assigned the most very good labels, and pathologists assigned more very good labels to the top results than other users. The top results were also assigned more very good labels than the other results. In a general way, the results were all quite positive.

2.2.4 Luigi: Large-scale histopathological image retrieval system using deep texture representations

The paper "Luigi" [Komura et al., 2018] proposes the first publicly available CBIR system for histopathological images.

2.2.4.1 Methods

The network used as the backbone of Luigi is VGG-16, which is only trained on ImageNet.

For the retrieval, the feature map of the layer "conv3 1" is used to compute an approximation of the following matrix :

$$G_{ij} = \sum_k F_{ik} F_{jk}$$

where G, F and k are respectively the correlation matrix, the feature map and the position in the image. This operation is called bilinear pooling, and results in a representation that has a high spatial invariance, but that has a too high dimension (65,536). Therefore, Compact Bilinear Pooling [Gao et al., 2016] is used. It is a method that approximates bilinear pooling with two random matrices, and does not need training. The final dimension of the feature vector is 1024. When making a query, the feature vector is computed for the query image and also for its rotation by 90°, 180° and 270°. Then an approximation of the nearest neighbour is searched in the database with randomised kd-trees. The distance metric used is the cosine similarity.

When a WSI is indexed in the database, patches of 256×256 are extracted from approximately 50% of the tissue area. Tissue areas are detected with Otsu's method, which is used for automatic thresholding. Patches are extracted at scale $\times 10$ and $\times 20$. Then, the feature vector of each patch is computed. k-means clustering ($k = 500$) is used to remove similar patches within a same slide, and only the patches that are the nearest from their centroid are stored in the database.

2.2.4.2 Dataset and results

The slides used in the database are from The Cancer Genome Altas (public dataset). The diagnostic of some slides were downloaded from the National Cancer Institute GDC legacy archive. Slides without enough information were discarded. As already mentioned, no dataset was used for training.

To assess the performance of the system, the authors have made queries for 4 cancer types. Then, the average precision is computed for three queries of each type for the top 20 images. For each cancer type, the precision is over 0.75. But most of the retrieved incorrect images seemed to be similar to the diagnosis of the pathologists, because the classification of cancer type is not always strict.

2.2.5 Discussion about the litterature

Based on the papers presented above, it seemed to make sense to explore the Randomised Trees system, Smily, Yottixel and Luigi more deeply.

This section discusses the characteristics of these systems, to highlight what can be exploitable to create a new effective architecture.

The randomised vectors system and Smily both propose a distributed approach. This makes them good candidates for very large collections of images: by splitting the images on different servers, the search will naturally be faster. In addition, in the randomised vectors system, the image servers can be added or removed at will, which means that it is fault-tolerant, in other words, a crashed image server will not cause the whole system to fail.

The randomised vectors system and Yottixel both present an architecture that uses patches of medical images for the similarity search, even though the first one is for images while the second one is for WSIs. But it can be assumed that if it works for WSIs, it can work for images.

Three systems, Smily, Yottixel and Luigi, are not trained on specific data. Smily is trained on natural images, and Yottixel and Luigi are only pretrained on ImageNet. This leads to the idea that, for this research, an exploration of datasets containing natural images will have to be carried out to check whether this can possibly impact positively on accuracy compared with an untrained model.

Finally, both Smily and Luigi retrieve similar images based on a nearest neighbour search of the feature vector of the images. A similar method for fast nearest neighbour search will be described in the next section.

2.3 Publications about general image similarity

This section, through summaries of a number of publications, outlines the deep learning methods for image similarity, and describes Faiss, a library for fast approximate nearest neighbours search, often used for image retrieval systems.

The purpose of this section is twofold. It wants, on the one hand, to look for relevant material to construct the new architecture, and, on the other hand, to render the above described systems comprehensible for the readers.

2.3.1 Learning Fine-grained Image Similarity with Deep Ranking

This paper [Wang et al., 2014] explains the key notion of deep ranking (DR), crucial to understand the Smily architecture. A convolutional neural network trained with DR allows to measure the distance between two images. The similarity measure between two images P and Q is defined as :

$$D(P, Q) = \|f(P) - f(Q)\|^2 \tag{2.9}$$

which is the squared Euclidian distance between the embedding vectors of P and Q. The smaller this distance is, the more similar the images are. In Equation 2.9, the embedding function $f(\cdot)$, by mapping an image P to its corresponding embedding vector, describes the input image. This definition formulates the similar image ranking problem as finding the closest neighbour of an image in the Euclidian space. The goal of this technique is therefore to find an embedding function $f(\cdot)$ that assigns smaller distances to more similar image pairs.

The network is composed of three convolutional networks in parallel : a ConvNet, which can be any deep CNN, encodes strong invariance and captures image semantics. The other two parts take down-sampled images and use only one convolutional layer and a max pooling layer to capture the visual appearance (Figure 2.12).

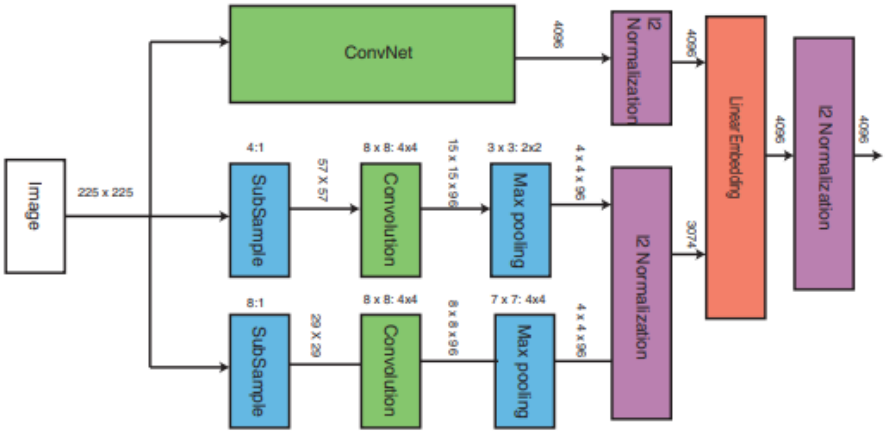

Figure 2.12: Architecture used for DR [Wang et al., 2014]

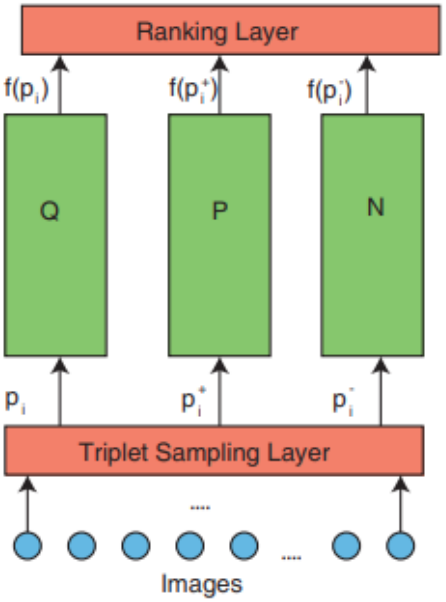

Figure 2.13: Training for DR [Wang et al., 2014]

During training, three of these networks, that share the same set of weights, are put in parallel: one is fed an image p_i, another one is fed a similar image p_i^+ and the last one is fed a dissimilar image p_i^- (Figure 2.13). Each of these networks computes the embedding vector of its input image. The following loss is evaluated and the gradient is back-propagated in each network:

$$\mathcal{L}(p_i, p_i^+, p_i^-) = \max(0, g + D(f(p_i), f(p_i^+)) - D(f(p_i), f(p_i^-)) \tag{2.10}$$

where g is a gap parameter.

The triplets $t_i = (p_i, p_i^+, p_i^-)$ are computed in the triplet sampling layer, for which the authors propose an algorithm to efficiently sample the triplets. This algorithm is based on the relevance of each image within its class. To compute this relevance,

they use the "golden feature", a weighted linear combination of 27 features. Some of the features are learned through image annotation data, and the weights of the linear combination are learned based on human rated data. This annotation can be a very tedious task. Another possibility is to randomly chose p_i^+ in the same class as p_i, and to select a random image from another random class for p_i^- (later referred to as the random sampling). But doing so will only be efficient if images from the same class are visually similar.

The ConvNet network is pretrained on the ImageNet dataset.

2.3.2 Deep metric learning

Just as in Deep Ranking, the goal of Deep Metric Learning (DML) is to find an embedding function $\phi : \mathcal{X} \to \Phi \subseteq \mathbb{R}^D$ mapping the datapoints $\mathbf{x} \in \mathcal{X}$ into an embedding space Φ [Roth et al., 2020]. It allows to measure the similarity between two datapoints $\mathbf{x}_i$, $\mathbf{x}_j$ as $d(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))$ with $d(\cdot, \cdot)$ a predefined distance function (in this case, the Euclidian distance). Therefore, Deep Ranking can be interpreted as a particular case of Deep Metric Learning.

Three key components have to be defined to train a DML model:

- the architecture of the model: the feature extractor that will serve as the embedding function $\phi(\cdot)$;
- the objective function: the training loss that will enforce the similarity measure;
- the data sampling strategy: a strategy that samples informative minibatches.

2.3.2.1 Margin loss

The Margin Loss [Wu et al., 2017] is a ranking-based objective used within the framework of DML. It leads to an objective similar to the DR objective: learning a function ϕ such that $d_\phi(\mathbf{x}_a, \mathbf{x}_n) - d_\phi(\mathbf{x}_a, \mathbf{x}_p) < \gamma$, where $\mathbf{x}_a$ is an anchor, $\mathbf{x}_p$ is a positive image (meaning $\mathbf{x}_a$ and $\mathbf{x}_p$ are of the same class), and $\mathbf{x}_n$ is a negative image ($\mathbf{x}_a$ and $\mathbf{x}_p$ are of different classes).

If $P = \{(i, j) | i, j \in \mathcal{B}\}$, $\mathcal{B}$ is a minibatch, then

$$\mathcal{L}_{margin} = \sum_{(i,j)\in P} \gamma + \mathbb{I}_{y_i=y_j}(d(\phi_i, \phi_j) - \beta) - \mathbb{I}_{y_i\neq y_j}(d(\phi_i, \phi_j) - \beta) \tag{2.11}$$

This loss differs from the triplet margin loss used for DR in the use of the learnable parameter β. In practice, the pairs (i, j) are obtained with tuple mining: as in DR, it is suboptimal to consider all the pairs you can make with the minibatch. Only one positive sample and one negative sample from the minibatch are selected per sample. In this case, distance weighted sampling [Wu et al., 2017] is used for the negative samples: the smaller the distance to the anchor is, the likelier this sample will be chosen to make a pair. The motivation is that the training will be more effective by selecting negative samples with small distances to the anchor than by selecting negative samples of already high distances.

$$q(d(\phi_i, \phi_j)) \propto d(\phi_i, \phi_j)^{D-2}[1 - \frac{1}{4}d(\phi_i, \phi_j)]^{\frac{D-2}{3}} \tag{2.12}$$

$$P(n|a) \propto \max(\lambda, \frac{1}{q(d(\phi_a, \phi_n))}) \tag{2.13}$$

The positive sample is a random sample from the minibatch of the same class as the anchor.

Roth et al. [Roth et al., 2020] noticed that for the network to generalise better, it is preferable to have a high number of features with significant variance than a few. As there are shifts in the distributions of the training set and the tests sets, having a few numbers of features with significant variance will lead to a model that fits the training set better but that will generalise less well. For this reason, they propose the following training regularisation for the ranking-based objective: they randomly switch the positive and the negative samples with a probability p_{switch}. This pushes samples of the same class apart, which leads to a model that generates feature vectors with a higher number of features of significant variance.

2.3.2.2 ProxyNCA++

ProxyNCA++ [Teh et al., 2020] is a proxy-based objective function: the model learns a class representative for each class, and pushes the feature vector of a sample towards its corresponding class representative. As the Margin Loss, ProxyNCA++ is used in DML. In Equation (2.14), A is the set of learnable proxies (the class representatives), f_y is the proxy for class y and T is a temperature that is typically set to $\frac{1}{9}$.

$$L_{ProxyNCA++} = -\frac{1}{b}\sum_{i\in\mathcal{B}} \log \frac{\exp\Big(-d(\frac{\phi_i}{\|\phi_i\|_2}, \frac{f_{y_i}}{\|f_{y_i}\|_2})/T\Big)}{\sum_{f_{y_j}\in A}\exp\Big(-d(\frac{\phi_i}{\|\phi_i\|_2}, \frac{f_{y_j}}{\|f_{y_j}\|_2})/T\Big)} \tag{2.14}$$

In the original paper, the authors use three different modifications they found to improve the results: instead of using the commonly used Global Average Pooling before the last layer of the feature extractor, they use Global Max Pooling [Lin et al., 2013]. The maximum value of each feature map is computed, and each maximum value corresponds to one element of the resulting vector. They also use a higher learning rate for the proxies than for the model because the gradient of the proxies is much smaller. Finally, they use layer normalisation [Ba et al., 2016] on top of the Global Max Pooling operation. If the input $\mathbf{a}^l$ of a layer is of size H, it is normalised with the following mean and variance:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \tag{2.15}$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \tag{2.16}$$

2.3.2.3 Normalised Softmax

The normalised softmax [Zhai and Wu, 2018] is another proxy-based objective, where p_i is the proxy for class i. The temperature $0 < \sigma < 1$ is used for exaggerating the differences between classes

$$L_{\text{soft}} = -\log \frac{\exp(\mathbf{x}^\top p_y/\sigma)}{\sum_{z\in Z}\exp(\mathbf{x}^\top p_z/\sigma)} \tag{2.17}$$

2.3.3 Faiss

Faiss [Johnson et al., 2019] is a library used for fast k-nearest neighbours searches. The search can either be real or approximate, and can be performed on the CPU or on the GPU. By default, the search is performed on the CPU and parallelised on all the cores.

Faiss achieves the real k-nearest neighbours search by brute-force search: the distance $\|\mathbf{x}_j - \mathbf{y}_i\|^2$ is expanded to $\|\mathbf{x}_j\|^2 + \|\mathbf{y}_i\|^2 - 2\langle \mathbf{x}_j, \mathbf{y}_i \rangle$, where $\langle ., . \rangle$ is the dot product. The first two terms are precomputed, so the bottleneck is to evaluate $\langle \mathbf{x}_j, \mathbf{y}_i \rangle$.

For the approximate search, Faiss uses following the approximation: $\mathbf{y} \approx q(\mathbf{y}) = q_1(\mathbf{y}) + q_2(\mathbf{y} - q_1(\mathbf{y}))$, where $q_1 : \mathbb{R}^d \to \mathcal{C}_1 \subset \mathbb{R}^d$ and $q_2 : \mathbb{R}^d \to \mathcal{C}_2 \subset \mathbb{R}^d$ are quantizers. q_1 is a coarse quantizer and q_2 is a fine quantizer.

q_1 is trained via k-means, where the number of centroids $|\mathcal{C}_1|$ is typically equal to $\sqrt{l}$ where l is the number of elements in the database. q_2 is a product quantizer: it interprets the vector $\mathbf{y}$ as b subvectors $\mathbf{y} = [\mathbf{y}^0, \mathbf{y}^1, \cdots, \mathbf{y}^{b-1}]$, where b is a divisor of d. The subvectors each possess its own subquantizer, which has 256 centroids to fit in one byte. The resulting quantization is then $q_2(\mathbf{y}) = 256^0 \times q^0(\mathbf{y}^0) + 256^1 \times q^1(\mathbf{y}^1) + \cdots + 256^{b-1} \times q^{b-1}(\mathbf{y}^{b-1})$. Each subquantizer is also trained with k-means.

With this formulation, the nearest neighbours search problem comes down to

$$L_{ADC} = k - \text{argmin}_{i=0:l} \|\mathbf{x} - q(\mathbf{y}_i)\|^2 \tag{2.18}$$

where ADC stands for "Asymmetrical Distance Computation". In this case, the search is not exhaustive in the database: vectors for which the distance is computed are preselected depending on the coarse quantizer

$$L_{IVF} = \tau - \text{argmin}_{\mathbf{c} \in \mathcal{C}_1} \|\mathbf{x} - \mathbf{c}\|^2 \tag{2.19}$$

where τ is the number of centroids to consider. This comes down to an exact nearest neighbour search on the centroids. Then, Johnson et al. compute

$$L_{IVFADC} = \underset{i=0:l \text{ s.t. } q_1(\mathbf{y}_i) \in L_{IVF}}{k - \text{argmin}} \|\mathbf{x} - q(\mathbf{y}_i)\|^2 \tag{2.20}$$

To proceed, they use an inverted file that groups the vectors $\mathbf{y}_i$ into $|\mathcal{C}_1|$ lists, where all the vectors that belong to a list are in the same cluster.

Chapter 3

Implementation of methods for image similarity computation and search

Based on our literature analysis where several promising methods were identified (in Chapter 2), we have decided to implement them into a common open-source library. This chapter presents the two families of methods and their variants that were implemented and tested, including those that did not meet expectations (see chapter 4). The two methodologies are fundamentally different for computing the most similar images. The first one is based on image patches and is inspired by both the randomised vectors technique [Marée et al., 2010] and Yottixel (Figure 3.1), while the second one is more inspired by Smily and Luigi (Figure 3.2). For the second methodology, we tested several variants (including different deep learning architectures) that were not previously tested in this context.

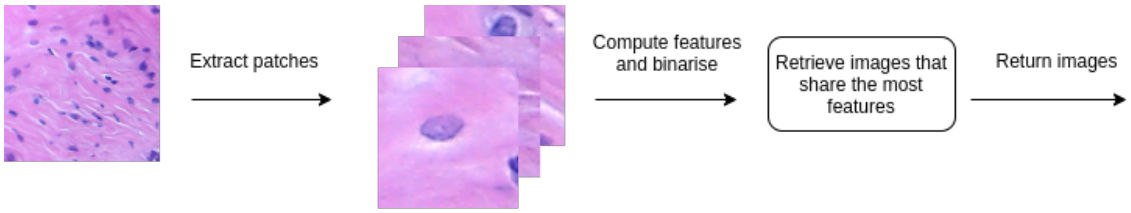

Figure 3.1: Overview of the first methodology. The retrieved images share the most feature vectors with the query image.

Figure 3.2: Overview of the second methodology. The retrieved images are to the nearest feature vectors of the query image feature vector.

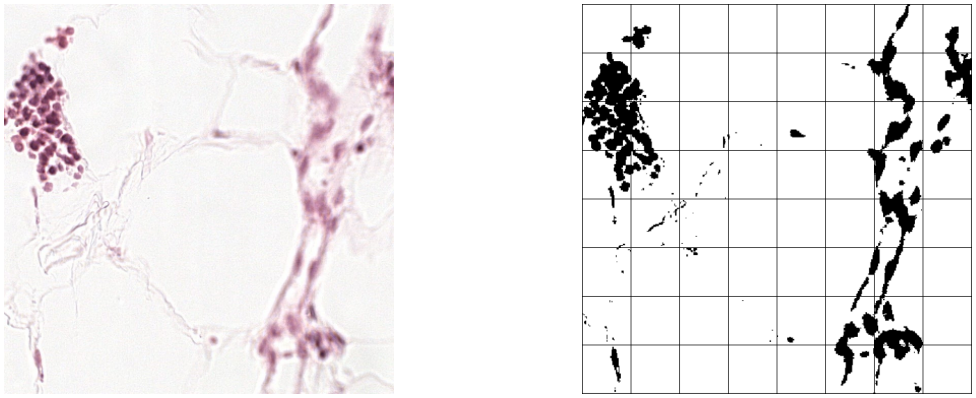

Figure 3.3: Left: histopathology image. Right: thresholded image. A patch that is more than 90% white is considered as not containing tissue.

3.1 First methodology: using a mosaic with Deep Ranking

For this work, it was initially intended to try a method similar to the previous randomised vectors technique (Section 2.2.1) to compute the similarity, but by using a deep learning network instead of randomised vectors.

Yottixel (Section 2.2.3) uses a similar approach to retrieve similar WSIs, and it motivated the choice of this first method.

3.1.1 Mosaic construction

In this section, in order to meet the goals set for this master thesis, a first step is to compare two different approaches: the k-means and the completely random mosaic approaches, to compute the mosaic which is meant to be a representative set of patches of an image.

3.1.1.1 k-means

In this first approach, inspired by Yottixel, the image is first resized to (1952,1952), and this resized image is then divided into 64 patches of size (224,224). In order to only have patches that contain tissue sections, the image is thresholded using Otsu's method. The thresholded patches that are more than 90% white are discarded (Figure 3.3).

If there are less than 8 patches left, these will form the mosaic. Otherwise, more processing is needed.

The histogram of each patch is computed and is then clustered using the k-means algorithm, with 8 clusters. On each of these clusters, k-means is once again applied, where the number of groups is this time set to $0.25\times$(the number of elements in the cluster).

Only one patch from each group is taken to form the mosaic of an image.

In the paper Yottixel, the authors explain that k-means clustering is used instead of random sampling because it is supposed to provide more consistent results. For example, if you ask to retrieve an image that is already available in the database,

it is unlikely that this image will be retrieved when random sampling is used. It is more likely to happen with the k-means extraction method.

A pseudocode for the k-means mosaic construction is presented below in Algorithm 1.

Algorithm 1 mosaic construction with k-means

```
Set k_CH (number of clusters)
Set p_M (percentage of patches to build the mosaic)
Set w_resized (width of resized image, multiple of 224)
procedure BuildMosaic(img)
    grid_size = 224
    nbr_el = w_resized / grid_size
    resized_img = resize(img, (w_resized, w_resized))
    thresholded_img = Otsu(resized_img)
    patch_list = []
    for i in range(nbr_el)*grid_size do
        for j in range(nbr_el)*grid_size do
            if sum(thresholded_img[i:i+grid_size, j:j+grid_size]) < 0.9 × grid_size × grid_size × 255 then
                patch_list.append(resized_img[i:i+grid_size, j:j+grid_size])
            end if
        end for
    end for
    if len(patch_list) <= k_CH then
        return patch_list
    end if
    histograms = []
    for i in patch_list do
        histograms.append(computeHist(i))
    end for
    C_1, C_2, ..., C_{k_CH} = k-means(histograms, k_CH)
    mosaic = []
    for i in range(k_CH) do
        end = p_M × |C_i|
        A_1, ..., A_end = k-means(C_i, end)
        for j in range(end) do
            mosaic.append(element of patch_list corresponding to A_j[0])
        end for
    end for
    return mosaic
end procedure
```

3.1.1.2 Completely random mosaic

In the completely random mosaic approach, the image is first resized to (224,224). A given number of sizes between 16 and 223 are then randomly generated, and random coordinates are generated for each of these sizes. Those random sizes are the

dimensions of patches extracted at those random locations. Those patches compose the mosaic. The procedure is summarised in Algorithm 2.

Algorithm 2 random mosaic construction

```
Set num_patches (number of extracted patches)
procedure BuildMosaic(img)
    resized_img = resize(img, (224, 224))
    sizes = random_int(16, 223, num_patches)
    positions = []
    for s in sizes do
        positions.append(random_int(0, 223 - s)
        positions.append(random_int(0, 223 - s)
    end for
    mosaic = []
    for i in range(num_patches)*2 do
        x, y = positions[i], positions[i+1]
        size = sizes[i/2]
        mosaic.append(img[x:x+size, y:y+size])
    end for
    return mosaic
end procedure
```

3.1.2 Indexation

These two different methods now being developed, the next step is to index images in a database by computing the mosaic of each image. The network used for computing the feature vectors of the mosaic patches is DenseNet-121 trained for deep ranking (DR) on ImageNet, with random horizontal and vertical flipping, random cropping and random hue and saturation shifts. On the one hand, the first three data augmentations are used given the image similarity measure should be invariant to linear transformations. On the other hand, the last two data augmentations are applied because the WSIs might be prepared with different methods, and therefore have different colours: if the hue and the saturation of the training images are slightly modified, this will have less impact on the similarity measure [Tellez et al., 2019]. Examples of those augmentations are shown in Figure 3.4.

The feature vector of each patch of each mosaic is then computed, and binarised with the following rule : $b_i = 1$ if $x_i >$ threshold else 0. These binarised vectors are keys in a hash table where each element is associated to a list of tuples. The tuples consist of two elements : the first one is the name of the image from which the binarised vector has been taken from; the second element is $\frac{1}{\#\text{ elements in the mosaic}}$. A diagram describing the indexation is found in Figure 3.5.

3.1.3 Retrieval

To allow the retrieval of an image, other actions are needed:

- first, the mosaic of the query image is computed;

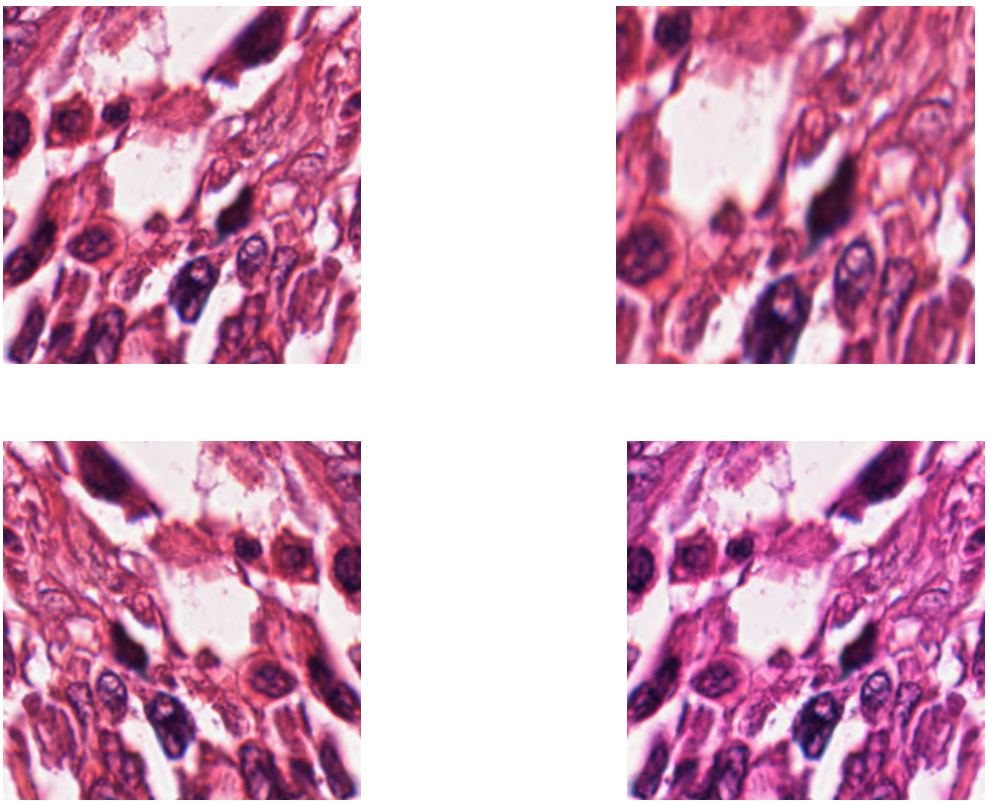

Figure 3.4: Illustration of the data augmentations used during training for the Deep Ranking network used for the mosaic. Top left: initial image. Top right: random crop. Bottom left: random flips. Bottom right: random hue and saturation shifts.

- second, the feature vector of each patch of the mosaic is computed with the same network used during the indexation;
- third, the feature vectors are binarised with the same rule as for the indexation;
- fourth, the binarised vectors are used as keys in the hash table.
- finally, the lists of the hash table corresponding to the keys are looped. For each image name appearing in those lists, a counter is maintained, to which the second member of the tuple is added.

The most similar image is the image that has the largest counter in the end. A diagram describing the retrieval is found in Figure 3.6.

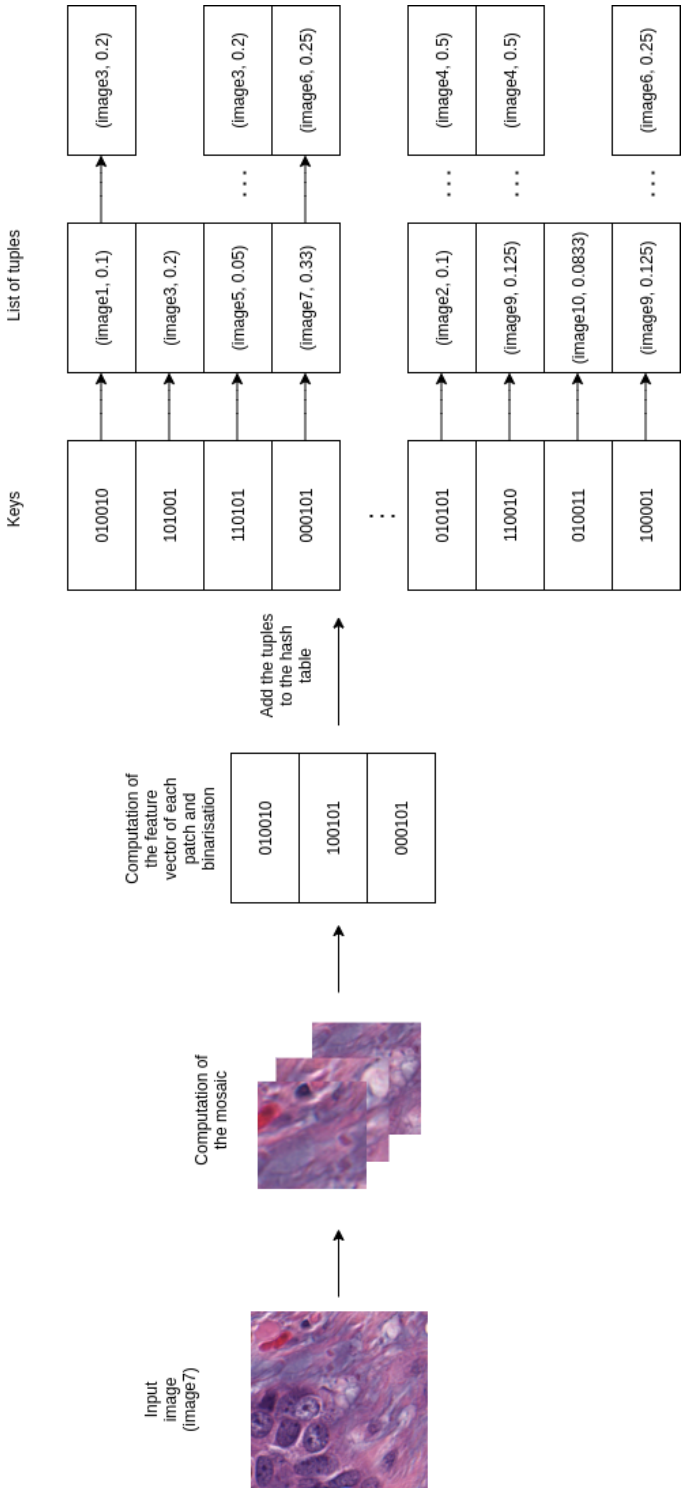

Figure 3.5: Diagram of indexation for the mosaic mathodology

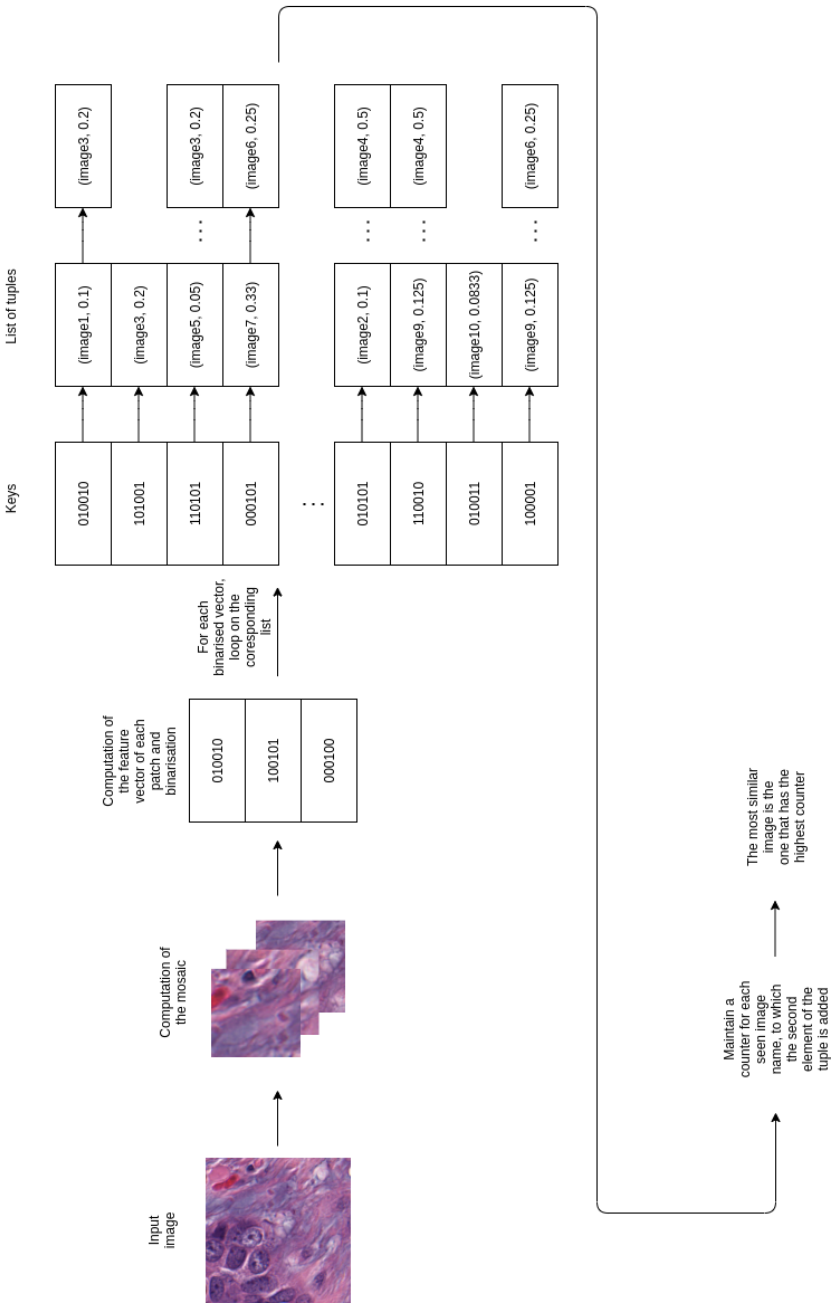

Figure 3.6: Diagram of retrieval for the mosaic methodology

3.2 Second methodology: using Faiss for similarity search

Another way to retrieve similar images is to assign near feature vectors to similar images, without explicit patch extraction but by relying directly on deep learning architectures to generate these feature vectors. Several such methods were tested:

- networks only pretrained on ImageNet;
- classic Deep Ranking (Section 2.3.1);
- Deep Metric Learning:

* Margin loss (Section 2.3.2.1) with different networks;
* ProxyNCA++ (Section 2.3.2.2) with different networks;
* Normalised Softmax loss (Section 2.3.2.3) with different networks.

With all these methods, to retrieve similar images, the nearest neighbours of the query image feature vector must be retrieved. However, a major drawback is that the nearest-neighbor based retrieval can take considerable time when the database expands to millions of objects. So, instead of undertaking an exhaustive search, an alternative solution is to conduct an approximate search of the nearest neighbours, given the similarity measure is not exact. We choose Faiss (Section 2.3.3) for brute-force and approximate search. The primary reason for why Faiss was chosen for similarity search is that a retrieval can be kept quick to carry out, even if it is done at the expanse of accuracy.

3.2.1 Networks pretrained on ImageNet

In this first method, pretrained networks on ImageNet are used as feature extractors. The classification layer of the network is replaced by a linear layer of fixed size with random weights. Such a network can be set as a baseline that a trained network should beat.

Moreover, the untrained DR version of the networks are also tested, where the two shallow convolutional networks and the last linear layer are randomly initialised.

The selected models are ResNet50 (Section 2.1.5), DenseNet-121 (Section 2.1.6) and DeiT (Section 2.1.7).

3.2.2 Classic Deep Ranking

For Deep Ranking (DR), the deep CNN chosen as the ConvNet is DenseNet-121, because it provides better classification results than AlexNet [Krizhevsky et al., 2012], the network used in the original paper.

As already mentioned in Section 2.3.1, sampling the training triplets based on the relevance of the images can be cumbersome, because the training images have to be rated by humans. For this reason, only the random sampling will be considered in this master thesis.

The same data augmentation as for the mosaic is applied (Section 3.1.2).

In the rest of this thesis, ResNet50_DR and DenseNet-121_DR will respectively correspond to ResNet50 and DenseNet-121 where the last layer is replaced by a linear layer of fixed size, and with the two parallel shallow convolutional networks.

3.2.3 Deep Metric Learning

As described in Section 2.3.2, three key components have to be defined to train a Deep Metric Learning (DML) model: the choice of the model, the objective function and the data sampling strategy.

3.2.3.1 The choice of the model

The tested architectures are the following:

- ResNet50 and DenseNet-121 where the last layer is replaced by a linear layer of fixed size;
- ResNet50_DR and DenseNet-121_DR;
- DeiT where the last layer is replaced with a linear layer of fixed size.

The training is tested with both frozen and unfrozen weights for the feature extractor, since the training might be faster by freezing the weights.

Two types of learning rate schedulers are tested:

- the exponential scheduling: $\text{lr} = \text{lr}_{\text{init}} \times \exp(-\gamma \times t)$ where t is the number of the epoch;
- the step scheduling: the initial learning rate is multiplied by γ when half the epochs have been performed.

For training, the same data augmentation as for the mosaic is once again applied (Section 3.1.2).

3.2.3.2 The objective function

The different objective functions used for DML are the following:

- the Margin loss (Section 2.3.2.1);
- the ProxyNCA++ loss (Section 2.3.2.2);
- the Normalised Softmax (Section 2.3.2.3).

3.2.3.3 The minibatch sampling

The "Samples Per Class" [Roth et al., 2020] strategy is used to create informative minibatches. Given a minibatch $\mathcal{B}$ of size b, $\frac{b}{n}$ different classes are randomly selected, from which n different images are selected.

3.2.4 Indexation

To index a dataset in the database, a loop iterates on the folder that contains it. Each image is resized to (224,224) and normalised, and then given to the model. Faiss does not allow the use of string ID's, only numbers can be used: a mapping between the Faiss ID and the image name has to be implemented. The simplest way to do so, is to append the name of the image to a list at the same time the image is added to the Faiss index, such that the first vector in the index corresponds to the first name in the list, the second vector in the index corresponds to the second name in the list, ... The operations are summarised in Figure 3.7.

3.2.5 Retrieval

To retrieve similar images, the input image is resized to (224,224) and normalised, and then fed to the model. The Faiss index is searched for the nearest neighbours, which gives the line number of the nearest neighbours in the index. Eventually, the names in the list at the corresponding line numbers are returned. The operations are summarised in Figure 3.8.

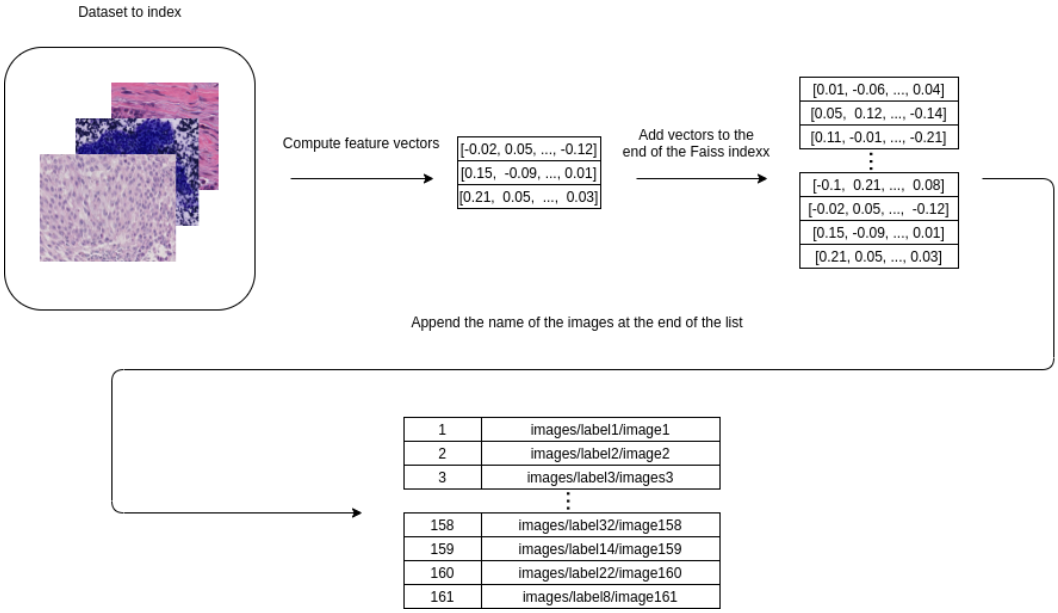

Figure 3.7: Indexing a dataset with Faiss. The feature vector of each image is computed, and added to the Faiss index. At the same time, the name of the images are added to a list, in order to provide a mapping between the Faiss ID and the image name.

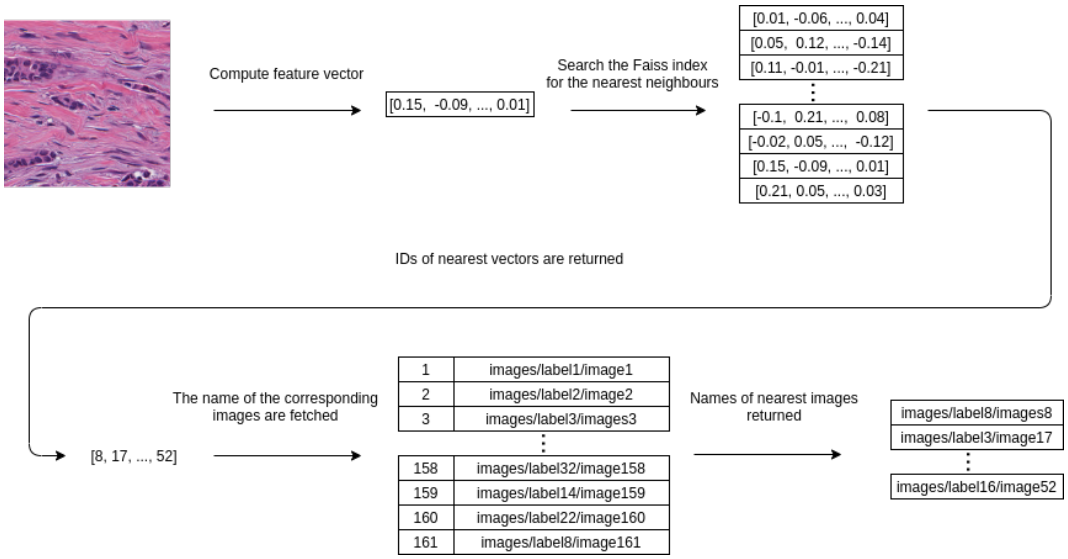

Figure 3.8: Searching similar images with Faiss. The feature vector of the query image is computed. At the same time, the name of the images are added to a list, in order to provide a mapping between the Faiss ID and the image name.

3.3 Implementation details and use of open-source code

The implementation of our open-source library for content-based image retrieval was made possible by the integration of already existing image retrieval and deep learning libraries. Here we describe open source components that were combined for the image similarity measure computations, as well as image indexing and searching. In Chapter 5 we will describe the whole system that allows to execute the whole retrieval process in a distributed fashion through web APIs.

In the first place, the whole code is written in Python and considerably relies on the PyTorch [Paszke et al., 2019] library for the deep learning framework. The pretrained ResNet50 and DenseNet-121 models come from torchvision, which is part of the Pytorch project.

The pretrained DeiT, for its part, comes from the huggingface [Wolf et al., 2020] library.

The mosaic methodology was heavily inspired by Yottixel (Section 2.2.3) and the randomised vectors system (Section 2.2.1). It makes use of the opencv [Itseez, 2015] library to compute the histogram of the patches and read the images, and the scikit-learn [Pedregosa et al., 2011] library for the k-means clustering. The hash table containing the list of tuples is a Redis [Redis, 2009] database for data persistance, and the Python code uses redis-py [McCurdy, 2011] library to communicate with it. Finally, to index a dataset, the code is parallelised on all cores using the joblib [Joblib Development Team, 2020] library.

As to DR, the random triplet sampling uses the following open-source code: [Tejaswi, 2019].

For DML, the Margin loss and its distance weighted sampling strategy, the Normalised Softmax come from [Roth, 2020]. The ProxyNCA++ loss comes from [Teh, 2020]. The "Samples Per Class" strategy is taken from [Roth, 2019].

When using Faiss as a database, the code does not need to explicitly be parallelised as it already is in backend. The code uses the pillow [Clark, 2015] library to read the images. Finally, it uses a Redis database for the list storing the names of the images for data persistance, where the key in the hashtable is the number of indexed images in the database.

The exact parameters that are used by each method will be discussed in their dedicated section in Chapter 4, but at this stage, it is important to point out that the hyperparameters were not tuned, except for the choice of the model, due to a lack of time.

Chapter 4

Image retrieval results

This chapter presents empirical results of the different approaches presented previously for content-based image retrieval on a large histopathological dataset.

First, we describe our datasets. Then, we describe the metric used to determine whether or not a model meets the requirements, and also how this metric is applied. Then, a summary table gives an overview of the results achieved with each evaluated method. Finally, more detailed results are provided for each of them, as well as for the previous randomised method (Section 2.2.1) precedently integrated into the Cytomine web application.

4.1 Datasets description

This section makes a presentation of the datasets used for the training and the tests to compute the accuracy of the retrieval.

The first dataset consists of a dataset of histopathology images, divided in three parts. The methods can be trained on one of these parts. All of them are assessed with the two other parts.

The two other datasets are only used for training. This allows to evaluate the impact of the training data on the results.

4.1.1 Histopathology

Our main dataset was previously used in [Mormont et al., 2020] and was downloaded from `https://dox.uliege.be/index.php/s/7BijhdSac8o9vDP`. It will be later referred to as the histopathology dataset. It is divided into a training set, a test and a validation set. Samples from each class are shown in Figure 4.1. This dataset contains 67 classes, and the training set, the test set and the validation set respectively contain 633,499, 106,357 and 96,192 images. The mean width and height for the images of each class is 230 and 230. The number of images and their frequency per class in the training, test and validation set are respectively presented in Tables 4.1, 4.2 and 4.3. When the names are identical but the last number, it means that they belong to the same original Cytomine project (e.g. an experimental study on a specific organ where images were acquired with the same acquisition protocol). For example, camelyon16_0 and camelyon16_1 are both from Camelyon16 project (`https://camelyon16.grand-challenge.org/`).

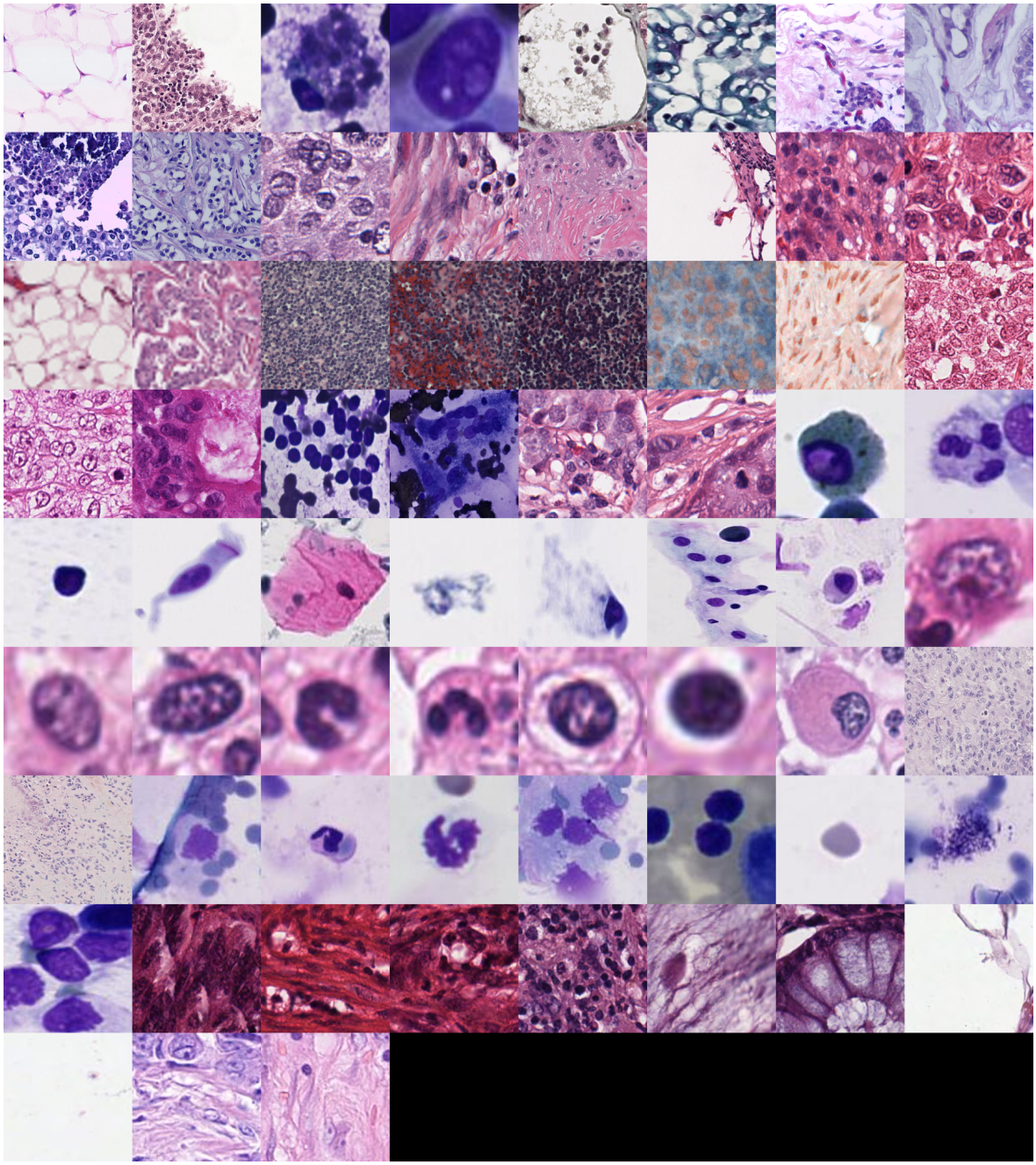

Figure 4.1: A sample for each class of the histopathology training set, including different organs, tissue and cell types, staining and preparation protocols.

Name	# per classs	Frequency
camelyon16_0	218938	0.3456
camelyon16_1	18815	0.0297
cells_no_aug0	1326	0.0021
cells_no_aug1	317	0.0005
glomeruli_no_aug0	9772	0.0154
glomeruli_no_aug1	2385	0.0038
iciar18_micro0	444	0.0007
iciar18_micro1	852	0.0013
iciar18_micro2	648	0.0010
iciar18_micro3	816	0.0013
janowczyk1_0	7051	0.0111
janowczyk1_1	10499	0.0166
janowczyk2_0	437	0.0007
janowczyk2_1	1264	0.0020
janowczyk5_0	12470	0.0197
janowczyk5_1	4090	0.0065
janowczyk6_0	160790	0.2538
janowczyk6_1	64032	0.1011
janowczyk7_0	408	0.0006
janowczyk7_1	504	0.0008
janowczyk7_2	438	0.0007
lbpstroma_0	650	0.0010
lbpstroma_1	297	0.0005
mitos2014_0	2645	0.0042
mitos2014_1	9150	0.0144
mitos2014_2	28569	0.0451
patterns_no_aug_0	714	0.0011
patterns_no_aug_1	465	0.0007
tupac_mitosis0	50114	0.0791
tupac_mitosis1	12760	0.0201
ulb_anapath_lba0	622	0.0010
ulb_anapath_lba1	939	0.0015
ulb_anapath_lba2	234	0.0004
ulb_anapath_lba3	322	0.0005

Name	# per class	Frequency
ulb_anapath_lba4	590	0.0009
ulb_anapath_lba5	686	0.0011
ulb_anapath_lba6	163	0.0003
ulb_anapath_lba7	64	0.0001
ulb_anapath_lba8	431	0.0007
ulg_bonemarrow0	28	0.0000
ulg_bonemarrow1	39	0.0001
ulg_bonemarrow2	28	0.0000
ulg_bonemarrow3	36	0.0001
ulg_bonemarrow4	78	0.0001
ulg_bonemarrow5	145	0.0002
ulg_bonemarrow6	83	0.0001
ulg_bonemarrow7	85	0.0001
ulg_lbtd2_chimio_necrose0	275	0.0004
ulg_lbtd2_chimio_necrose1	420	0.0007
ulg_lbtd_lba0	323	0.0005
ulg_lbtd_lba1	360	0.0006
ulg_lbtd_lba2	450	0.0007
ulg_lbtd_lba3	101	0.0002
ulg_lbtd_lba4	58	0.0001
ulg_lbtd_lba5	179	0.0003
ulg_lbtd_lba6	216	0.0003
ulg_lbtd_lba7	35	0.0001
umcm_colorectal_01	408	0.0006
umcm_colorectal_02	355	0.0006
umcm_colorectal_03	395	0.0006
umcm_colorectal_04	494	0.0008
umcm_colorectal_05	288	0.0005
umcm_colorectal_06	426	0.0007
umcm_colorectal_07	393	0.0006
umcm_colorectal_08	590	0.0009
warwick_crc0	848	0.0013
warwick_crc1	652	0.0010

Table 4.1: Number of images and their frequency per class in the training histopathology set.

Name	# per class	Frequency
camelyon16_0	23810	0.2240
camelyon16_1	2713	0.0255
cells_no_aug0	1452	0.0137
cells_no_aug1	369	0.0035
glomeruli_no_aug0	11720	0.1103
glomeruli_no_aug1	2888	0.0272
iciar18_micro0	504	0.0047
iciar18_micro1	228	0.0021
iciar18_micro2	312	0.0029
iciar18_micro3	276	0.0026
janowczyk1_0	4006	0.0377
janowczyk1_1	5669	0.0533
janowczyk2_0	434	0.0041
janowczyk2_1	862	0.0081
janowczyk5_0	3209	0.0302
janowczyk5_1	550	0.0052
janowczyk6_0	15089	0.1420
janowczyk6_1	5679	0.0534
janowczyk7_0	138	0.0013
janowczyk7_1	156	0.0015
janowczyk7_2	144	0.0014
lbpstroma_0	639	0.0060
lbpstroma_1	320	0.0030
mitos2014_0	1052	0.0099
mitos2014_1	2680	0.0252
mitos2014_2	7978	0.0751
patterns_no_aug_0	289	0.0027
patterns_no_aug_1	222	0.0021
tupac_mitosis0	5152	0.0485
tupac_mitosis1	2000	0.0188
ulb_anapath_lba0	95	0.0009
ulb_anapath_lba1	523	0.0049
ulb_anapath_lba2	61	0.0006
ulb_anapath_lba3	161	0.0015

Name	# per class	Frequency
ulb_anapath_lba4	29	0.0003
ulb_anapath_lba5	47	0.0004
ulb_anapath_lba6	57	0.0005
ulb_anapath_lba7	24	0.0002
ulb_anapath_lba8	26	0.0002
ulg_bonemarrow0	31	0.0003
ulg_bonemarrow1	49	0.0005
ulg_bonemarrow2	30	0.0003
ulg_bonemarrow3	47	0.0004
ulg_bonemarrow4	88	0.0008
ulg_bonemarrow5	185	0.0017
ulg_bonemarrow6	108	0.0010
ulg_bonemarrow7	101	0.0010
ulg_lbtd2_chimio_necrose0	26	0.0002
ulg_lbtd2_chimio_necrose1	65	0.0006
ulg_lbtd_lba0	207	0.0019
ulg_lbtd_lba1	557	0.0052
ulg_lbtd_lba2	228	0.0021
ulg_lbtd_lba3	365	0.0034
ulg_lbtd_lba4	61	0.0006
ulg_lbtd_lba5	244	0.0023
ulg_lbtd_lba6	153	0.0014
ulg_lbtd_lba7	31	0.0003
umcm_colorectal_01	217	0.0020
umcm_colorectal_02	270	0.0025
umcm_colorectal_03	230	0.0022
umcm_colorectal_04	131	0.0012
umcm_colorectal_05	337	0.0032
umcm_colorectal_06	199	0.0019
umcm_colorectal_07	232	0.0022
umcm_colorectal_08	35	0.0003
warwick_crc0	283	0.0027
warwick_crc1	217	0.0020

Table 4.2: Number of images and their frequency per class in the test histopathology set.

Name	# per class	Frequency
camelyon16_0	25697	0.2673
camelyon16_1	2253	0.0234
cells_no_aug0	121	0.0013
cells_no_aug1	52	0.0005
glomeruli_no_aug0	1941	0.0202
glomeruli_no_aug1	507	0.0053
iciar18_micro0	252	0.0026
iciar18_micro1	120	0.0012
iciar18_micro2	240	0.0025
iciar18_micro3	108	0.0011
janowczyk1_0	1806	0.0188
janowczyk1_1	2694	0.0280
janowczyk2_0	105	0.0011
janowczyk2_1	300	0.0031
janowczyk5_0	3691	0.0384
janowczyk5_1	860	0.0089
janowczyk6_0	22859	0.2378
janowczyk6_1	9075	0.0944
janowczyk7_0	132	0.0014
janowczyk7_1	174	0.0018
janowczyk7_2	150	0.0016
lbpstroma_0	279	0.0029
lbpstroma_1	128	0.0013
mitos2014_0	948	0.0099
mitos2014_1	3190	0.0332
mitos2014_2	8661	0.0901
patterns_no_aug_0	94	0.0010
patterns_no_aug_1	73	0.0008
tupac_mitosis0	6267	0.0652
tupac_mitosis1	1560	0.0162
ulb_anapath_lba0	139	0.0014
ulb_anapath_lba1	41	0.0004
ulb_anapath_lba2	13	0.0001
ulb_anapath_lba3	92	0.0010
ulb_anapath_lba4	17	0.0002
ulb_anapath_lba5	2	0.0000
ulb_anapath_lba6	24	0.0002
ulb_anapath_lba7	8	0.0001
ulb_anapath_lba8	10	0.0001
ulg_bonemarrow0	8	0.0001
ulg_bonemarrow1	9	0.0001
ulg_bonemarrow2	8	0.0001
ulg_bonemarrow3	8	0.0001
ulg_bonemarrow4	18	0.0002
ulg_bonemarrow5	36	0.0004
ulg_bonemarrow6	20	0.0002
ulg_bonemarrow7	23	0.0002
ulg_lbtd2_chimio_necrose0	14	0.0001
ulg_lbtd2_chimio_necrose1	82	0.0009
ulg_lbtd_lba0	207	0.0022
ulg_lbtd_lba1	95	0.0010
ulg_lbtd_lba2	163	0.0017
ulg_lbtd_lba3	54	0.0006
ulg_lbtd_lba4	3	0.0000
ulg_lbtd_lba5	64	0.0007
ulg_lbtd_lba6	98	0.0010
ulg_lbtd_lba7	32	0.0003
umcm_colorectal_01	0	0.0000
umcm_colorectal_02	0	0.0000
umcm_colorectal_03	0	0.0000
umcm_colorectal_04	0	0.0000
umcm_colorectal_05	0	0.0000
umcm_colorectal_06	0	0.0000
umcm_colorectal_07	0	0.0000
umcm_colorectal_08	0	0.0000
warwick_crc0	284	0.0030
warwick_crc1	216	0.0022

Table 4.3: Number of images and their frequency per class in the validation histopathology set.

Figure 4.2: Some samples from the INaturalist training set

4.1.2 INaturalist and ImageNet training sets

The Smily architecture (Section 2.2.2) is trained on natural images. For this thesis, the same kind of training datasets were looked for, but it goes without saying that those datasets contain far less images.

The INaturalist dataset of 2021[1] is the first that was used. It is made of natural images, such as plants, mushrooms, insects, birds,... The training dataset contains almost 2.7M images from 10,000 different classes. It was specially made for fine-grained image classification, which could prove to be useful for image retrieval. A small sample of the INaturalist training dataset is shown in Figure 4.2.

The second dataset is the well-known ImageNet-2012[2]. It is made of 1.2M natural images from 1,000 different classes.

4.2 Evalutation protocol

The metric used for the evaluation is the top-k accuracy:

$$\text{top-}k = \frac{\sum_{\mathbf{x}_i \in \mathcal{X}_{\text{test}}} \mathbb{I}(\exists \mathbf{x}_i \in \mathcal{F} \text{ s.t. } y_i = y_q)}{|\mathcal{X}_{\text{test}}|}, \qquad \mathcal{F} = \underset{|\mathcal{F}|=k}{\text{argmin}}\, d(\phi(\mathbf{x}_i), \phi(\mathbf{x}_q)) \tag{4.1}$$

When $k = 1$, the image retrieval problem comes down to a classification problem, where the predicted class of the input image is the class of the retrieved image.

In our protocol, the learning set is only used to train methods that require learning. To compute the accuracy, the test dataset is indexed in the database, and the validation set is used to perform image retrieval queries. The problem is that some classes are a lot more represented in both the test set and the validation set (camelyon16_0 and janowczyk6_0 e.g.): it would be misleading to compute the accuracy with the whole validation set, simply because images from more represented classes will have a higher chance of being retrieved. And since they are more represented in

[1] `https://github.com/visipedia/inat_comp/tree/master/2021`
[2] `https://www.image-net.org/download.php`

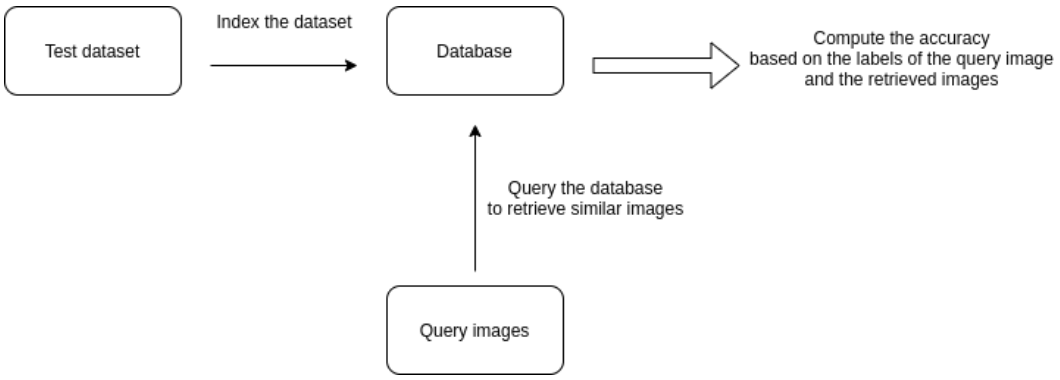

Figure 4.3: Proposed evaluation protocol for each tested method.

both sets, the correct predictions of those classes will completely drown the wrong predictions of underrepresented classes. To correct this drawback, we created an alternative validation set where the same number of images per class are randomly selected from the original validation set when possible (some of them do not have images at all, or just a few). In total, only 1,000 images from the original validation set were selected, as quite a significant number of classes have less than 15 images per class. Such a small number of images could seem surprising, but it allows to better perceive at first glance if a model is just fair or excellent. This protocol choice will be compared to two other more classic evaluation protocols in Section 4.10.

A figure explaining the three evaluation protocols is shown in Figure 4.3.

Concerning the computation time, each methodology is evaluated on the same computer, that has the following configuration:

- CPU: AMD Ryzen 9 3900X of 12 cores (3.8 GHz);
- GPU: Nvidia RTX 3070;
- 32GB of RAM.

For the methodology with Faiss, the images are indexed in the database by batches of 128 using Pytorch Dataloader with 12 workers.

4.3 Overall results

This section presents the best results obtained with each method in table form (Table 4.4). Those results will be discussed in more detail in their dedicated section. In this table, "frozen" means that only the weights of the feature extractor are frozen during training: the weights of the last linear layer and possibly the two parallel shallow convolutional layers are updated during the training.

4.4 Results with Randomised Trees

The previous system used on Cytomine was based on randomised vectors (Section 2.2.1). This method, when it was developed, was not evaluated on a large histopathology dataset. Therefore, in this subsection, we provide baseline results with this method, see Sable 4.5 and 4.6.

Method	Training Dataset	top-1 accuracy	top-5 accuracy
Randomised vectors technique (1,000 random patches, 5 test vectors)	/	24%	44%
Mosaic with DenseNet-121_DR (k-means, feature vector of size 128)	ImageNet-2012	14%	36%
Pretrained ResNet50	/	45%	69%
Pretrained ResNet_DR	/	48%	72%
Pretrained DenseNet-121	/	44%	69%
Pretrained DenseNet-121_DR	/	49%	73%
Pretrained DeiT	/	54%	76%
Deep Ranking with DenseNet-121_DR (frozen weights, 5 epochs)	histopathology	40%	68%
Deep Ranking with DenseNet-121_DR (frozen weights, 5 epochs)	ImageNet-2012	35%	60%
Margin Loss with ResNet50 (5 epochs, frozen weights)	histopathology	53%	79%
Margin Loss with ResNet50_DR (5 epochs, frozen weights)	histopathology	57%	81%
Margin Loss with DenseNet-121_DR (5 epochs, frozen weights)	histopathology	61%	84%
Margin Loss with DenseNet-121_DR (20 epochs, frozen weights, exp scheduling)	histopathology	64%	84%
Margin Loss with DenseNet-121_DR (20 epochs, frozen weights, step scheduling)	histopathology	63%	84%
ProxyNCA++ with DenseNet-121 (5 epochs, frozen weights)	histopathology	52%	78%
Margin Loss with ResNet50 (50 epochs, exp scheduling)	histopathology	77%	89%
Margin Loss with ResNet50_DR (50 epochs, exp scheduling)	histopathology	77%	90%
MarginLoss with DenseNet-121 (50 epochs, exp scheduling)	histopathology	78%	89%
Margin Loss with DenseNet-121_DR (50 epochs, exp scheduling)	histopathology	78%	90%
Margin Loss with DeiT (50 epochs, exp scheduling)	histopathology	53%	79%
Margin Loss with DeiT (14 epochs, step scheduling)	INaturalist	32%	56%
Margin Loss with DenseNet-121 (11 epochs, step scheduling) did not have time to finish training	INaturalist	50%	75%
ProxyNCA++ with DenseNet121 (30 epochs, step scheduling)	Imagenet-2012	46%	69%
MarginLoss with DenseNet-121 (30 epochs, exp scheduling)	ImageNet-2012	44%	68%

Table 4.4: Summary of the accuracies obtained with the different tested methods. For pretrained, see section 3.2.1. For *_DR, see Section 3.2.2. For the scheduling, see Section 3.2.3.1. Computed with 1,000 queries from the validation set.

Number of random patches	top-1	top-5	Time per request (s)
100	17%	40%	3
500	21%	40%	8
1000	24%	44%	16

Table 4.5: Results and time per request of the randomised vectors technique, by modifying the number of random patches extracted per image. Computed with 1,000 queries from the validation set.

Number of test vectors	top-1	top-5	Time per request (s)
2	14%	31%	5
5	24%	44%	16
10	20%	37%	22

Table 4.6: Results and time per request of the randomised vectors technique, by modifying the number of test vectors. Notice that with 10 vectors, it was impossible to index the whole test dataset on a computer with 32GB of memory, only 70,000 images were effectively indexed. Computed with 1,000 queries from the validation set.

4.5 Results with the mosaic

The results obtained with the methodology based on the mosaic construction are discussed in this section.

DenseNet-121_DR was trained on ImageNet for 5 epochs, with frozen weights for DenseNet-121 (except for the replaced last linear layer) and a batch size of 32. The Adam optimiser was used during training, with a learning rate of 0.01, which is divided by 2 at the fourth epoch. Those values were set in an arbitrary manner, and the weights of DenseNet-121 were frozen.

The accuracy on the test dataset used for indexation is computed with a feature vector size of 32. Figure 4.4 shows the accuracy on the test dataset for different sizes of feature vectors. Here, the k-means extraction method was used to compute the mosaic, and a threshold of 0.5 was used for the binarisation, but in retrospect, this threshold value was probably set way too high, and this might be the cause of the poor results of this method. In this graph, the hit rate is the number of keys in the hash table that were used to find the similar images over the size of the mosaic.

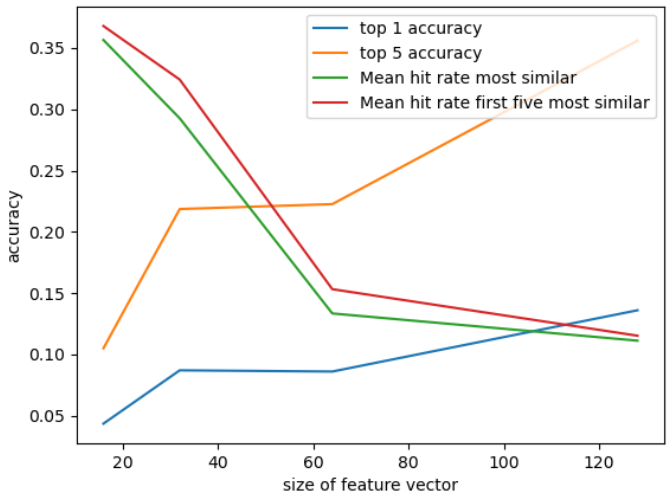

Figure 4.4: Accuracy on the test dataset with the mosaic using k-means extraction. Comptuted with 1,000 queries from the valitation set.

4.6 Results with Classic Deep Ranking

DenseNet-121_DR is trained on the histopathology dataset for 5 epochs, with frozen weights for DensNet-121 (except for the last linear layer), and a batch size of 32. Once again, the Adam optimiser is used with an initial learning rate of 0.01, where the learning rate is divided by 2 at the fourth epoch. The feature vector size is set to 128, and the weights of DenseNet-121 are frozen.

With DR, on the test dataset, a top-1 accuracy of 40 % and a top-5 accuracy of 68 % are obtained, which is less than the pretrained DenseNet-121_DR (Table 4.4). There are two reasons for those poorer results: first the random sampling for the triplets, and second, the fact that some images from the same class do not look visually alike in the training set. Using the "golden feature" (Section 2.3.1 for definition) would help alleviate this problem, but since there is no other training set to train the golden feature at disposal, another method less sensitive to images of the same class not visually looking alike will have to be used. Tables 4.7 and 4.8 are produced when the test set is indexed and with the validation set is used for the queries, with the trained DR model on the histopathology dataset, and Faiss used in brute force. For indexation and retrieval, it takes the same time as for DML when DenseNet-121_DR is used as the network (section 4.7).

4.7 Results with DML

All the models were trained with a feature vector size of 128 (as in Smily), the Adam optimiser, a batch size of 128 and $\gamma = 0.3$ [Roth et al., 2020] if learning rate scheduling is used. Once again, the hyperparameters were not optimised. Unless mentioned otherwise, the hyperparameters used were, for each different loss:

- Margin Loss: a learning rate of 10^{-4} and a weight-decay of 4×10^{-4} for the model, and a learning rate of 5×10^{-4} for the learnable boundary β, as suggested by [Roth et al., 2020].

Name	Precision	Name	Precision	Name	Precision
camelyon16_0	1.0	janowczyk7_1	0.58	ulg_bonemarrow1	0.22
camelyon16_1	0.26	janowczyk7_2	0.26	ulg_bonemarrow3	0.12
cells_no_aug0	0.63	lbpstroma_0	0.68	ulg_bonemarrow4	0.28
cells_no_aug1	0.11	lbpstroma_1	0.58	ulg_bonemarrow5	0.42
glomeruli_no_aug0	0.84	mitos2014_0	0.11	ulg_bonemarrow6	0.26
glomeruli_no_aug1	0.47	mitos2014_1	0.16	ulg_bonemarrow7	0.47
iciar18_micro0	0.37	mitos2014_2	0.37	ulg_lbtd2_chimio_necrose0	0.71
iciar18_micro2	0.32	patterns_no_aug_0	0.32	ulg_lbtd2_chimio_necrose1	0.37
iciar18_micro3	0.16	patterns_no_aug_1	0.58	ulg_lbtd_lba0	0.11
janowczyk1_0	0.16	tupac_mitosis0	0.26	ulg_lbtd_lba1	0.21
janowczyk1_1	0.21	tupac_mitosis1	0.05	ulg_lbtd_lba2	0.26
janowczyk2_0	0.21	ulb_anapath_lba0	0.11	ulg_lbtd_lba3	0.11
janowczyk2_1	0.32	ulb_anapath_lba1	0.42	ulg_lbtd_lba5	0.26
janowczyk6_0	0.89	ulb_anapath_lba2	0.15	ulg_lbtd_lba6	0.05
janowczyk6_1	0.63	ulb_anapath_lba3	0.21	ulg_lbtd_lba7	0.05
janowczyk7_0	0.37	ulb_anapath_lba4	0.06	warwick_crc0	0.26
janowczyk7_1	0.58	ulb_anapath_lba8	0.1	warwick_crc1	0.21

Table 4.7: Top-1 precision per class with DR on test dataset, Faiss used in brute force. Computed with 1,000 queries from the validation set.

Name	Precision	Name	Precision	Name	Precision
camelyon16_0	1.0	janowczyk7_1	0.95	ulg_bonemarrow1	0.44
camelyon16_1	0.53	janowczyk7_2	0.63	ulg_bonemarrow2	0.38
cells_no_aug0	0.95	lbpstroma_0	0.84	ulg_bonemarrow3	0.25
cells_no_aug1	0.58	lbpstroma_1	1.0	ulg_bonemarrow4	0.67
glomeruli_no_aug0	1.0	mitos2014_0	0.47	ulg_bonemarrow5	1.0
glomeruli_no_aug1	0.84	mitos2014_1	0.79	ulg_bonemarrow6	0.79
iciar18_micro0	0.89	mitos2014_2	0.95	ulg_bonemarrow7	0.74
iciar18_micro1	0.58	patterns_no_aug_0	0.63	ulg_lbtd2_chimio_necrose0	0.86
iciar18_micro2	0.58	patterns_no_aug_1	0.89	ulg_lbtd2_chimio_necrose1	0.63
iciar18_micro3	0.68	tupac_mitosis0	0.63	ulg_lbtd_lba0	0.42
janowczyk1_0	0.47	tupac_mitosis1	0.26	ulg_lbtd_lba1	0.79
janowczyk1_1	0.68	ulb_anapath_lba0	0.16	ulg_lbtd_lba2	0.68
janowczyk2_0	0.63	ulb_anapath_lba1	0.89	ulg_lbtd_lba3	0.47
janowczyk2_1	0.53	ulb_anapath_lba2	0.62	ulg_lbtd_lba5	0.63
janowczyk5_0	0.37	ulb_anapath_lba3	0.63	ulg_lbtd_lba6	0.16
janowczyk5_1	0.16	ulb_anapath_lba4	0.24	ulg_lbtd_lba7	0.21
janowczyk6_0	1.0	ulb_anapath_lba6	0.26	warwick_crc0	0.42
janowczyk6_1	0.89	ulb_anapath_lba8	0.2	warwick_crc1	0.53
janowczyk7_0	0.47	ulg_bonemarrow0	0.25		

Table 4.8: Top-5 precision per class with DR on test dataset, Faiss used in brute force. Computed with 1,000 queries from the validation set.

Model	top-1 accuracy	top-5 accuracy
ResNet50	77%	89%
ResNet50_DR	77%	90%
DenseNet-121	78%	89%
DenseNet-121_DR	78%	90%
DeiT	53%	79%

Table 4.9: Results for models trained with unfrozen weights with the Margin Loss on histopathology test dataset. Computed with 1,000 queries from the validation set.

Model	Indexation time	Time spent in model per request	Time spent in search per request
ResNet50_	3 min 56 s	6.42 ms	4.36 ms
ResNet50_DR	3 min 48 s	7.48 ms	4.37 ms
DenseNet-121	4 min 25 s	15.3 ms	4.44 ms
DenseNet-121_DR	4 min 40 s	15.5 ms	4.46 ms
DeiT	7 mins 43 s	16.75 ms	4.41 ms

Table 4.10: Retrieval time for different models, 100,000 images indexed in the database, Faiss used in brute force on CPU.

- ProxyNCA++: a learning rate of 4×10^{-3} for the model and of 4×10^{2} for the proxies, with no weight decay, as suggested by [Teh et al., 2020].
- Normalised Softmax: a learning rate of 10^{-4} and a weight-decay of 4×10^{-4} for the model, and a learning rate of 10^{-5} for the proxies, as suggested by [Roth et al., 2020].

This section will only present the results obtained without frozen weights for the network, as these were the best obtained results. Still, all the results obtained are reported in Table 4.4.

For the histopathology dataset, Table 4.9 regroups the results. All those models were trained with exponential scheduling and the Margin Loss for 50 epochs. The best models for which the best results are obtained is DenseNet-121_DR, but ResNet50_DR is only very slightly behind.

On the INaturalist and ImageNet-2012 datasets, the results were lower than those obtained with the models only pretrained on Imagenet for both the Margin Loss and ProxyNCA++. This is not the case when DenseNet-121 is trained with the Margin Loss on the INaturalist, but still, the results obtained lag far behind those obtained when the network is trained on the histopathology dataset, even after 11 epochs. By comparison, a top-1 accuracy of 77% and a top-5 accuracy of 88% are obtained by training DenseNet-121 with the Margin Loss on the histopathology dataset for 11 epochs.

The two best models are therefore ResNet50_DR and DenseNet-121_DR. But by analysing the search time for each models (Table 4.10), it can be observed that the search time for ResNet50_DR is substantially faster. For this reason, this model will be the final model retained for similarity search. Tables 4.11 and 4.12 summarise the top-1 and top-5 accuracies per class. Those tables show that some underrepresented classes in the training, testing, and validation sets are less well retrieved, e.g. ulb_anapath_lba8.

In order to analyse which classes are retrieved for each tested class, it is possible to plot a confusion matrix. In Section 4.2, it was already mentioned that when computing a top-1 accuracy, the image retrieval problem came down to a classification

Name	Precision	Name	Precision	Name	Precision
camelyon16_0	1.0	janowczyk7_1	1.0	ulg_bonemarrow1	0.89
camelyon16_1	0.74	janowczyk7_2	0.89	ulg_bonemarrow2	1.0
cells_no_aug0	0.89	lbpstroma_0	1.0	ulg_bonemarrow3	0.88
cells_no_aug1	0.89	lbpstroma_1	1.0	ulg_bonemarrow4	0.94
glomeruli_no_aug0	1.0	mitos2014_0	0.05	ulg_bonemarrow5	0.84
glomeruli_no_aug1	0.89	mitos2014_1	0.63	ulg_bonemarrow6	0.95
iciar18_micro0	0.63	mitos2014_2	0.74	ulg_bonemarrow7	1.0
iciar18_micro1	0.89	patterns_no_aug_0	1.0	ulg_lbtd2_chimio_necrose0	1.0
iciar18_micro2	0.42	patterns_no_aug_1	0.37	ulg_lbtd2_chimio_necrose1	1.0
iciar18_micro3	0.58	tupac_mitosis0	0.68	ulg_lbtd_lba0	0.89
janowczyk1_0	0.68	tupac_mitosis1	0.32	ulg_lbtd_lba1	0.74
janowczyk1_1	0.79	ulb_anapath_lba0	0.84	ulg_lbtd_lba2	1.0
janowczyk2_0	0.95	ulb_anapath_lba1	1.0	ulg_lbtd_lba3	0.63
janowczyk2_1	0.79	ulb_anapath_lba2	0.77	ulg_lbtd_lba4	0.33
janowczyk5_0	0.37	ulb_anapath_lba3	0.84	ulg_lbtd_lba5	0.84
janowczyk5_1	0.16	ulb_anapath_lba4	0.94	ulg_lbtd_lba6	0.32
janowczyk6_0	0.95	ulb_anapath_lba6	0.63	ulg_lbtd_lba7	0.58
janowczyk6_1	0.95	ulb_anapath_lba7	0.62	warwick_crc0	0.79
janowczyk7_0	0.95	ulg_bonemarrow0	0.75	warwick_crc1	0.68

Table 4.11: Top-1 precision per class on test dataset with ResNet50_DR trained with the Margin Loss, Faiss used in brute force. Computed with 1,000 queries from the validation set.

Name	Precision	Name	Precision	Name	Precision
camelyon16_0	1.0	janowczyk7_2	0.95	ulg_bonemarrow1	1.0
camelyon16_1	0.95	lbpstroma_0	1.0	ulg_bonemarrow2	1.0
cells_no_aug0	1.0	lbpstroma_1	1.0	ulg_bonemarrow3	1.0
cells_no_aug1	1.0	mitos2014_0	0.42	ulg_bonemarrow4	0.94
glomeruli_no_aug0	1.0	mitos2014_1	0.84	ulg_bonemarrow5	0.95
glomeruli_no_aug1	0.95	mitos2014_2	0.89	ulg_bonemarrow6	0.95
iciar18_micro0	0.79	patterns_no_aug_0	1.0	ulg_bonemarrow7	1.0
iciar18_micro1	0.95	patterns_no_aug_1	0.68	ulg_lbtd2_chimio_necrose0	1.0
iciar18_micro2	0.58	tupac_mitosis0	1.0	ulg_lbtd2_chimio_necrose1	1.0
iciar18_micro3	0.79	tupac_mitosis1	0.63	ulg_lbtd_lba0	0.95
janowczyk1_0	1.0	ulb_anapath_lba0	0.89	ulg_lbtd_lba1	0.89
janowczyk1_1	1.0	ulb_anapath_lba1	1.0	ulg_lbtd_lba2	1.0
janowczyk2_0	1.0	ulb_anapath_lba2	1.0	ulg_lbtd_lba3	0.79
janowczyk2_1	1.0	ulb_anapath_lba3	1.0	ulg_lbtd_lba4	0.67
janowczyk5_0	0.74	ulb_anapath_lba4	0.94	ulg_lbtd_lba5	0.95
janowczyk5_1	0.58	ulb_anapath_lba5	0.5	ulg_lbtd_lba6	0.47
janowczyk6_0	1.0	ulb_anapath_lba6	0.68	ulg_lbtd_lba7	0.79
janowczyk6_1	1.0	ulb_anapath_lba7	0.62	warwick_crc0	0.84
janowczyk7_0	1.0	ulb_anapath_lba8	0.1	warwick_crc1	0.95
janowczyk7_1	1.0	ulg_bonemarrow0	0.88		

Table 4.12: Top-5 precision per class on test dataset with ResNet50_DR trained with the Margin Loss, Faiss used in brute force. Computed with 1,000 queries from the validation set.

problem, where the predicted class is the class of the retrieved image. Therefore, it seems logical to plot a confusion matrix. This confusion matrix is plotted when the training set is indexed in the database, and the test set is used to query the database. Those datasets are used because the validation set does not contain images for some classes, and the confusion matrix would be less informative in that case. It is shown in Figure 4.5, and the ordinate is the true class, and the abscissa is the predicted class. It took 24 minutes to index the training dataset and 26 s to compute the 1,000 queries. The time spent in the Faiss searches is 19.5 s, that is 0.02 s for each query. It is an indication that Faiss should be used in approximate search. The impact of the approximate search on the search time and the precision will be studied in Section 4.9.

As a general remark, it must be said that most mislabelled images are mislabelled with classes from the same Cytomine project: images from ulg_lbtd_lba are only mislabelled with images from this project, as well as images from ulg_bonnemarrow are only mislabelled with images from ulg_bonemarrow. Since they belong to the same project, those images have the same visual look (Figure 4.6).

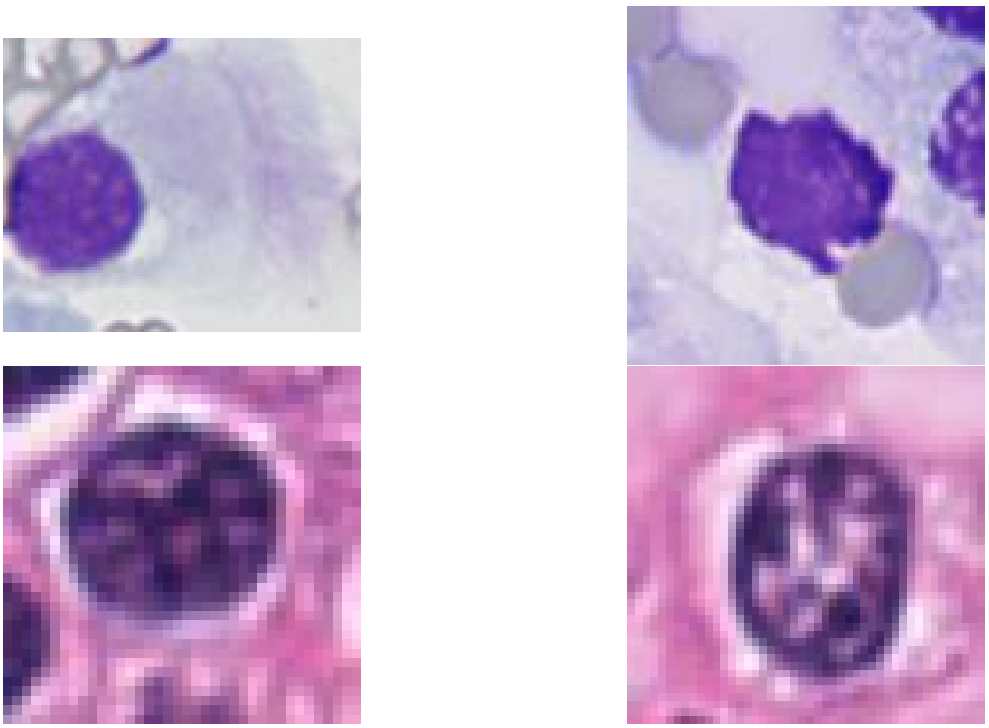

Figure 4.6: Top: two images from ulg_lbtd_lba from different sub-classes. Bottom: two images from ulg_bonnemarrow from different sub-classes.

But some classes, as ulb_anapath_lba7, are not only mislabelled with classes from their same Cytomine project. This particular class is also mislabelled with glomeruli_no_aug (first row of Figure 4.7). The first retrieved image does not look alike at all, except for the turquoise borders. For the other mislabelled classes with classes from other projects, the retrieved images do look alike (rest of Figure 4.7). Those other mislabelled classes are:

- janowczyck_5 with tupac_mitosis, that are both relatively well represented;
- ulg_lbtd_lba7 with patterns_no_aug_0, patterns_no_aug_0 is 10 times more represented in the training set;

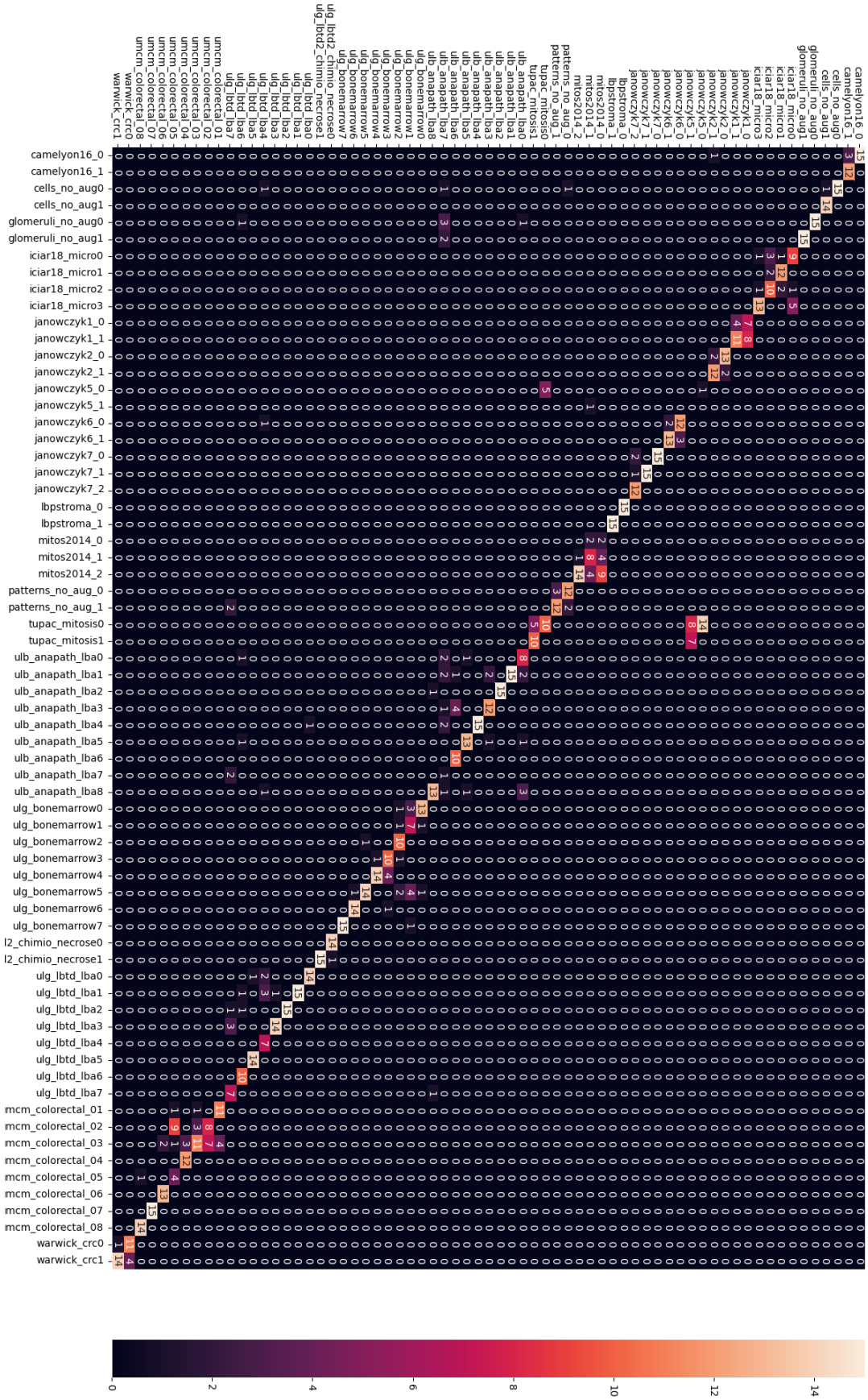

Figure 4.5: Confusion matrix obtained with ResNet50_DR trained on the histpathology dataset for 50 epochs with the Margin Loss. The dataset indexed is the training histopathology dataset, and the queries are made with images from the test set.

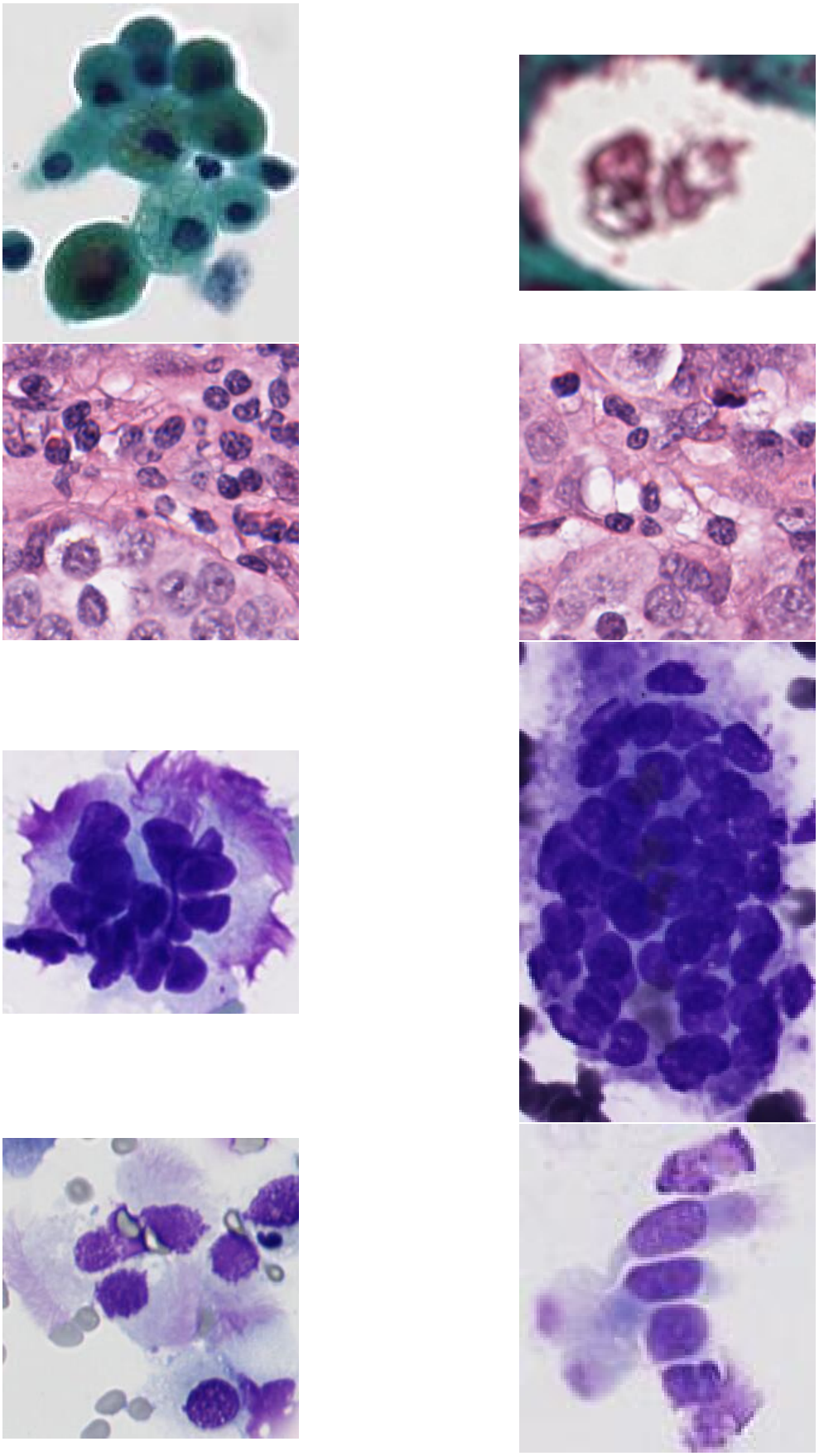

Figure 4.7: Examples of mislabelled images when ResNet50_DR is trained with the Margin Loss on histopathology. Left: query image. Right: retrieved image from another Cytomine project.

janowczyk2_0	iciar18_micro0
janowczyk2_1	iciar18_micro1
lbpstroma_0	iciar18_micro2
lbpstroma_1	iciar18_micro3
patterns_no_aug_0	tupac_mitosis0
patterns_no_aug_1	tupac_mitosis1
mitos2014_0	camelyon16_0
mitos2014_1	camelyon16_1
mitos2014_2	umcm_colorectal_01
ulg_lbtd_lba0	umcm_colorectal_02
ulg_lbtd_lba1	umcm_colorectal_03
ulg_lbtd_lba2	umcm_colorectal_04
ulg_lbtd_lba3	umcm_colorectal_05
ulg_lbtd_lba4	umcm_colorectal_06
ulg_lbtd_lba5	umcm_colorectal_07
ulg_lbtd_lba6	umcm_colorectal_08
ulg_lbtd_lba7	warwick_crc0

Table 4.13: List of classes used to trained the network when performing the test on generalisation.

- ulg_lbtd_lba7 with ulb_anapath_lba7, that are equally represented.

Except for ulb_anapath_lba7 that seems to be a real error, all the mislabelled classes outside the project are mislabelled with classes looking alike and at least as well represented.

Some examples of retrieved images are shown in Figure 4.8. Examples for each class of the dataset are shown in Appendix B.

For validation, the training set is indexed in the database, and the test set is used to make the queries. A top-1 accuracy of 77% and a top-5 accuracy of 82% are obtained.

4.8 Generalisation to unseen classes

While the network must provide good results on trained classes, it should also be able to generalise well to unseen classes, as it will surely be used more often on classes it has not been trained on.

In order to test the generalisation performance, we carried out an additional experiment where the network is trained on only the first half of the classes from the training histopathology set (the exact list of the used classes is provided in Table 4.13). When doing this test, the other half of the classes from the test set are indexed in the database and the queries are also made from the other half of the classes from the validation set. A top-1 accuracy of 57% and a top-5 accuracy of 82% are obtained, which is still acceptable, as the retrieved images are still visually looking alike.

Samples of retrieved images are shown in Figure 4.9.

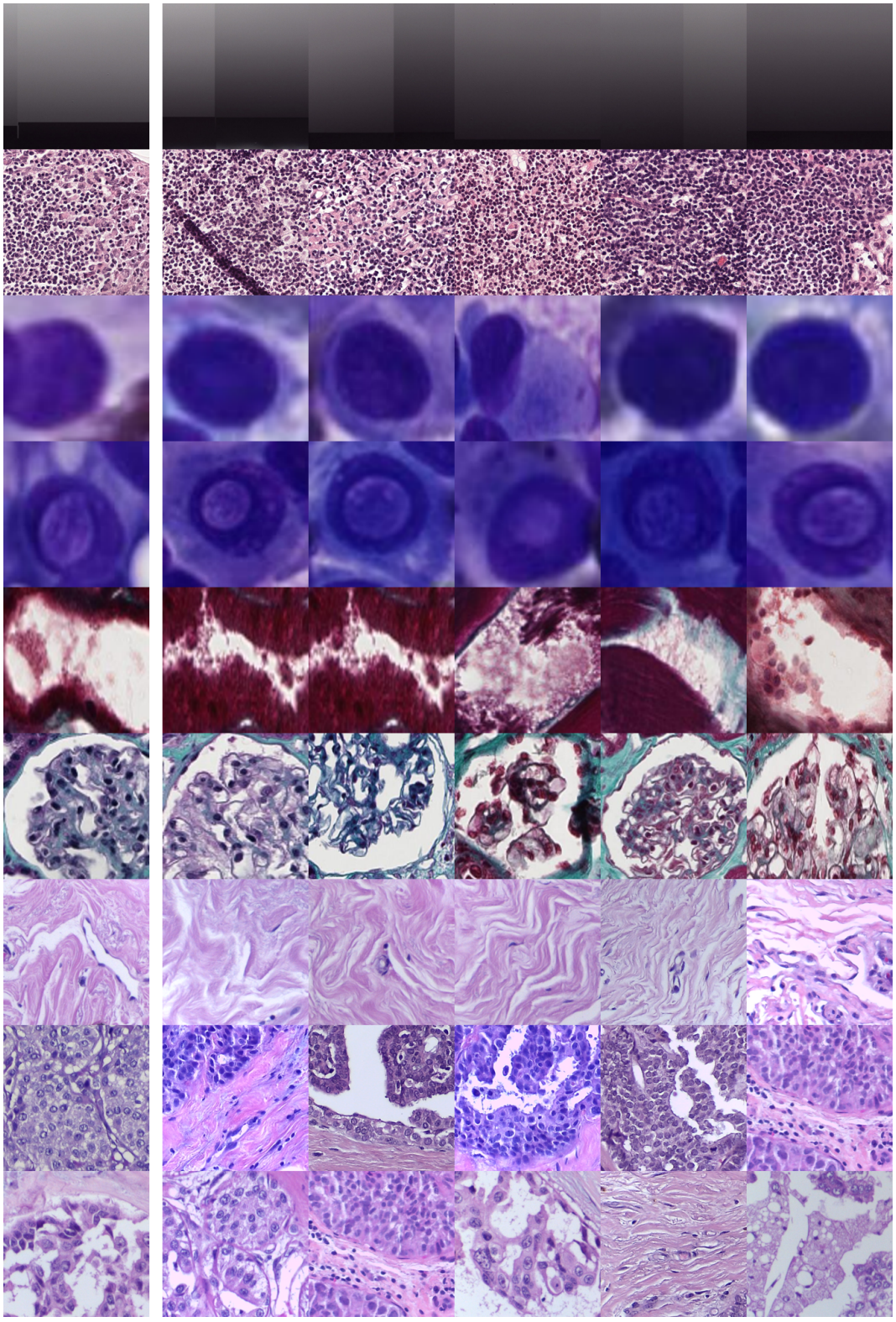

Figure 4.8: Examples of retrieved images when ResNet50_DR is trained with the Margin Loss on histopathology. Left: query image. Right: first five retrieved images.

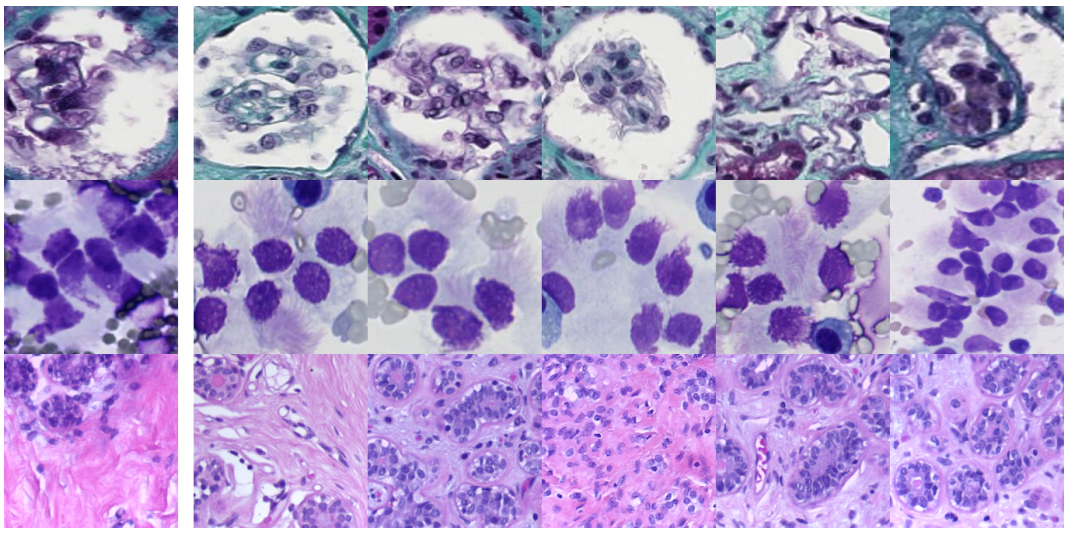

Figure 4.9: Retrieved images when ResNet50_DR is trained on half the image classes. Left: query image. Right: first five retrieved images.

4.9 Training Faiss for faster similarity search

While previous experiments yield promising results in terms of image retrieval accuracy, their actual implementation requires additional developments and evaluation due to the larger amounts of images that might be encountered in practice. In such a very large scale setting, a classical nearest neighbour search over feature vectors might not fulfil requirements in terms of response time.

This section studies the impact of the approximate search on the search time and the precision. To study this impact, the whole training dataset is indexed in the database (as it is the largest one with more than $600K$ images), and the test set will be used to make the queries. First, only the ideal case where the index is trained on the whole database will be studied. Then, a more difficult case will be studied: the index will be trained on the test set, the training set will be added to the database, and the validation set will be used for the queries.

As a first result, when the training dataset is indexed in the database, the time spent in the Faiss search is 2.5 s for 1,000 queries, or 2.5×10^{-3} s per query. That is 7.8 times faster than the brute force search. Concerning the accuracy, a top-1 accuracy of 77% and a top-5 accuracy of 82% are obtained, which is exactly the same as for the brute force search. Figure 4.10 is obtained by indexing several times the images of the training dataset with data augmentation. Up to 5,701,491 images were indexed in the database in total. As expected, the top-1-accuracy is pretty similar to the top-1 accuracy of the exact search: it keeps in line with the comment in section 3.2 that said that since the similarity measure is not exact, another approximation will not have much of an impact. But the top-5 accuracy drops close to the top-1, meaning that if the first retrieved image is not of the same class as the query image, no image of the same class is retrieved. By analysing the obtained results, the reason for this behaviour is immediately identified: the first five retrieved images are all coming from the same original image but with data augmentation (Figure 4.11). This illustrates the fact our network architectures are robust to variations generated by data augmentation.

On another hand, the search is more than 10 times faster using the approximate

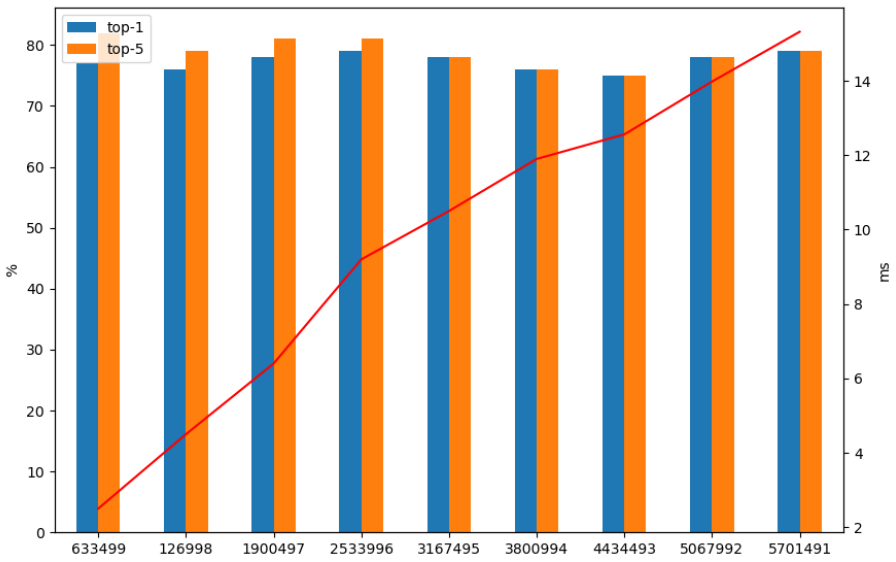

Figure 4.10: Top-1 and Top-5 accuracy with trained Faiss index as a function of the number of images indexed in the database (left y-axis). Red curve is the mean time spent in the Faiss search (right y-axis).

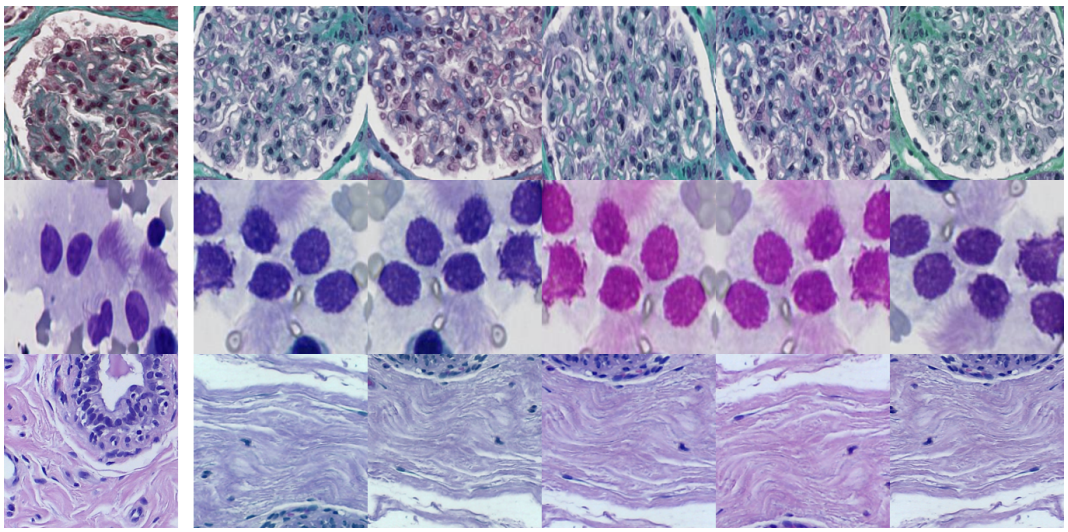

Figure 4.11: Retrieved images with trained Faiss index. At some points, the index retrieves several times the same image as it was indexed several times in the database (due to data augmentation).

Method	Training dataset	Top-1	Top-5
Selected ResNet50_DR	histopathology	82	94
Pretrained DenseNet-121	/	70	88
Pretrained DenseNet-121_DR	/	71	88
MarginLoss with DenseNet-121 (30 epochs, exp scheduling)	ImageNet-2012	67	86
Margin Loss with DenseNet-121_DR (20 epochs, frozen weights, exp scheduling)	histopathology	73	91

Table 4.14: Accuracy on the test dataset using all the images from the validation set as queries

search. This speed-up factor might be reduced or increased at will: by tuning the parameter τ (Equation 2.19), it is possible to consider more or less centroids during the similarity search. A larger τ will bring a better accuracy, but will lead to a slower retrieval time, and the search time increases linearly with τ. In our experiment, it is set to $\sqrt{\#\text{ indexed vectors}}/10$, so that the retrieval time does not increase linearly with the size of the database. Given that the top-1 accuracy is not impacted by much, the factor τ could be reduced even more. In practice, this could be adjusted to meet the good accuracy-speed compromise.

When the index is trained on the test dataset, and the training set is indexed, the following results are obtained with the previously used validation set (with 1,000 queries): a top-1 accuracy of 77%, a top-5 accuracy of 85% and a total Faiss search time of 2.67 s. This means that if the training distribution is a representative sample of the indexed database, the quality of the search is not significantly impacted.

For an even faster search, Faiss could be used on the GPU. But it was chosen not to do so for reasons presented in Chapter 5.

4.10 Legitimacy of the protocol

This section compares the evaluation protocol to two more classic ones, that will be called first protocol and second protocol.

The first protocol is very simple. The accuracy is just computed by using the images from the validation set as queries. But it must be remembered that by doing so, the wrong predictions of underrepresented classes will be drowned by the correct predictions of overrepresented classes.

The second protocol provides a solution to alleviate this problem: the two most represented classes, camelyon16_0 and janowczyk6_0, are removed from the validation set.

Tables 4.14 and 4.15 display the results obtained with both protocols with previously tested methods.

The results in the tables correspond to the results obtained in Table 4.4: the model that are better in Table 4.4 are also better in Table 4.14 and 4.15, and the contrary is also true for models that are worse.

The idea of the proposed evaluation protocol was that by eliminating the disparities in the validation set, the accuracy would reflect more the quality of the search, and that it would be easier to discard a model without analysing its results further. It seems that the second protocol leads to similar results.

Method	Training dataset	Top-1	Top-5
Selected ResNet50_DR	histopathology	70	91
Pretrained DenseNet-121	/	49	77
Pretrained DenseNet-121_DR	/	49	78
MarginLoss with DenseNet-121 (30 epochs, exp scheduling)	ImageNet-2012	45	75
Margin Loss with DenseNet-121_DR (20 epochs, frozen weights, exp scheduling)	histopathology	54	83

Table 4.15: Accuracy on the test dataset using the images from the validation set, by removing the classes camelyon16_0 and janowczyk6_0.

If it were not for the lack of time, all the results would have been recomputed with this second protocol.

4.11 Discussion about the results

As Table 4.4 shows, the 90% top-5 accuracy obtained with ResNet50_DR comes from a long way, as the first implemented method with the mosaic methodology only obtained a top-5 accuracy of 36%. The entirety of the results computed with the mosaic were not presented in this work as they are quite poor, and this, with both the k-means extraction and the completely random mosaic. This method, even if it is used with a small dataset consisting in a carefully selected number of classes from this histopathology dataset, is very slow and moreover provides poor results compared to using a Faiss index.

In our study, it was initially thought that the weights of the network should be frozen during training, and only the last linear layer should be updated. This is because the network is used as a feature extractor, and therefore, it should not have needed to be updated. Afterwards, it was hypothesised that the reason for the poor results was that images from the same class in the training histopathology dataset do not necessarily look alike (Figure 4.12). This is the reason why, ultimately, other datasets containing natural images were explored, similar to what was done with Smily, a successful architecture where the network is exclusively trained on natural images. But the datasets used were either not adapted for this task, or not large enough for this task, as the results provided by the networks trained on both INaturalist and ImageNet-2012 provided results less than their baseline. And this, with both frozen and unfrozen weights.

It appears that this initial hypothesis was very wrong. As soon as the network DenseNet-121_DR was trained with the Margin Loss under the same training conditions, but with unfrozen weights, it provided much better results. In fact, DenseNet-121_DR and ResNet50_DR provide similar results when trained with the Margin Loss on the histopathology dataset, but since ResNet50_DR is considerably faster, it is the network that was retained. Moreover, ResNet50_DR generalises quite well to unseen classes when trained with the margin loss.

In this master thesis, a more recent architecture than the usually used ResNet50 and DenseNet-121 for image retrieval, was also tried. The tested architecture is DeiT, simply because it is was already successfully used for image retrieval [El-Nouby et al., 2021]. But both DenseNet-121 and DeiT appear to be much slower

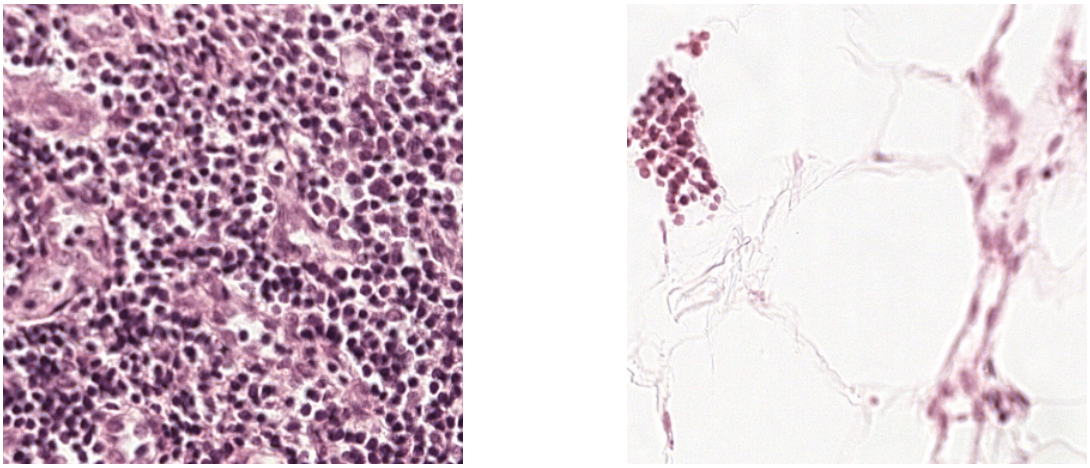

Figure 4.12: Two images from the class camelyon16_0 that do not visually look alike

than ResNet50 (more than twice). And while the results obtained with DenseNet-121 and ResNet50 are on par, DeiT results lag far behind. Either it was not trained for long enough (it is trained for at least 2,000 epochs in [El-Nouby et al., 2021]) or the Margin Loss is not well suited for DeiT.

Concerning the generalisation, ResNet50_DR provides more than satisfactory results. As a reminder, retrieval results for ResNet50_DR for one image of each class of the histopathology dataset is shown in appendix B.

In Luigi, the authors suggest to perform some transformations on the query image, such as rotations and flipping, and to compute the nearest neighbours of each transformed image and the query, and to return the k images to which the least distances are associated. Doing so resulted in a drop of 5% top-5 accuracy and a computation time 4 times longer for the retrieval.

In Smily, vectors of augmented images are indexed. This does not bring any accuracy improvement as it was shown in Section 4.9: indexing augmented images will lead to the retrieval of the same image for the first few images. Our experiments suggest it is better to apply geometry data augmentations during training.

Finally, the resulting model has a much larger accuracy on the dataset than the Randomised Trees technique, i.e. the previous method used on the Cytomine website.

Chapter 5

Distribution on different servers

In order to handle large datasets in a reasonable amount of time, they need to be stored on different servers, to parallelise the search. This is a solution that was already adopted by Smily (Section 2.2.2) and the Randomised Trees system (Section 2.2.1).

The solution proposed in this work is a master-worker architecture, as it simplifies the synchronisation between the different servers. Moreover, the clients only need to know the IP address of the master, which means that the image servers can be added or removed at will without the need of notifying the clients about the change.

In this setting, clients interact with the master only. They can ask to retrieve similar images, or to index a new image for example. The role of the master is merely to be an interface between the clients and the image servers. Each image server stores a part of the global dataset. Figure 5.1 presents an overview of the architecture.

The model is deployed on each image server, as computing the feature of an image is where a large part of the time is spent (cf. Table 4.10). Deploying it on the master could be a bottleneck.

A REST Api is created for the clients. The master and the servers are developed using FastAPI[1]. It was preferred to Flask[2] as it allows the use of the new asynchronous capabilities of Python with the `await` mechanism. A problem encountered during the development was that the Faiss indices stored on the GPU would sometimes be duplicated for an unknown reason, leading to an out-of-memory error. This is the reason why Faiss is used on the CPU, as the indices are never duplicated.

The different implemented actions are described in the following sections.

5.1 Retrieval of similar images

The most important operation of the proposed system is indeed the retrieval of similar images sent by the client.

To do so, the client must send the input image to the master along with his/her Cytomine public and private key and the number n of desired similar images. The master first check if the public/private key pair is valid. Then the master acquires a lock related to the client: it needs to do so, as several concurrent requests from a same client might corrupt data related to that client on the image servers.

[1] `https://fastapi.tiangolo.com/`

[2] `https://flask.palletsprojects.com/en/2.0.x/`

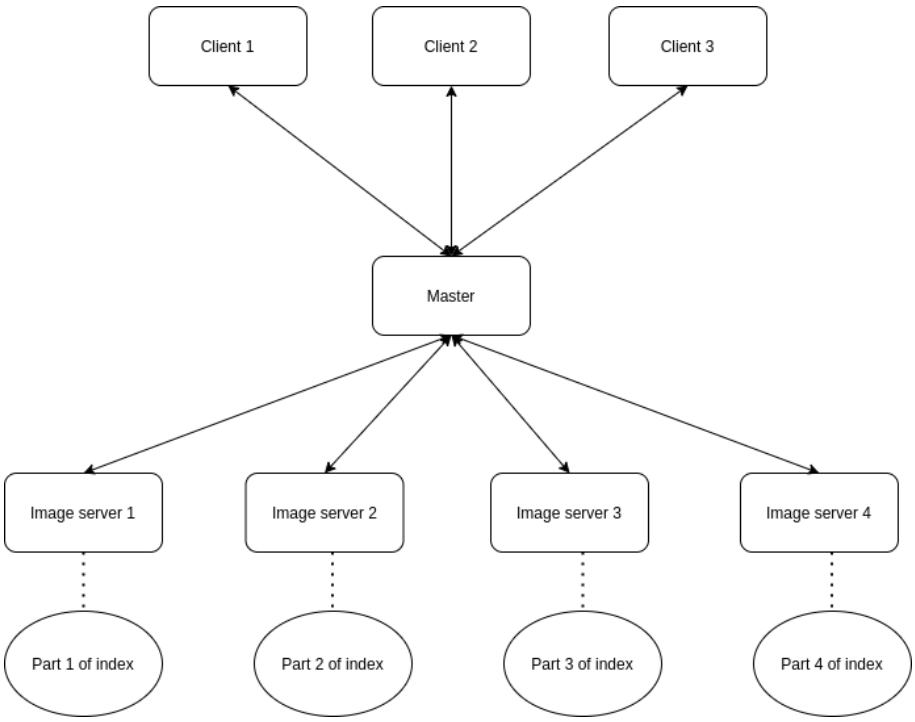

Figure 5.1: Illustration of the proposed architecture for the distribution. Several clients can interact with the master, that itself interacts with several servers, each indexing a part of the dataset.

When the master has checked the validity of a public/private key pair, it forwards the image, and the key pair to the image servers. If no server replies, the master sends an error to the client. The servers then each check the validity of this pair as a security measure. Each server acquires a reader-writer lock in reader mode. The reason is that Faiss is thread-safe for concurrent reading (only when the index is on the CPU, it is not thread-safe for concurrent readings when the index is on the GPU), but not for concurrent reading and modification of the index, such as adding or removing vectors. This means that every operation on the index, such as reading or modifying the index, must be protected.

When locked, each image server retrieves the names and the distances of the n most similar images in their own index. They need to store those names, and send the distances to the master. This is the reason why there is a lock related to the client on the master: two concurrent requests from a same client could corrupt the name list.

Once the master receives the distances from each server, it can figure out which images are the closest and request them, along with their names, their class, and the distances between the query and the retrieved images. It can finally send them to the client with their names. Figure 5.2 summarises the operations.

In fact, the client can actually ask to retrieve either only labelled, unlabelled or a mix of both labelled and unlabelled images, depending on the application. The server returns either the label, if the image is labelled, or "Unkown" if the image is not labelled. For this reason, each image servers actually manages two Faiss indices, that are queried according to the type of the user's query. Only one Redis database is used since it is sufficient to append "labelled" or "unlabelled" to the ID of the Faiss vectors.

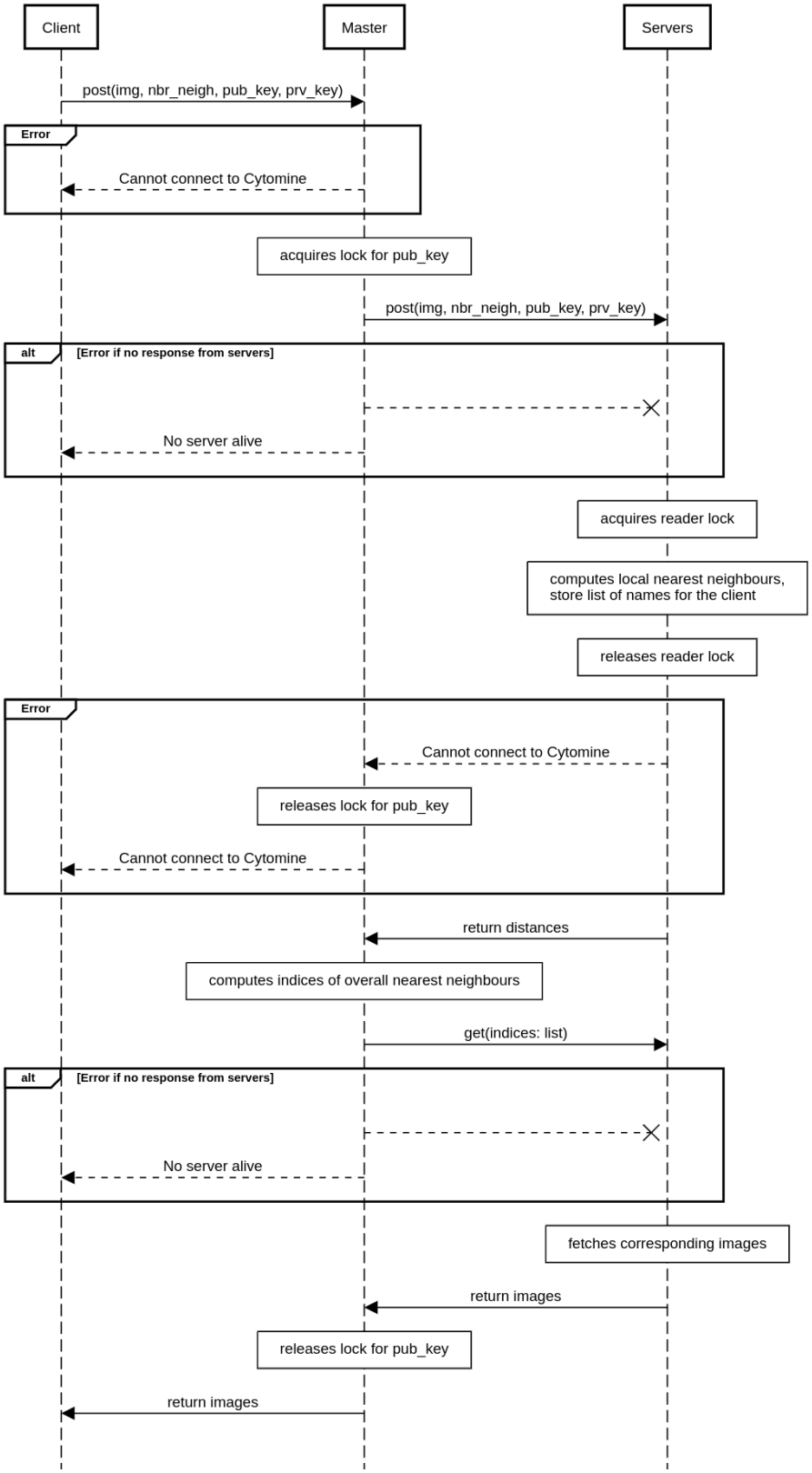

Figure 5.2: Workflow of operations for the retrieval

REST endpoint: /get_nearest_images, with the following query parameters:

- nrt_neigh: the number of similar images the user wants to retrieve;
- client_pub_key: the public key of the user;
- client_pri_key: the private key of the user;
- only_labeled: if 'true', it retrieves only labelled images, if 'false', it retrieves only unlabelled images, if 'mix', a mix of both.

The user must also send his/her image as multipart form-data. The endpoint returns a json string with the following keys:

- images: contains the list of returned images encoded in base64;
- cls: class of the returned images;
- name: name of the retrieved images;
- distances: distance between each retrieved image and the query.

5.2 Uploading a single image from the client's computer

For a client to post an image, he/she must post the image, his/her public/private key pair and possibly a label on the master. Once again, the master first checks if the public/private key pair is valid. The master posts the image to the server that contains the less images, in order to provide a similar search time on each server.

The server receives the image, and once again checks if the public/private key is valid, as a security measure. The server first saves the image. Then, it computes the feature vector of the image, and acquires a writer lock. It adds the image to the right index, depending on whether the image is labelled or not. It finally adds it to the Redis database by appending "labelled" or "unlabelled" to the key. It also adds the image name as a key and along with the Faiss ID, so that the image can be removed without searching the whole keyspace of the Redis database.

Figure 5.3 summarises the operations.

REST endpoint: /index_image, with the following query parameters:

- client_pub_key: the public key of the user;
- client_key: the private key of the user;
- label: the optional label of the image.

The user must also send his/her image as multipart form-data.

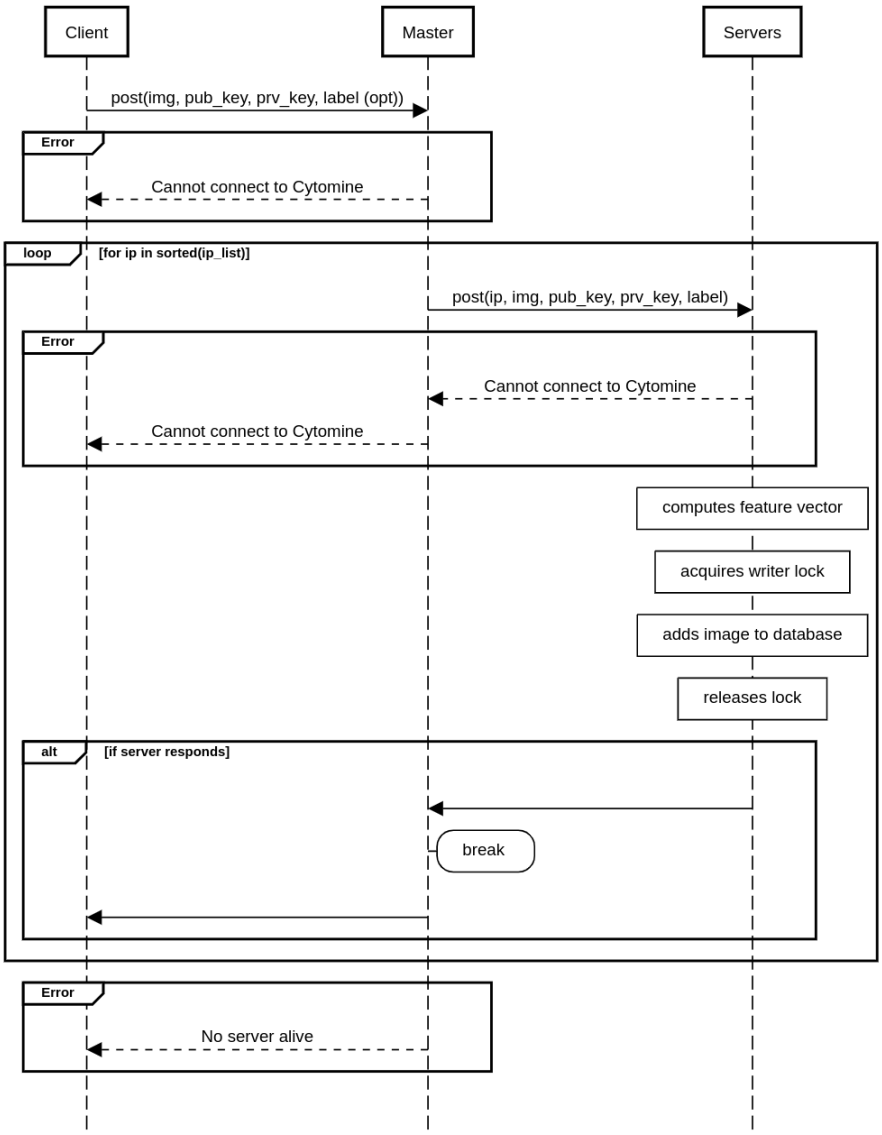

Figure 5.3: Indexing a single image

5.3 Uploading a folder from the client's computer

For a client to index a folder of images, he/she must post the folder as a zip file, along with a public/private key pair, and a boolean variable telling if the images are labelled or not, on the master. It is not possible to upload a folder that contains labelled and unlabelled images at the same time. If the images are labelled, then the label is the name of the subfolder the image is in. If the images are not labelled, then the folder should not contain any subfolder. Once again, the master checks if the key pair is valid. If it is valid, the master forwards the zip file to the server that contains the less images.

The server checks the validity of the pair, as a security measure, and then checks if the folder respects the format. If the format is not respected, an error is returned to the client. If there is no error, the server unzips the folder, and indexes all the images by batches, acquiring a writer lock at each loop.

Figure 5.4 summarises the operations.

REST endpoint: /index_folder, with the following query parameters:

- client_pub_key: the public key of the user;
- client_key: the private key of the user;
- labeled: True or False, depending on if the image to index are labeled or not.

The user must also send his/her folder as multipart form-data.

5.4 Removing an image

If a client receives an image that needs to be deleted for any reason, he/she can do so by providing the name of the image to the master, along with its public/private key pair. Once again, the master checks the validity of the pair. If it is valid, the master sends the name of the image to each server.

The server receives the name and the key pair. It checks the validity of the pair. If the pair is valid, the server checks if it stores the image. If it does, the server acquires a writer lock and fetches the Faiss ID by providing the image name to the Redis database. It also knows that way if it is a labelled image or not. It can then remove all the data concerning the image stored on the Faiss index, the Redis database and the image file.

Figure 5.5 summarises the operations.

REST endpoint: /remove_image, with the following query parameters:

- client_pub_key: the public key of the user;
- client_key: the private key of the user;
- name: the name of the image to remove.

5.5 Indexing labelled patches of slides stored on Cytomine servers

If a user wants to add all the annotations from a Cytomine project, he/she can do so by providing the project ID, the label of the patches he wants to index (e.g.

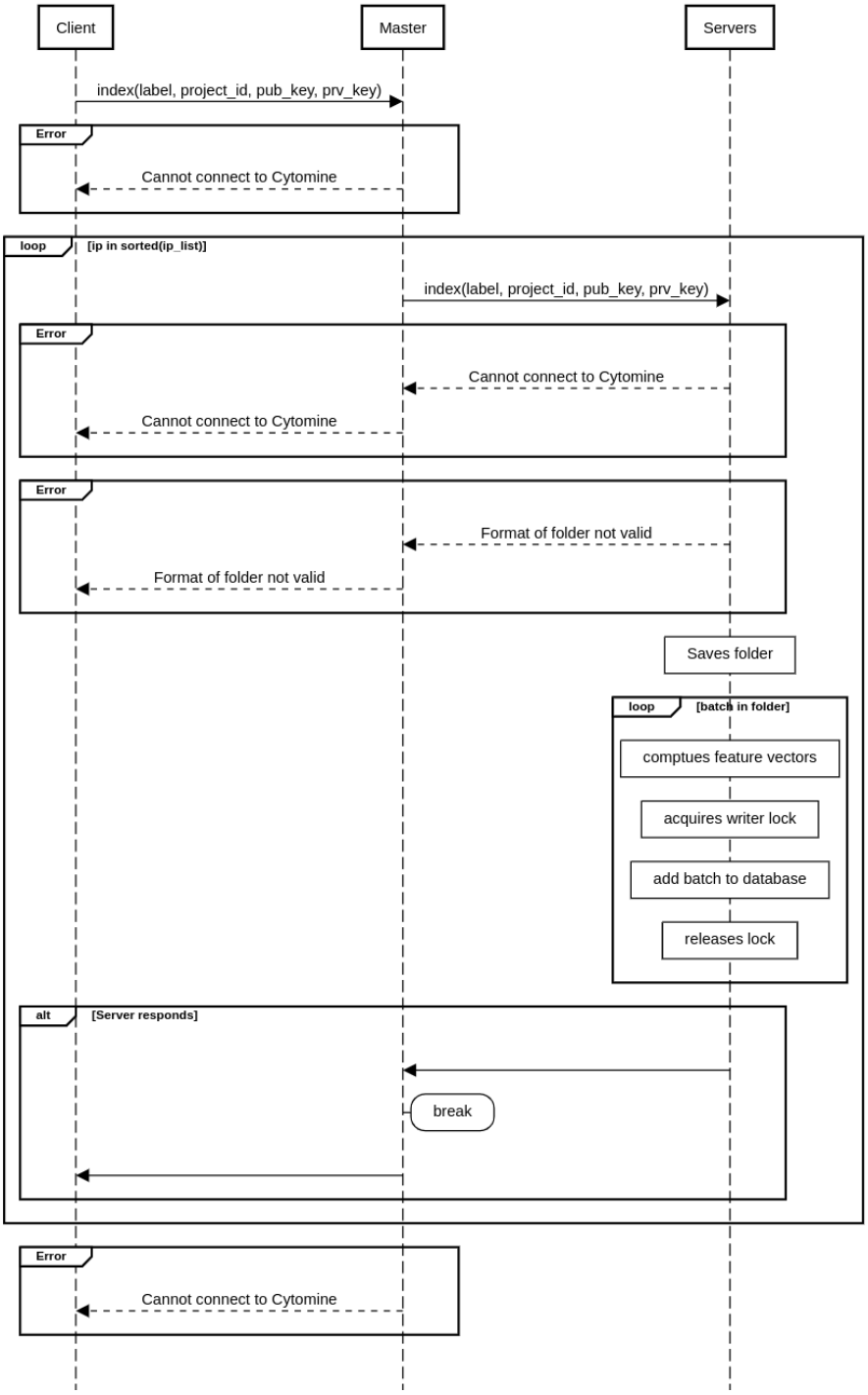

Figure 5.4: Indexing a folder

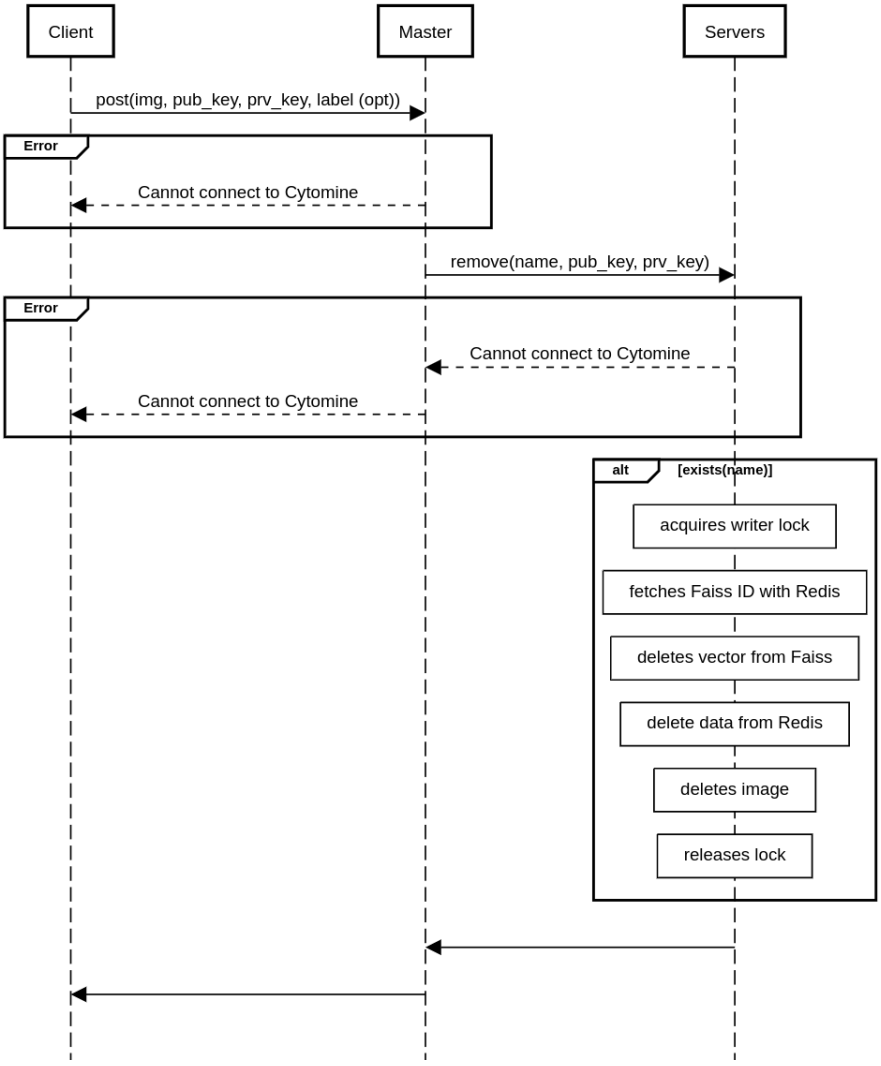

Figure 5.5: Removing an image

TUMOR), and of course his/her public/private key pair. The master first checks the validity of the key pair. If it is valid, it asks the server that has the less images to index the slides.

Once again, the server first checks the validity of the key pair. If it is valid, it fetches all the annotations of the project annotated with the provided label. The indexed annotations must have an area between 700mm^2 and 5000mm^2, so that they are neither too small nor too big. The patches are saved, and then added by batches, acquiring a writer lock at each loop.

Figure 5.6 summarises the operations.

REST endpoint: /index_slide_annotations, with the following query parameters:

- client_pub_key: the public key of the user;
- client_key: the private key of the user;
- project_id: the ID of the desired Cytomine project;
- label: the label of the annotations to index.

5.6 Indexing a project of slides stored on Cytomine servers.

As a very large amount of different patches can be extracted from a WSIs, the size of the database could rapidly increase if all those patches are indexed. In order to extract a minimum amount of informative patches, the same method as that used for Luigi is applied (Section 2.2.4.1): the tissue section of the slide is extracted with Otsu thresholding method. Then, patches of size 224 $\times$ 224 at different magnification levels, and the feature vectors are computed for each of those patches. Those vectors are clustered with k-means (k=500), and only the patches that are the nearest to their clusters are stored in the database.

For a client to index patches from a project of slides, he/she needs to send the ID of the project to the master, along with his/her public/private key pair. The master checks the validity of the pair, and fetches all the necessary information about the slides of the project. Then the master sends, slide by slide, the information about a single slide to the server that indexes the less images, along with the public/private key pair of the client.

The server checks once again the validity of the public/private key pair. It then downloads the slide, and applies the Luigi's method to gather the patches to index. It saves those patches, and then indexes the patches by batches, acquiring a writer lock at each loop. It must be taken into account that those patches are indexed as unlabelled.

Figure 5.7 summarises the operations.

REST endpoint: /index_slides, with the following query parameters:

- client_pub_key: the public key of the user;
- client_key: the private key of the user;
- project_id: the ID of the desired Cytomine project.

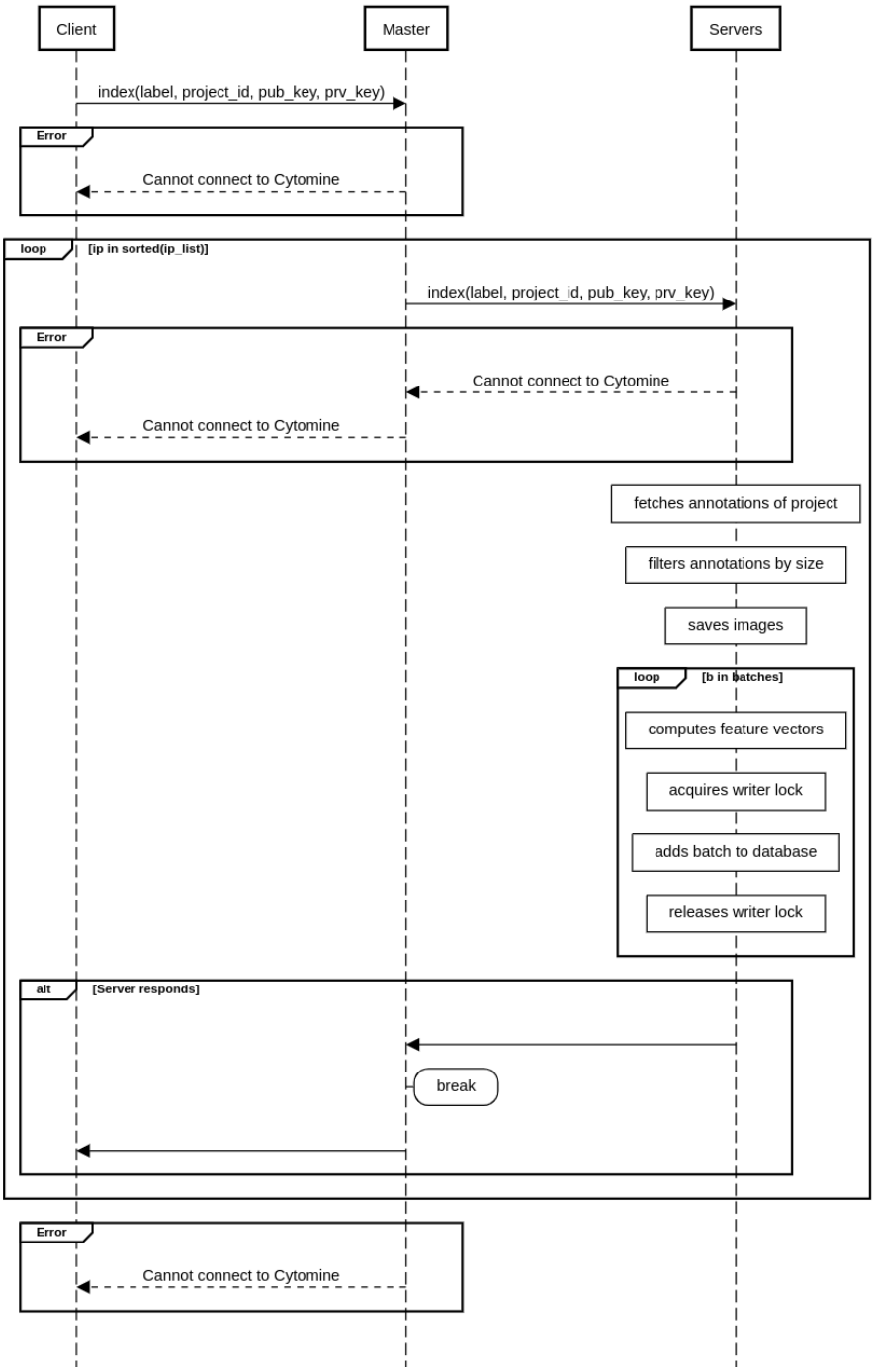

Figure 5.6: Adding slide annotations

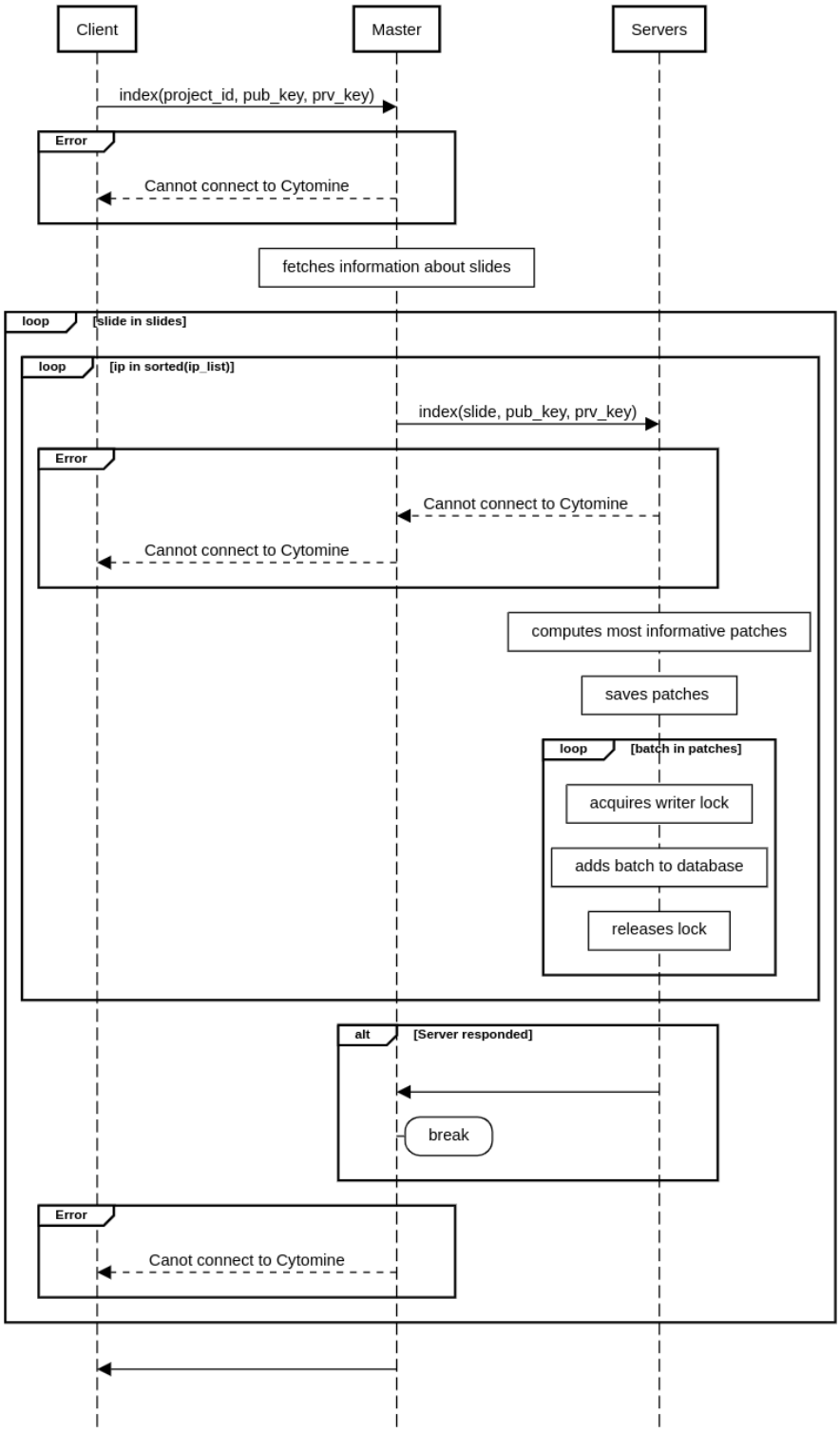

Figure 5.7: Adding project of slides.

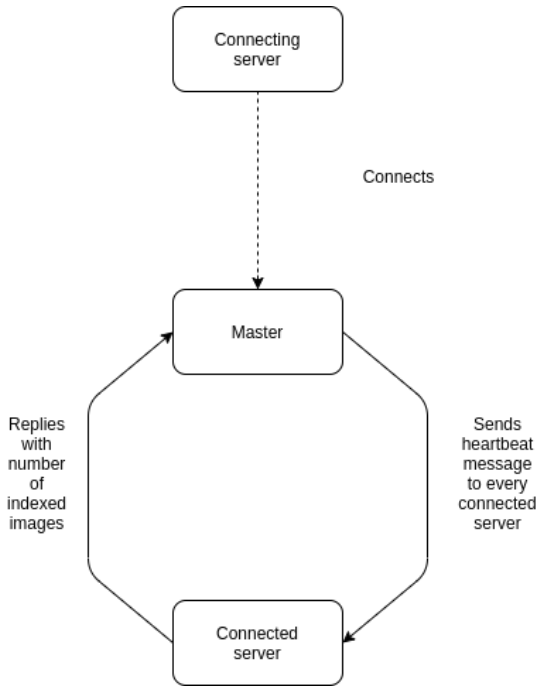

Figure 5.8: Connection and hearbeat message.

5.7 Heartbeat messages

For the search time to be minimal, the number of images needs to be equally spread on each image servers. When a user makes a query to retrieve similar images, the master needs to receive the response from each server to send it to the client. If one image server contains more images than the others, it will take more time to reply to the master, and as a result, the master will take more time to reply to the client.

So, when a user wants to index images, the master needs to know the number of images each server contains, in order to send the images on the server that contains the less.

This is done with a heartbeat mechanism: every few seconds, the master sends a heartbeat message to every worker, which replies with their number of indexed images. When the master does not receive the response of a worker, it considers that it crashed.

For the master to be aware of a server, the server must send a "connect" request beforehand, which is typically done when the server is started.

The operations are summarised in figure 5.8.

5.8 Tests conducted

This section describes the few numbers of tests that were conducted with the distributed architecture. For tests that require several servers, two servers were run on two different machines.

A client can successfully retrieve images from the two different servers. He/She can also successfully put images from his/her own computer on one of the servers, one by one or by sending a zip file, even when one crashes just before or during the operation. A client can also successfully delete an image from the global index.

For the following tests, the master and the only worker were run on the same computer. The test dataset is already indexed in the database, and the obligatory connection to Cytomine is removed, in order to simulate a large amount of connections at the same time.

For the retrieval time, the system can retrieve 100 "simultaneous" requests from different users in about 10 seconds.

For indexing a new image, it takes the system 18 seconds to handle 100 "simultaneous" requests from different users.

When combining 50 indexations and 50 retrievals, it takes the system 13 seconds to handle.

Chapter 6

Conclusions

This study responds to a request from Cytomine R&D and to needs in the digital pathology field. In 2010 Cytomine developed a Content-Based Image Retrieval system that could be applied on small scale datasets. Their wish was now to integrate into their platform a new one applicable to million or billion images.

For this master thesis, they set the following goal to achieve: to design a new architecture and a training method to allow the retrieval of histopathology images on a large scale.

But to go even further, we decided to aim at an additional goal: to store the dataset on different servers.

Have these objectives been achieved?

At the beginning of this research, many hopes were placed in the mosaic method, but after a while, it became clear that it was not adequate to bring about the expected results. It was thus necessary to turn to other models found in academic articles to move forward.

Three publications largely contributed to the construction of an architecture yielding very encouraging results: Smily and Luigi, for the idea of the approximate nearest neighbour search, and "Revisiting training strategies and generalization performance in deep metric learning" by [Roth et al., 2020], for exposing us the Margin Loss.

By slightly modifying ResNet50 and training it with the Margin Loss, the developed architecture is able to retrieve similar histopathology images. On the evaluated dataset, it retrieves an image of the same class as the query image in the top-5 results nine times out of ten.

Concerning the search time, the results can be available in 20 ms on a database of more than 5,000,000 images thanks to the use of a trained Faiss index for approximate search. Considering these results, we can say this project fulfils the expectations Cytomine expressed.

As far as our objective is concerned, it was also achieved as the solution is distributed on different servers, and users can retrieve most similar images spread on each of them

Limitations and future works

This study obviously contains some limitations.

First, the solution was tested on only one dataset. Although this dataset was quite large, we cannot be sure of how it will fare in real conditions.

Second, this work was not reviewed by practitioners or researchers. They might not have the same idea of visually looking alike histopathology images as we have.

The network we built provides results we had not hoped for, but there is of course still room for improvement.

The scores could certainly be increased by tuning the hyperparameters of the network, what was not done for this research, and/or by choosing another loss than the Margin Loss.

To maintain the best possible results during the lifetime of the solution, the network could, for instance, be retrained from time to time on the newly gathered labelled dataset.

Our architecture and REST API were created to facilitate the seamless integration of the distributed solution into the Cytomine platform. But it must be noted that, so far, this integration has not yet been carried out, and that the architecture can still be optimised.

Our code is open-source and available at `https://github.com/stephdef08/tfe` (for the Mosaic methodology) and `https://github.com/stephdef08/tfe2` (Deep metric learning and Faiss).

List of Figures

List of Tables

Bibliography

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[Clark, 2015] Clark, A. (2015). Pillow (pil fork) documentation. `https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf`.

[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*.

[El-Nouby et al., 2021] El-Nouby, A., Neverova, N., Laptev, I., and Jégou, H. (2021). Training vision transformers for image retrieval. *arXiv preprint arXiv:2102.05644*.

[Gao et al., 2016] Gao, Y., Beijbom, O., Zhang, N., and Darrell, T. (2016). Compact bilinear pooling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 317–326.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hegde et al., 2019] Hegde, N., Hipp, J. D., Liu, Y., Emmert-Buck, M., Reif, E., Smilkov, D., Terry, M., Cai, C. J., Amin, M. B., Mermel, C. H., et al. (2019). Similar image search for histopathology: Smily. *NPJ digital medicine*, 2(1):1–9.

[Huang et al., 2017] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[Itseez, 2015] Itseez (2015). Open source computer vision library. `https://github.com/opencv/opencv`.

[Joblib Development Team, 2020] Joblib Development Team (2020). Joblib: running python functions as pipeline jobs. `https://joblib.readthedocs.io/`.

[Johnson et al., 2019] Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

[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*, 65:101757.

[Komura et al., 2018] Komura, D., Fukuta, K., Tominaga, K., Kawabe, A., Koda, H., Suzuki, R., Konishi, H., Umezaki, T., Harada, T., and Ishikawa, S. (2018). Luigi: Large-scale histopathological image retrieval system using deep texture representations. *biorxiv*, page 345785.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

[Lin et al., 2013] Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

[Louppe, 2021] Louppe, G. (2021). Info8010: Deep learning. `https://github.com/glouppe/info8010-deep-learning`.

[Marée et al., 2010] Marée, R., Denis, P., Wehenkel, L., and Geurts, P. (2010). Incremental indexing and distributed image search using shared randomized vocabularies. In *Proceedings of the international conference on multimedia information retrieval*, pages 91–100.

[McCurdy, 2011] McCurdy, A. (2011). redis-py. `https://github.com/andymccurdy/redis-py`.

[Mormont et al., 2020] Mormont, R., Geurts, P., and Marée, R. (2020). Multi-task pre-training of deep neural networks for digital pathology.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Redis, 2009] Redis (2009). Redis website. `https://redis.io/`.

[Roth, 2019] Roth, K. (2019). Samples per class. `https://github.com/Confusezius/Deep-Metric-Learning-Baselines/blob/60772745e28bc90077831bb4c9f07a233e602797/datasets.py#L428`.

[Roth, 2020] Roth, K. (2020). Deep metric learning losses and tuple mining. `https://github.com/Confusezius/Revisiting_Deep_Metric_Learning_PyTorch`.

[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 *International Conference on Machine Learning*, pages 8242–8252. PMLR.

[Teh, 2020] Teh, E. W. (2020). Proxynca++ implementation. `https://github.com/euwern/proxynca_pp/blob/master/loss.py`.

[Teh et al., 2020] Teh, E. W., DeVries, T., and Taylor, G. W. (2020). Proxynca++: Revisiting and revitalizing proxy neighborhood component analysis. In *European Conference on Computer Vision (ECCV)*. Springer.

[Tejaswi, 2019] Tejaswi, S. (2019). Deep ranking dataset. `https://github.com/SathwikTejaswi/deep-ranking/blob/master/Code/data_utils.py`.

[Tellez et al., 2019] Tellez, D., Litjens, G., Bándi, P., Bulten, W., Bokhorst, J.-M., Ciompi, F., and van der Laak, J. (2019). Quantifying the effects of data augmentation and stain color normalization in convolutional neural networks for computational pathology. *Medical image analysis*, 58:101544.

[Tizhoosh et al., 2016] Tizhoosh, H. R., Zhu, S., Lo, H., Chaudhari, V., and Mehdi, T. (2016). Minmax radon barcodes for medical image retrieval. In *International Symposium on Visual Computing*, pages 617–627. Springer.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, 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*, pages 1386–1393.

[Wolf et al., 2020] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

[Wu et al., 2017] Wu, C.-Y., Manmatha, R., Smola, A. J., and Krahenbuhl, P. (2017). Sampling matters in deep embedding learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2840–2848.

[Zhai and Wu, 2018] Zhai, A. and Wu, H.-Y. (2018). Classification is a strong baseline for deep metric learning. *arXiv preprint arXiv:1811.12649*.

[Zhang et al., 2021] Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2021). Dive into deep learning. *arXiv preprint arXiv:2106.11342*.

Appendices

Appendix A

Using the REST API

The REST API can be used with each architecture presented in this thesis, except for the mosaic.

To launch the master, the following command must be executed:

```
$ python rest/master.py --host CYTOMINE_HOST [--ip IP
   --port PORT --http]
```

where --host is the Cytomine host. The following arguments are optional: --ip is the IP address of the server to use (default: 127.0.0.1), --port is the port to use (default: 8000). If the flag --http is raised, the server uses the HTTP protocol instead of HTTPS.

Before launching a server, a Redis server must first be launched, then the server:

```
$ redis-server redis.conf
$ python rest/server.py --host CYTOMINE_HOST [--
   master_ip MASTER_IP --master_port MASTER_PORT --ip
   IP --port PORT --model MODEL --num_features
   NUM_FEATURES --weights WEIGHTS --use_dr --gpu_id ID
   --folder FOLDER --http --db_name NAME]
```

where

- --host is the Cytomine host;
- --master_ip is the ip address of the master (default: 127.0.0.1);
- --ip is the ip address of the server to use (default: 127.0.0.1);
- --master_port is the port of the master (default: 8000);
- --port is the port used by the server (default: 8001);
- --model is the model of the network, either 'resnet', 'densenet' or 'transformer' (default: densenet);
- --num_features is the number of features the model extracts (default: 128);
- --weights is the name of the file storing the weights of the network;
- --use_dr uses the _DR version of the network (not for the transformer);
- --folder is the folder where the added images will be stored;

- --http to use the HTTP protocol instead of HTTPS;
- --db_name is the name of the files storing the Faiss indices.

Appendix B

Examples of retrieved patches

In this appendix are shown the results of the retrieval for each class of the histopathology dataset that are obtained with ResNet50_DR, trained with the Margin Loss on the histopathology training set. The training set is indexed in the database, and the test set is used to make the queries.