AI-assisted annotation of large and multimodal imaging datasets

Auteur : Bernard, Simon
Promoteur(s) : Geurts, Pierre; 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/2268.2/13291

Master Thesis

AI-assisted annotation of large and multimodal imaging datasets

Author :
BERNARD Simon
Supervisor :
MARÉE Raphaël
Co-Supervisor :
GEURTS Pierre

2^{nd} year of master in computer science and engineering
Academical year 2020-2021

Abstract

The annotation of histological images through different stains is an important task for diagnosis of diseases such as cancer, but it is also very time-consuming. Despite its repetitive nature, doing such annotations for a same tissue in several stains still requires specialized skills to be done. Nevertheless, the usage of computer vision and machine learning techniques may be used to reduce the time needed to perform this task.

This thesis will try to reduce the time needed by developing methods allowing to use the annotation from one tissue image and transfer it to its other modalities (i.e. images with other stains). The annotations considered are freehand polygons, delimiting the area of interest, and up to 25 stains per tissue are available in the dataset used.

To achieve such a transfer of annotation, several global feature-based and pixel-based registrations of the whole images are compared. Afterwards, local registrations are performed following the global ones to enhance the results. Those local adjustments are done either using a second time the best techniques from the global registration or by performing a local segmentation using a deep neural network. From those techniques, only the feature-based methods lead to honorable results, with a second local adjustment achieving either a much better or much worse registration. Even though the performances are not sufficient to reliably perform the fully automatic transfer of annotations, the feature-based methods may be used to give an estimate and reduce the interaction required from the annotator.

Acknowledgements

I would like to thanks those who help me in the creation of this thesis : Raphaël Marée for his constant support on this project, Pierre Geurts for the suggestions on the methods and the redaction, and Laurence Roussel for the proofreading.

Table of contents

Chapter 1

Introduction

The world is full of information which comes in plenty of flavour. For example, when an object burns, it is possible to sense it in multiple ways. The most natural way to perceive it is to *see* the flame, the smoke and so on. Moreover, it is also possible to *smell* the smoke or to *feel* the heat on the skin. All those channels of perception are called *modalities*. When people try to solve a problem by using different available modalities of a single object or event, they work on a *multimodal* framework. However, being able to link such modalities can be very challenging for many reasons such as the difference in physical units, in resolution or the synchronization of the different available modalities [1].

Multimodality can be used to accomplish many interesting tasks that can be divided in 5 categories [2]. The first is the *representation* whose goal is to summarize the information embedded in multiple modalities into a single object [3]. Then there is the *translation* where the objective is to build one modality from another one [4][5][6]. The third one is the *fusion* where the information is fused from multiple modalities often to perform a prediction. This step of fusion can take place at many different levels [7][8]. The fourth one is the *co-learning* where other modalities are used at the training to enhance the comprehension of a first modality, while this first modality will be the only one available at the prediction step. Finally, there is the *alignment* which aim at synchronizing the information embedded in different modalities [9]. The latter is the one that will be interesting to tackle the given problem. Though, those categories are not mutually exclusive as an alignment or a translation is often a good start to perform other multimodal tasks.

Researchers encounter multimodality in many fields. Some are quite natural like audio-visual multimodality [10] where sounds and images are both modalities used or meteorological monitoring where rain, wind or pressure (among many others) may be used to create forecast. Multimodality can also be out of the usual human perception field. In remote sensing, many types of images such as *hyperspectral*, *multispectral*, *synthetic aperture radar* or *light detection and ranging* can be used in addition to usual optical images [7][8]. The usage of multiple types of images is also present in the biomedical domain, especially for histopathology, such as *bright-field* or *second-harmonic generation* [3].

The histopathology is the field on which this work will focus. To be more precise, the histopathology is "the diagnosis and study of diseases of the tissues, and involves examining tissues and/or cells under a microscope" to cite the Royal College of Pathologists in UK[11]. They consider that around 20 millions slices are examined in the UK each year. An important part of the histopathology concerns the detection of cancer cells and their seriousness, but it is used for many other diseases/viruses. However, to be able to analyze the microscopical structure, they often need to use *stains* (e.g. Fig 1.1). Those stains allow to highlight different parts of the tissues, different types of cells or different parts of the cells. It is often useful to consider multiple stains (i.e. multiple modalities) for a given tissue. In this work, 51 different stains will be used. Moreover, multiple stains can also be used in a single image. In this category, 45 unique mixings of stains are present in the dataset that will be used[1].

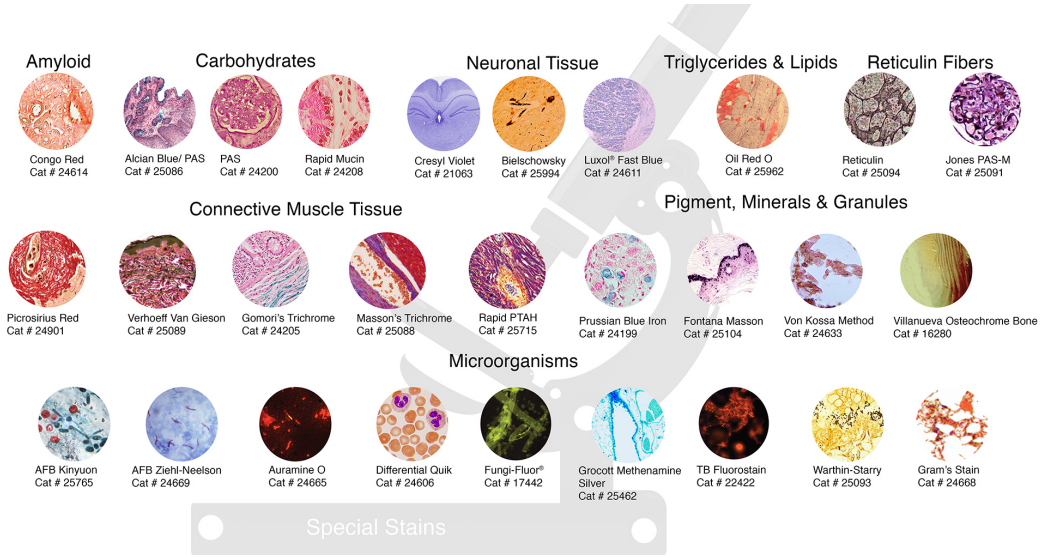


Figure 1.1 – Example of stains that can be used, multiple stains can be used for a single image. [12]

With the rise of machine learning (and more recently deep learning) techniques, people have tried to apply those techniques to histopathology. But such techniques often require the usage of annotated dataset which can be quite troublesome to do. To ease this task, a software called *Cytomine* [13][14] has been created.

[1] Actually, most of the images in the dataset are stained using a mix of different stains.

The core part of Cytomine is an "Open-source rich internet application for collaborative analysis of multi-gigapixel images using machine learning". However, the software is also available for images of more standard resolutions. It allows multiple collaborators to create annotations of many kinds : bounding boxes, landmarks or freehand polygons (e.g. Fig 1.2). Those annotations are then available through an API for Python and Java [15].

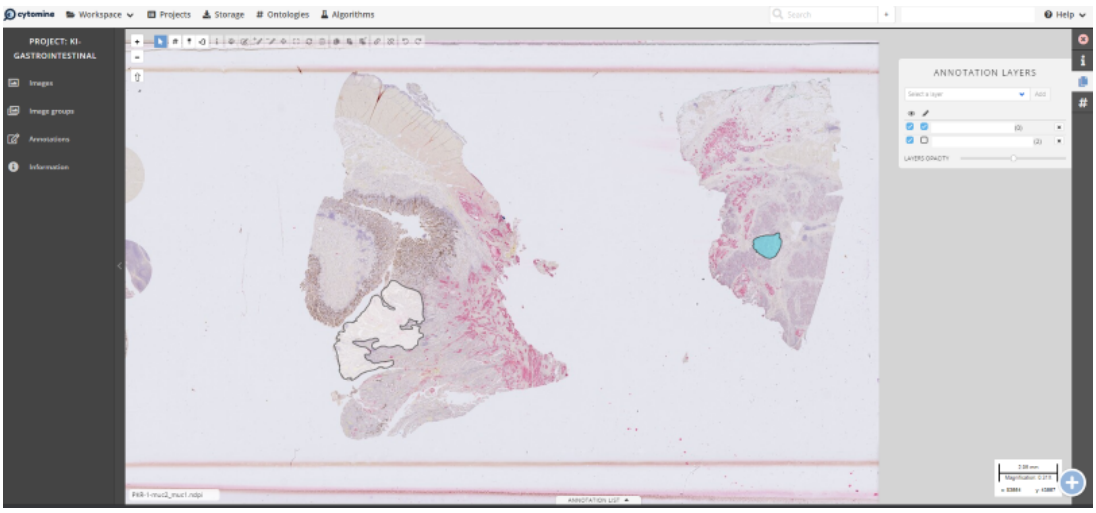


Figure 1.2 – Example of freehand polygons' annotations. In this image two annotations can be seen, one on the left part and another, highlighted in blue, on the right.

Those features make researchers' lives easier. Nevertheless, Cytomine aims at making their lives even easier by doing research and development in machine learning, image informatics, and big data to enhance the features available.

Recently, they released a new package of features enabling to create *image group* (Fig 1.3) and *linked annotation* (Fig 1.4) to enhance the way people can deal with multimodal

dataset. This allows to visualize them more easily through the web application but also to detect easily such group through the API.

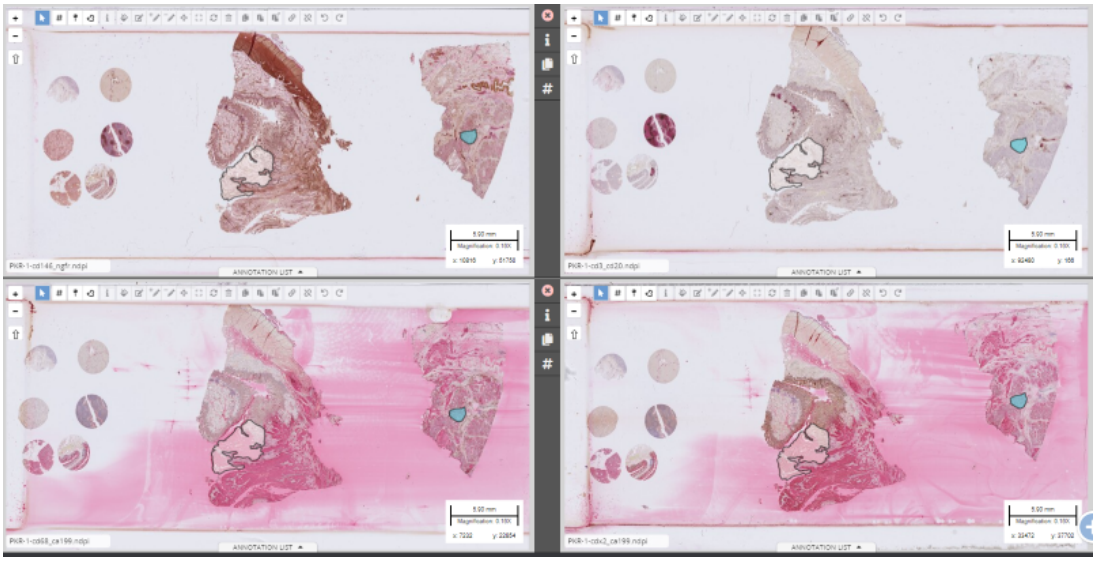


Figure 1.3 – Example of a few images from the same image group seen through the web service of Cytomine. The 4 images (from a group containing a total of 17 images) are stained with different stains' mixings. The two images at the bottom look similar because they were stained using a mixing of stains including the stain *ca199* in both case.

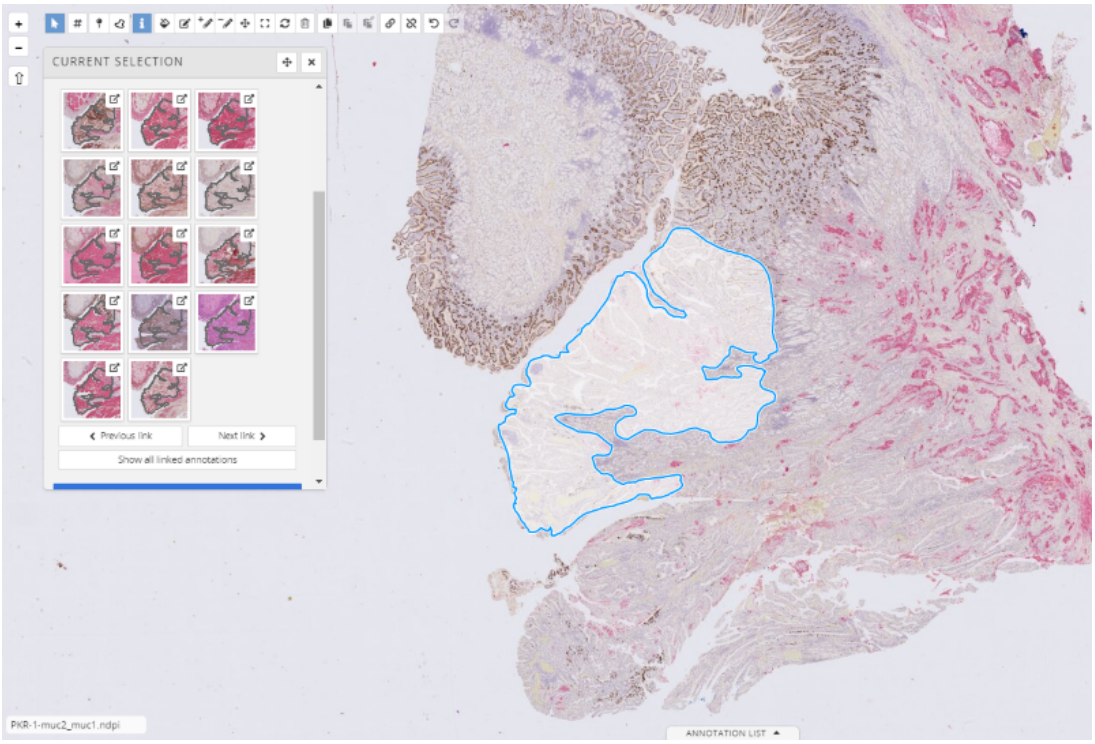


Figure 1.4 – Example of linked annotations seen through the web service of Cytomine. When selecting the annotation in the viewed image, the other annotations (in the other images from the same image group) of the same region that were linked to this one can be seen on the left panel.

Annotating such dataset is still very time consuming. The creation of linked annotations allow to transfer naively an annotation from an image to another of the same image group by printing the same polygon at the center of the view. As this may save a bit of time, it is still quite inefficient. Indeed, doing multiple stains of the same tissue is not an easy task. It involves to manipulate the tissue and it often ends up with the tissue not well aligned in the different pictures. Moreover, while it is sometimes possible to wash a stain to use another one, it is a heavy procedure. Most of the time, the different stains are applied to successive slices of the tissue which leads to non-rigid deformation among the different modalities. Finally, the images used in histopathology are captured in very high resolution (several gigapixels). They are known as *whole slide images* (WSI) and present opportunities for some applications but should also be manipulated with caution as many image processing techniques can not deal with such resolution in a reasonable time. For all those reasons, transferring annotations is thus quite challenging.

Chapter 2 will first present more formally the problem that will be treated and the dataset used to test the different methods. Then Chapter 3 will provide the required knowledge to understand the object manipulated and general techniques used in this work. Afterwards, Chapter 4 will describe the exact protocols in which the elements presented in the previous chapter are used and combined to solve the problem. The different protocols include first global registration using feature-based detector or similarity metric, and local registration using feature-based detectors again or a segmentation technique based

on a deep neural network. This chapter will also present the metrics used to assess their respective performances. Chapter 5 will follow and provide the results of the protocols. The last two chapters, Chapter 6 and Chapter 7, will respectively present the limitations and perspectives of this work in a first time, then provide a final conclusion to it.

Chapter 2

Problem & Dataset

This chapter will focus on describing more formally the specificity of this work. First the problem tackled will be precised, then the dataset used to experiment the different methods will be presented.

2.1 Problem specification

This section will try to give a better view of the objective of this thesis. As said in the previous section, Cytomine has developed a new system of linked annotations within annotation group. The current tool allows to add an annotation in one of the images and to paste it, either at the center of the view, either at the same pixel position in the other image (Fig 2.1). Then a few tools allow basic transformations : translation, scaling and rotation.

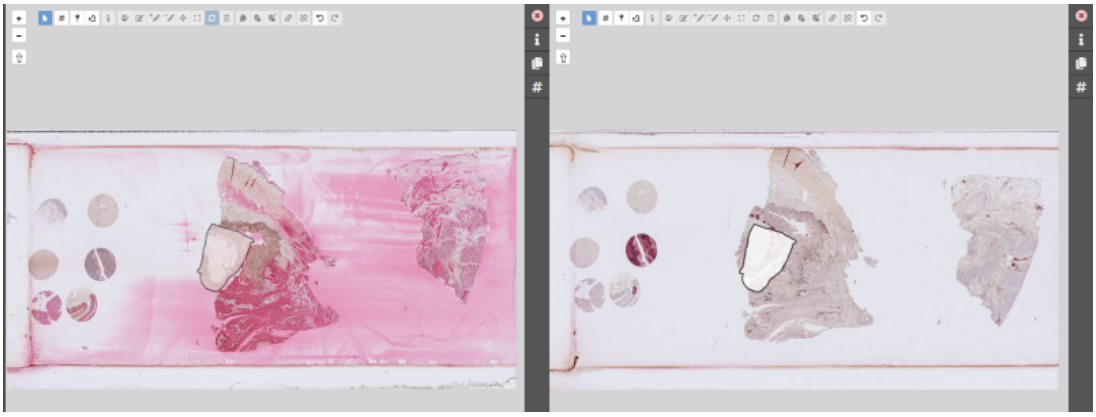
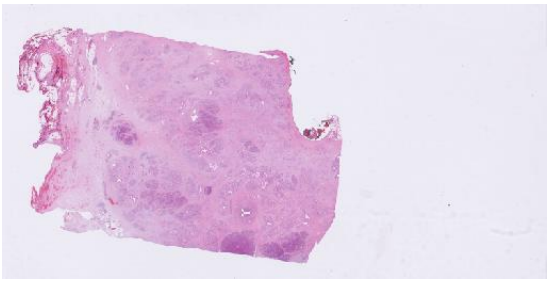


Figure 2.1 – Example of direct transposition. The annotation was done on the left image and transferred at the same location in the right image (from the same image group). In this case, the transfer ends up a bit too low and not exactly with the right rotation in the second image. Such transfer is however often much further than this from the right location.

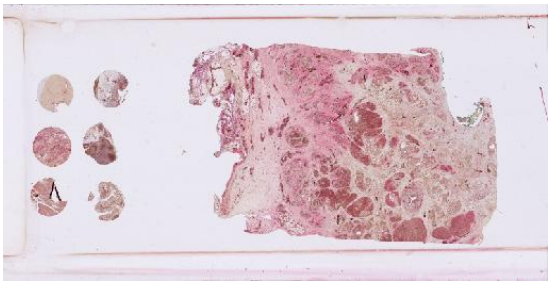
The idea here is to improve this feature by providing a more precise way to transfer the

annotation through the images. More precisely, this work will focus on freehand polygons. The end goal is to have the annotators to only annotate one of the images from an image group, then an automatic annotation would be made for all the other images. Afterwards, the only other interaction from the annotators may be an approbation as quality check which would be much faster than the actual procedure.

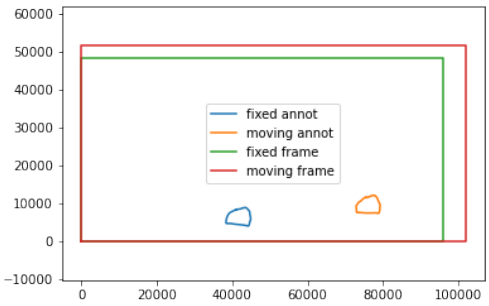
Fig 2.2 shows a simple example of an annotation registration. The first two images are respectively the *fixed* image on the left, the one where the annotation is supposed to be missing, and the *moving* image on the right, which contains the original annotation. The two following images show the initial respective places of the annotation to be found (the ground truth) in the fixed image, the blue one, and the original annotation from the moving image, the orange one. Finally, the last two pictures show an attempt to translate the original annotation on the ground truth.

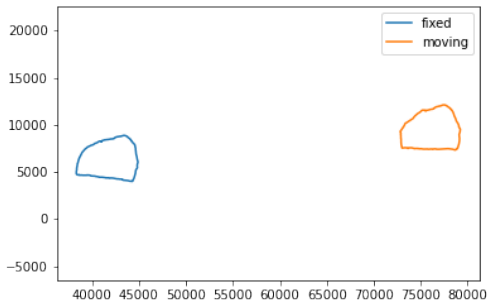


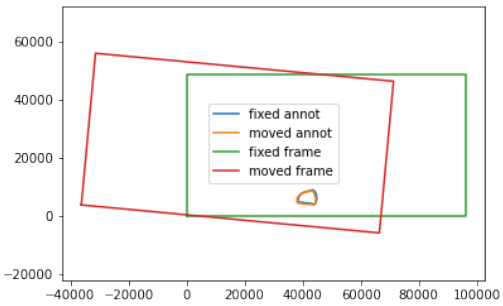
Fixed image



Moving image







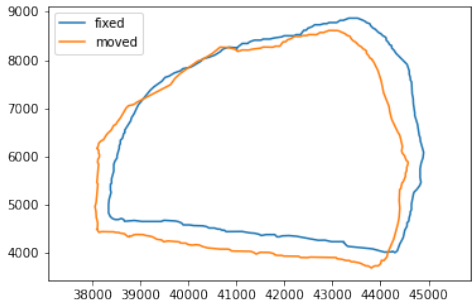


Figure 2.2 – Simple example of annotation translation. In this case a global registration of the moving image to the fixed image is performed (the frames represent the border of the images). Aligning globally the whole tissues (represented by the red and green frames) allow to also align more or less their substructures (colored in orange and blue).

In practice, the manually annotated image would contain many annotations (Fig 2.3), each of them transferred to all the other images in the image groups.

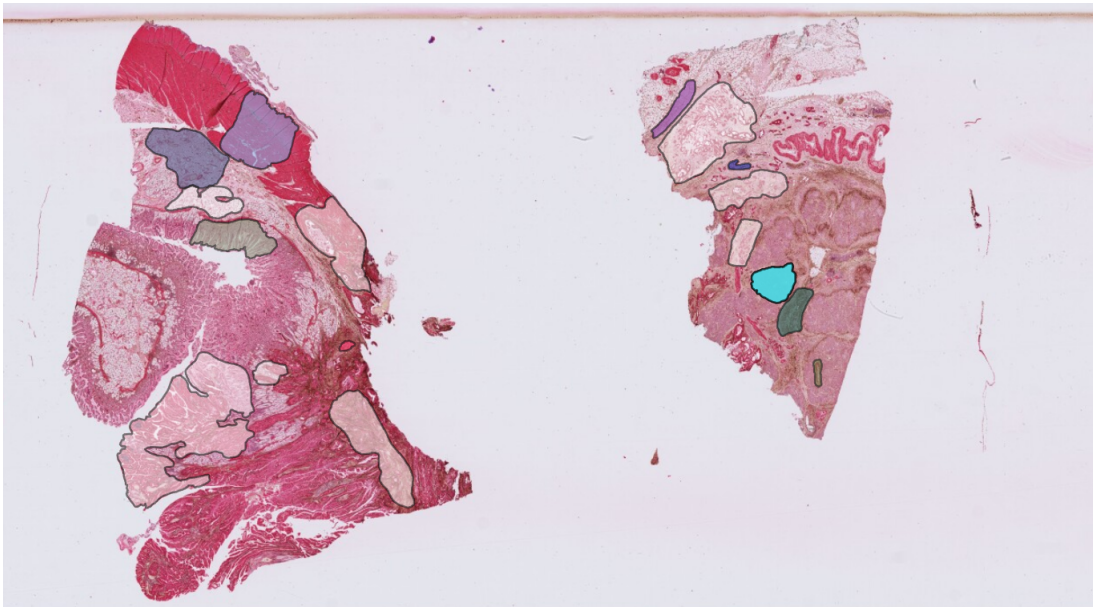


Figure 2.3 – Example of an image with all its annotations that should be transferred to the other images in its image group.

This is a quite novel challenge for several reasons. First, it needs to work for any type of organ (lung, bladder,..) as well as tissue subtypes (tumor regions, inflammatory cells, ...). Secondly, it also involves many different stains. Usual papers dealing with those kind of images often limit themselves to precise stains, precise organs and/or precise tissues [9] [16].

Another important part of the challenge is the usage of freehand polygons instead of simpler annotations such as landmarks (as in the ANHIR challenge [17]). Considering the non-rigid nature of the transformation for the tissue regions, the best annotation for the second image may not be achievable through a direct geometric transformation of the annotation in the first image. As an improvement of the current usage of the pasting, annotators may however be satisfied with such a transfer of annotation. This work will try to cover both possibilities to find the best compromise.

2.2 Dataset presentation

The dataset used for this work is called "KI-GASTROINTESTINAL" from the Karolinska Institute [18] and is currently still in creation process, as it started at the birth of this project. Carlos Fernandez Moro and Marco Gerling should be thanked for the creation of the dataset. This however also means that, unfortunately, there will not be other results to compare this work with at this stage. This section will thus try to be as precise as possible concerning the current state of the dataset used.

The dataset currently contains 291 images and 454 manual annotations. The details of

the partition into image groups can be found in Table 2.1 and an example of all the images present in an image group (PKR-1) is shown in Fig 2.4. The original images are about 100k x 100k pixels.

Image group	Number of images	Total number of annotations
PKR-1	17	47
PKR-2	25	68
PKR-3	19	35
PKR-4	20	37
PKR-5	20	36
PKR-6	11	33
PKR-7	15	51
PKR-8	19	42
PKR-9	12	41
PKR-10	23	64
PKR-11	17	0
PKR-12	16	0
PKR-13	16	0
PKR-14	17	0
PKR-15	9	0
LVR-1	17	0
CRLM-*	18	0

Table 2.1 – General state of the dataset

Figure 2.4 – All images present in the image group PKR-1

However, to assess the result of this work, only the annotations that are linked within images can be used. So here is the summary of such available groups (19), represented by their unique id, in Table 2.2. Two things should be noted from this table. First, the number of annotations in a group is sometimes much less than the number of images in that group (represented by the "$/x$", x being the number of images in the image group). Second, some annotations are very small compared to their respective images. This is shown in the last column which tells the surface of the annotation with respect to the image (averaged over all the annotation of the groups). Some of the groups have an average below 0.1% which may represent a much harder task than for the larger ones. The total number of annotations involved in those groups is also shown at the bottom of the table, which represent a bit more than half of the total number of annotations. All annotations from 529103480 (the id of the annotation group) in PKR-1 can be seen in Fig 2.5 with the annotations organized as their native images in Fig 2.4. The different annotation groups can also be seen through their first annotation in the appendix at Section 8.1.

Group ID	Image group	Number of annotations	Average size
526756968	PKR-2	25/25	0.441%
527083083	PKR-2	25/25	0.110%
527268885	PKR-7	15/15	0.125%
529103480	PKR-1	15/17	1.139%
529104012	PKR-1	15/17	0.127%
529108168	PKR-1	2/17	0.018%
529118038	PKR-3	18/19	0.409%
529119861	PKR-3	3/19	0.559%
529121751	PKR-4	20/20	0.302%
529123859	PKR-5	20/20	0.547%
529125794	PKR-6	11/11	0.453%
529129437	PKR-7	12/15	1.439%
529830333	PKR-8	15/19	0.030%
529832314	PKR-9	11/12	0.761%
529836154	PKR-9	2/12	1.841%
529837579	PKR-10	16/23	0.048%
529839089	PKR-10	16/23	0.038%
529840812	PKR-10	15/23	0.224%
529842978	PKR-10	2/23	0.058%
Total	/	258	/

Table 2.2 – Description of the annotation groups in the dataset

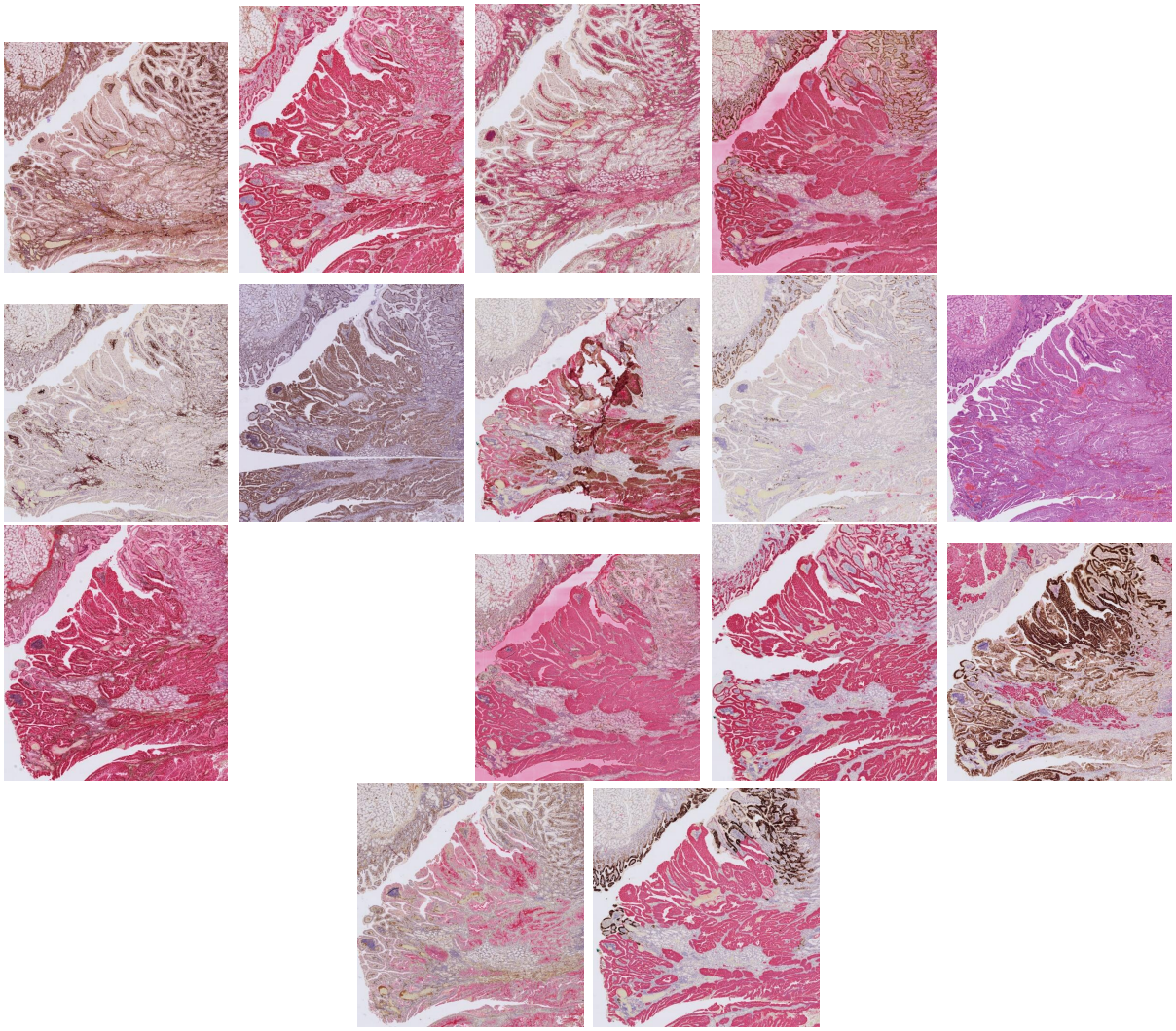


Figure 2.5 – First annotation group (529103480) in the PKR-1 image group. The annotation are ordered as in Fig 2.4 and the two images that does not have the annotation are left blank.

As stated in Chapter 1, many stains are used in the different images. The stains can be use either alone or combined with others in a single image with up to 4 stains in a single image. To understand the impact of the stains on the registration process, results per stain will be performed as well. But beforehand, some statistics associated to each stain are presented here. In the Tables 2.3 and 2.4 (45 entries), each image staining is considered as a unique stain, whatever the number of stains used within it (called mixed stains here). Examples for each of these stains are available in the appendix at Section 8.2. Then in Tables 2.5 and 2.6 (51 entries) each unique stain is considered, meaning that an image (and its annotations) containing 4 stains will be classified in these 4 stains (called single stains here).

The statistics performed for each stain, for both cases, include several things. First the number of images using this stain is counted. In the case of the mixed stain it will correspond to the number of image groups with an image using such combination, but for

the single stain several images in the same image group can use combination including that stain. The second statistic is the number of annotation. So if an image found above contains several annotations (linked with at least one other in its image group), they will all count as one in this category. The third statistic is the number of pairings. The number of pairings is the number of times an annotation from a moving image using such stain can be used. In other words, if an annotation from an image stained with *muc5ac_ muc6* is linked with 10 others annotations (i.e. 10 other stains in the same image group), it will account for 10 pairings as it is the number of times this annotation will be tested as a moving annotation by the various algorithms. The last statistic is the average relative size of the annotations within their image for each stain[1]. The first three statistics represent a kind of diversity : the number of organs, the number of tissue subtypes and the number of other stains it is paired with. The last states more about the "difficulty" as a smaller region will likely be harder to find, there are however many other factors more subjective such as how different they are from their surrounding that can not easily be described with numbers.

[1]The size is averaged over the number of annotations and not on the number of pairings.

Stains	# of image	# of annotation	# of pairing	Average size
muc5ac_muc6	8	12	190	0.335%
wt1_ca125	10	15	234	0.405%
ki67_p16	1	2	48	0.268%
smad4	6	10	172	0.360%
ki67_vim	8	12	199	0.344%
p53_cd34_cald_ck19	9	12	190	0.484%
pdl1_cd8	1	2	48	0.268%
pdl1	6	8	133	0.481%
p53_d240_cald_maspin	9	19	234	0.429%
p40_ck5	2	3	67	0.355%
muc2_muc1	7	11	165	0.339%
p63_ck17	4	8	130	0.264%
he	8	13	175	0.521%
ent1	4	6	112	0.507%
ck20_ck7	7	11	189	0.306%
ck17_ceam	10	15	234	0.401%
chra_cd56	4	7	128	0.257%
ceam	1	2	48	0.281%
cdx2_ca199	7	11	185	0.371%
cd146_ngfr	10	15	234	0.408%
cd68_ca199	6	8	136	0.367%
cd15_ck19	1	2	48	0.300%
hmga2_ck19	7	11	180	0.326%

Table 2.3 – Mixed stains statistics in the dataset [1/2]

Stains	# of image	# of annotation	# of pairing	Average size
cd3_cd20	6	9	144	0.454%
cd3_actinsm	2	3	65	0.292%
cd4_cd8	2	3	39	0.518%
cd68_ck18	1	2	25	0.762%
ck18_actinsm	3	4	49	0.534%
foxp3	1	1	14	0.162%
m30	1	1	14	0.176%
mpo	1	1	14	0.178%
chroma_cd56	1	1	19	0.298%
ck5_cd10	2	4	63	0.152%
berep4_ema	1	1	19	0.547%
cd10	1	1	19	0.547%
ck19_ck18	1	1	19	0.547%
d240_ck18	1	1	19	0.547%
maspin	1	1	19	0.547%
mesoth	1	1	19	0.547%
muc5ac	1	1	19	0.547%
muc6	1	1	19	0.547%
pdx1	1	1	19	0.547%
cd4	1	1	10	0.453%
cd8	1	1	10	0.453%
trypsin	1	3	44	0.103%

Table 2.4 – Mixed stains statistics in the dataset [2/2]

Stain	# of image	# of annotation	# of pairing	Average size
muc5ac	9	13	209	0.352%
muc6	9	13	209	0.352%
wt1	10	15	234	0.405%
ca125	10	15	234	0.405%
ki67	9	14	247	0.333%
p16	1	2	48	0.268%
smad4	6	10	172	0.360%
vim	8	12	199	0.344%
p53	18	31	424	0.450%
cd34	9	12	190	0.484%
cald	18	31	424	0.450%
ck19	18	26	437	0.405%
pdl1	7	10	181	0.438%
cd8	4	6	97	0.424%
d240	10	20	253	0.434%
maspin	10	20	253	0.434%
p40	2	3	67	0.355%
ck5	4	7	130	0.239%
muc2	7	11	165	0.339%
muc1	7	11	165	0.339%
p63	4	8	130	0.264%
ck17	14	23	364	0.353%
he	8	13	175	0.521%
ent1	4	6	112	0.507%
ck20	7	11	189	0.306%
ck7	7	11	189	0.306%

Table 2.5 – Single stain statistics in the dataset[1/2]

Stain	# of image	# of annotation	# of pairing	Average size
ceam	11	17	282	0.387%
chra	4	7	128	0.257%
cd56	5	8	147	0.262%
cdx2	7	11	185	0.371%
ca199	13	19	321	0.369%
cd146	10	15	234	0.408%
ngfr	10	15	234	0.408%
cd68	7	10	161	0.446%
cd15	1	2	48	0.300%
hmga2	7	11	180	0.326%
cd3	8	12	209	0.413%
cd20	6	9	144	0.454%
actinsm	5	7	114	0.430%
cd4	3	4	49	0.502%
ck18	6	8	112	0.594%
foxp3	1	1	14	0.162%
m30	1	1	14	0.176%
mpo	1	1	14	0.178%
chroma	1	1	19	0.298%
cd10	3	5	82	0.231%
berep4	1	1	19	0.547%
ema	1	1	19	0.547%
mesoth	1	1	19	0.547%
pdx1	1	1	19	0.547%
trypsin	1	3	44	0.103%

Table 2.6 – Single stain statistics in the dataset[2/2]

Chapter 3

Theoretical background

In the *theoretical background* chapter, the different technical contents required to understand this work are developed. The first three sections describe general purpose or academical contents whereas the last gets deeper in the specific state of the art techniques/models used.

3.1 Computer vision

In this section, several generalities about how to deal with digital images, transformations and other computer vision tools will first be explained. Then the two main frameworks of the registration, pixel-based and feature-based, will be presented.

3.1.1 Image representation

A picture is like a glimpse of the world. However, whereas the world is continuous, it is not affordable to represent such a glimpse continuously on digital memory. The solution to settle this issue is thus to limit an image to a set of points organized as a matrix, each point having its own value.

The following problem is to represent colors. The usual way to represent physically a color is through its wavelength (roughly from 380nm to 750nm for the visible spectrum). But the colors perceived are a mixing of several wavelengths at their own intensities, which is again continuous. The idea this time is to mimic the human eyes. The human eyes possess three types of cones whose responses to wavelengths vary. Those three responses are commonly simplified to red, green, and blue. There are many other ways to estimate colors through colorspaces [19] but the RGB one is the most common, and most cameras are designed to perceive colors through those three color channels with Bayer filter [20]. The image is now represented by a tri-dimensional matrix where the two first dimensions are the width and the height and the third represents the different color channels (Fig 3.1).

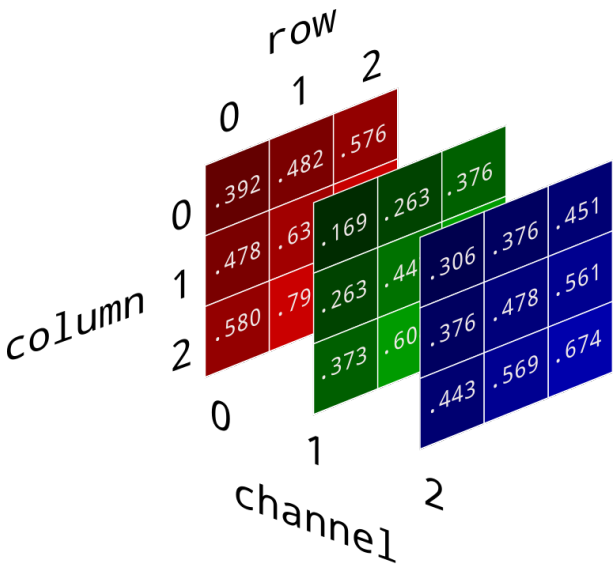


Figure 3.1 – RGB representation of an image in a tri-dimensional matrix. The depth is used to store the different color components.
Credits : Diane Rohrer

Afterwards, one may want to convert an RGB image to a grayscale image, thus turning the three color channels into a single one. Such a conversion may be useful in image processing for several reasons. The main ones are that it is enough to encompass the useful information, to identify features such as edges for example, and that it avoids to treat less useful information, focusing on what is important. Doing this leads to a significant gain in complexity (for the task as well as for the time). Grayscale can be seen as a representation of the luminance, which is itself roughly the brightness of a pixel. As the perception for red, green, and blue of the brightness is not equal at equal intensities, the following weighted average has been empirically found by researchers :

$$Grayscale = 0.2126 \cdot Red_{lin} + 0.7152 \cdot Green_{lin} + 0.0722 \cdot Blue_{lin} \tag{3.1}$$

However, there is another factor. The eye is more sensitive to variation in low intensity. In order to make a better usage of the memory allocated to a color, they are usually stored using gamma compression [21]. To apply the formula above, one should first revert the gamma compression process through gamma expansion to recover the linear values of the three channels. The non-linearity of the process makes it quite heavy, so most applications prefer to use the *ITU-R 601-2 luma transform* (3.2) which is a linear approximation applied directly to the pixels RGB values.

$$Grayscale = 0.299 \cdot Red + 0.587 \cdot Green + 0.114 \cdot Blue \tag{3.2}$$

Finally, the last kind of image used in this work is the binary mask. In this case the image contains a single channel (as for grayscale), but the values are either 0 or 1. Such images are often used for segmentation purpose, where the pixels corresponding to the object of interest in another image are represented by $1's$ in the binary mask while irrelevant pixels are set to $0's$ (Fig 3.2).

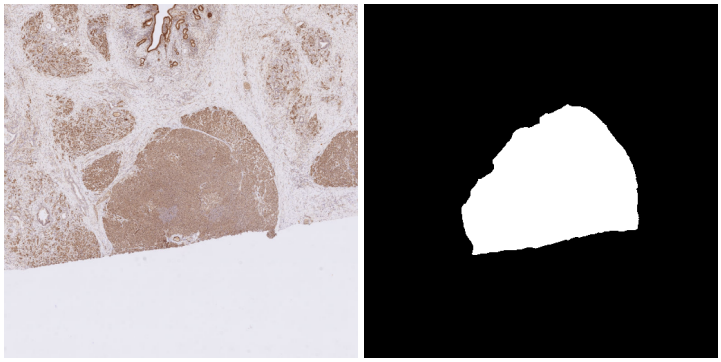


Figure 3.2 – Example of an image and a mask highlighting an object/a region of interest

3.1.2 Coordinate systems & Transformations

In geometry, points are represented through their (x,y) coordinates in a cartesian system (Fig 3.3). This system is unfortunately not sufficient to perform efficiently transformation of space. A new representation, the *homogeneous coordinates*(3.3), is introduced to solve that issue.

$$(x, y) \equiv (x, y, 1) = (\lambda x, \lambda y, \lambda) \tag{3.3}$$

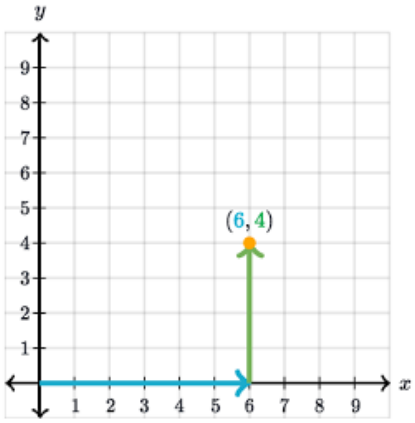


Figure 3.3 – Cartesian coordinate system
Credits : Khan Academy

With the homogeneous coordinates, one can easily represent any transformation from a point $\mathbf{x}$ to a new point $\mathbf{x}'$ through a 3x3 matrix (for 2D images) which will multiply the original point $\mathbf{x}$ (3.4). This process can be repeated as much as wanted. So if multiple successive transformations $\mathbf{T_1}$, $\mathbf{T_2}$ and $\mathbf{T_3}$ have to be performed, one may simply compute $\mathbf{T} = \mathbf{T_3} \cdot \mathbf{T_2} \cdot \mathbf{T_1}$ (with $\mathbf{T_1}$ the first to be applied) and directly multiply by the matrix $\mathbf{T}$ obtained instead of doing 3 vector-matrix multiplications for each point to transform.

$$\mathbf{x}' = \mathbf{T} \cdot \mathbf{x} \tag{3.4}$$

The main category of transformations is the affine transformation, which preserves the parallelism and the linearity of the lines. There exist five sub-categories within affine transformations.

The first one is the translation (3.5). As it does not depend on the current position of the point, it will use the trailing 1 at the end of the homogeneous coordinates to impact the values of the transformed x' and y'.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.5}$$

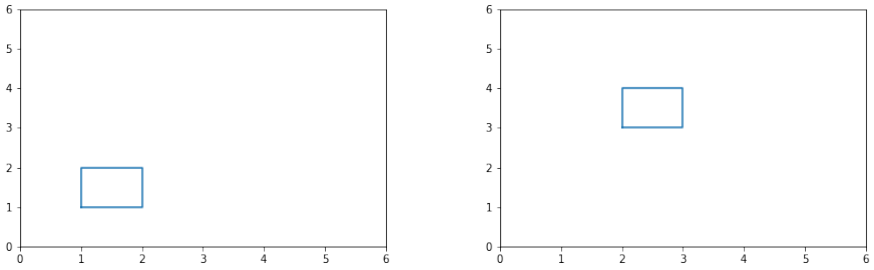


Figure 3.4 – Example of translation with $t_x = 1$ and $t_y = 2$

The following one is the reflection (3.6) where a symmetry around the axis x and/or y is performed. The idea is simply to take the opposite value for each x and/or y.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.6}$$

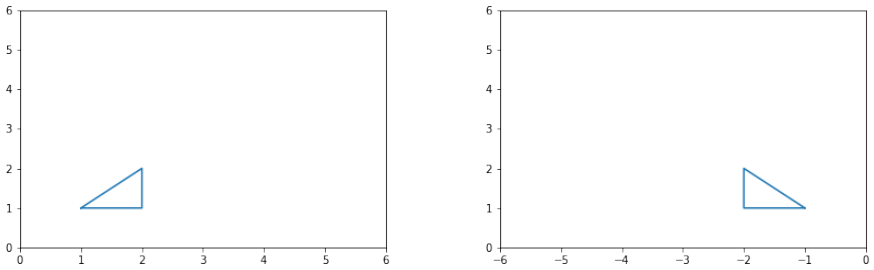


Figure 3.5 – Example of reflection (for x values i.e. around y axis)

The third one is the rotation (3.7). This time a rotation of angle θ is performed around the point (0,0). To perform a rotation around another center, one must combine the

rotation with 2 translations (one before to center the object on that center and one after to go back to the initial frame of reference).

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.7}$$

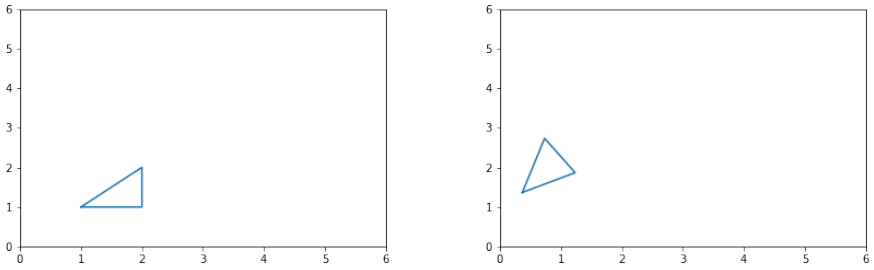


Figure 3.6 – Example of rotation with $\theta = 30°$

The fourth one is the scaling (3.8) where every coordinates are multiplied by a given factor. This means that, for a scaling of 2, the objects will be twice larger and will be twice as far as they were before from (0,0). When $s_{11} = s_{22}$, it is called a uniform/isotropic scaling. When they are not equal it is called a non-uniform/anisotropic scaling.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_{11} & 0 & 0 \\ 0 & s_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.8}$$

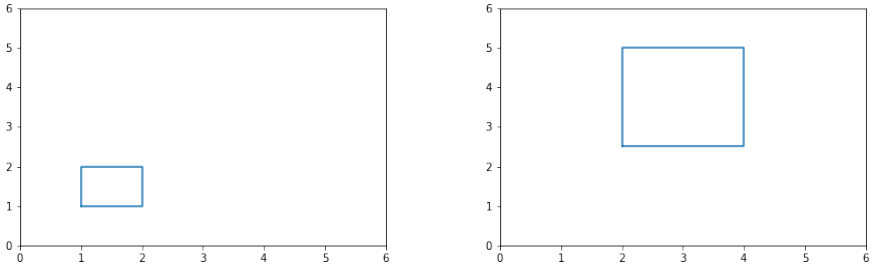


Figure 3.7 – Example of scaling with $s_{11} = 2$ and $s_{22} = \frac{5}{2}$

The last one is the shearing (3.9). This affine transform is the one responsible for the change of angle after transformation (whereas parallelism is always conserved).

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & s_{12} & 0 \\ s_{21} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.9}$$

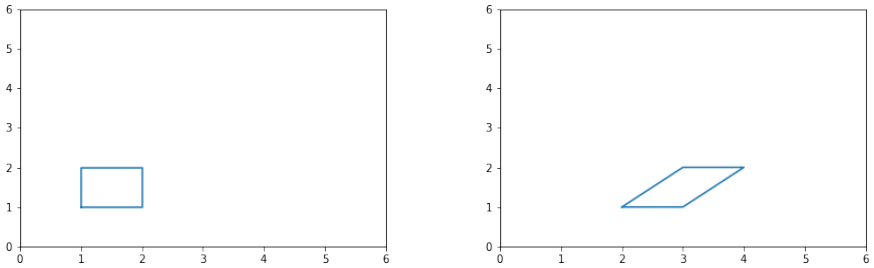


Figure 3.8 – Example of scaling with $s_{12} = 1$ and $s_{21} = 0$

In general, any affine transform is defined as $\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$, and any multiplication of such matrices keeps this form. This means that any application of affine transform to an homogeneous point will leave its trailing 1 unchanged.

There however exists another type of transformation known as *homography*, also known as projective transformation, where the two 0's at position T_{31} and T_{32} may be non-zero. In this case the trailing 1 is affected and each transformed point needs to be normalized with a trailing 1 to recover the cartesian coordinates. This kind of transformation basically encompasses the possible projections of a plan over another one (Fig 3.9).

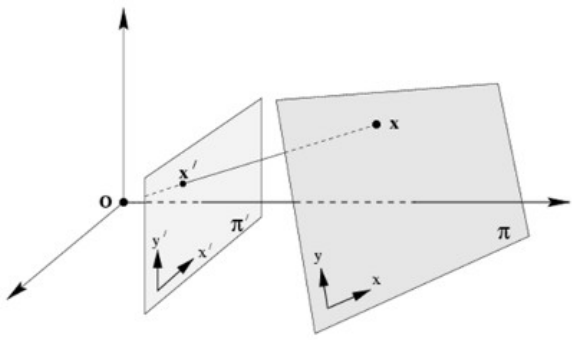


Figure 3.9 – Homography example
Credits : OpenCV, Basic concepts of the homography explained with code

The last remark concerning coordinate systems is about how images are stored on computer. While cartesian system is commonly used in geometry, computers (and many image processing applications) index the images differently. Instead of having the (0,0) at the bottom left of the image, they chose to start the (0,0) at top left corner instead. This is done because screens start the rendering from the top left. When dealing with several sources using both systems, some conversions may be needed to swap from one to another. The transformations needed in this case are a reflection for y values followed by a translation again for y of the image height. Such transform is shown in (3.10) and is its

own inverse, meaning that it is applied for swapping in both directions.

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & \text{HEIGHT} \\ 0 & 0 & 1 \end{pmatrix} \tag{3.10}$$

Additionally, when two images of different heights are superimposed, it also means that the non-overlapping region is different as the (0,0) are matched. In the cartesian system case it is the top part of the highest image that does not overlap while in the computer system case it is the bottom part.

3.1.3 Morphological operations

The mathematical morphology is a theory and technique for the analysis and processing of geometrical structures, based on set theory [22]. Even though it is applicable to grayscale images, this section will limit to binary images (such as binary mask presented in the end of Section 3.2).

The two main basic operators are the *dilation* and the *erosion*. The rough idea is to modify the shape of a polygon by applying another shape, the structuring element B (which is often a disk), at its frontier. In other words, the structuring element is pasted with its center on the frontier of the original shape, for all points on this frontier. If the operator is the dilation, the additional area covered by all those pasting is added to the shape. For the erosion, every area covered by the pasting which is part of the original shape should be removed from it. Those explanations may not be totally accurate in the case of non-symmetric structuring element but such structuring element will not be needed.

Mathematically, the dilation is written as :

$$A \oplus B = \bigcup_{b \in B} A_b \tag{3.11}$$

where A_b is the original shape translated by a vector b, which is a vector starting from (0,0) and whose endpoint is included in the structuring element (considered itself centered at (0,0)). So here it is defined as the union of all translations of the original shape where the vector of translation is limited to the structuring element. This has the exact same effect than the explanation given above (for a symmetric structuring element).

The erosion is defined in the same fashion as this :

$$A \ominus B = \bigcap_{b \in B} A_{-b} \tag{3.12}$$

This time it is defined as the intersection of all translations of the original shape where the vector of translation is limited to the structuring element (actually the opposite vector but it is meaningless for a symmetric structuring element).

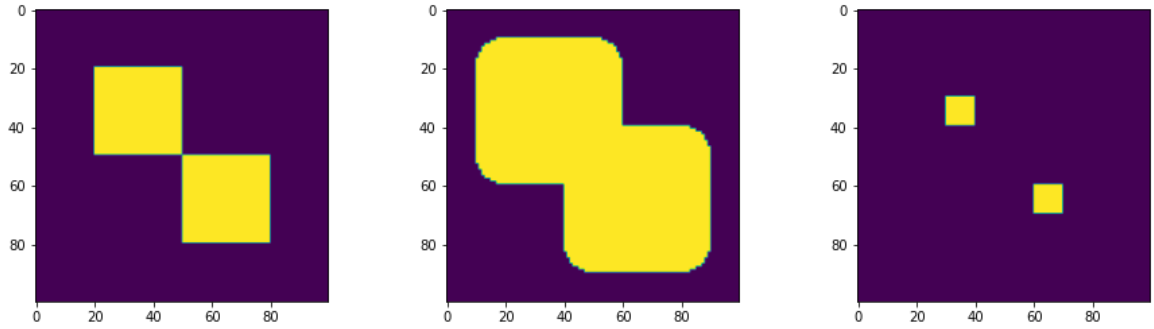


Figure 3.10 – Example of an arbitrary mask (on the left) and its dilation/erosion (respectively on the middle and on the right) by a disk of radius 10

Once those two basic operators are defined, the two others can be directly derived from them. The first is the *closing* (3.13) which is a dilation followed by an erosion, using the same structuring element. The main effects of the closing, with a disk, is to absorb small holes and to smooth interior corners. The second one is the opening (3.14) which is an erosion followed by a dilation, again using the same structuring element. For this one, the effect will be to remove thin parts of the shape and to smooth exterior corners. The effect of smoothing can be combined by applying successively both operators but the order of application may impact the final result.

$$A \bullet B = (A \oplus B) \ominus B \tag{3.13}$$

$$A \circ B = (A \ominus B) \oplus B \tag{3.14}$$

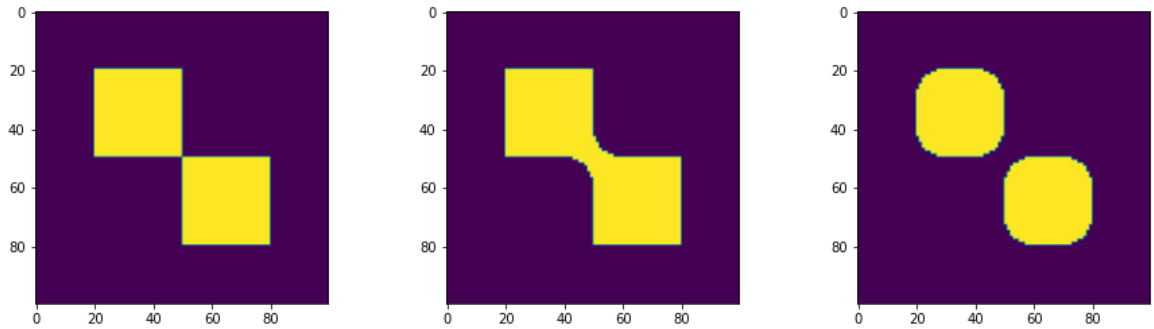


Figure 3.11 – Example of an arbitrary mask (on the left) and its closing/opening (respectively on the middle and on the right) by a disk of radius 10

3.1.4 Histograms

An interesting measure in image processing is the *histogram*. The histogram of an image is a 1D function that counts the number of times each intensity value is present in an image

(considering pixel values discretized) as in Fig 3.12. One can also compute a histogram that counts intensity values included in several ranges that cover the whole intensity space. Those ranges are often called *buckets* or *bins*.

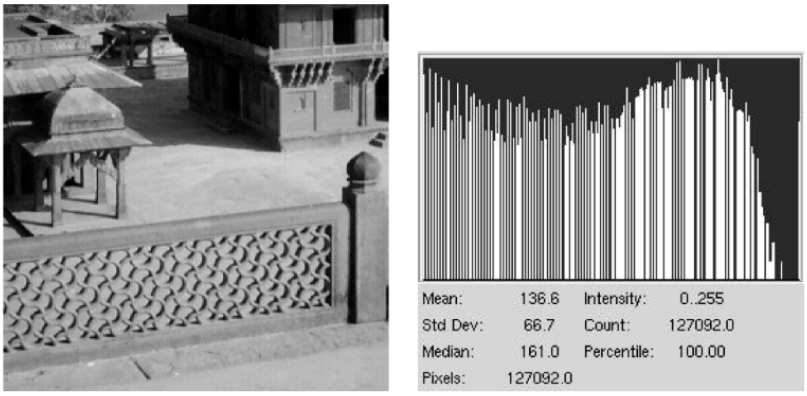
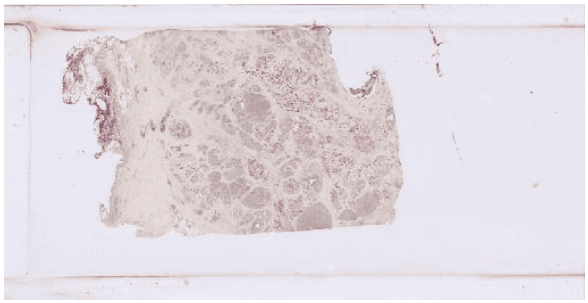


Figure 3.12 – Example of an histogram of an image (256 bins). The occurrence of each intensity in the image on the left is represented on the bar plot on the right. [23]

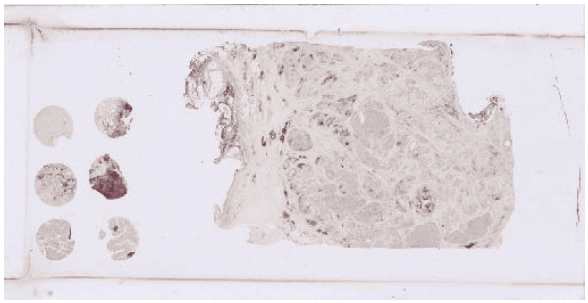
When two images are compared, the concept of histogram can be extended to *joint histogram*. This time all pixels sharing the same location in both images are matched by pair, then the joint histogram is built as a bi-dimensional input function where the first axis represents the bin in which the pixel from the first image fall and the second axis the bin in which the pixel from the second image fall. So two pixels sharing the same value in the first image will always fall in the same bin in the marginal histogram. However, they may not be in the same bin in a joint histogram with a second image.

3.1.5 Registration

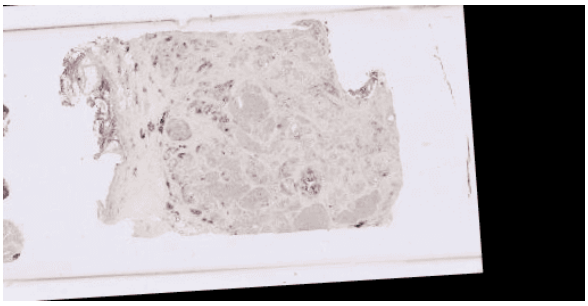
The *image registration* is simply "the process of spatially aligning two images of a scene/object so that corresponding points assume the same coordinates" [23]. So registration deals with pairs of images. The first one, the reference, is called the *fixed* image. The fixed image is the one that will never move during the process of registration. The other one is called the *moving* image and is the one that will be affected by the transform obtained. There are two main families of registration algorithms : the pixel-based and the feature-based.



(a) Fixed image



(b) Moving image



(c) Moving image registered to fixed image

Figure 3.13 – Example of a registration. The moving image (on the right) is translated and rotated such that it is aligned with the fixed image (on the left).

Additionally, the notion of *pattern matching* can be introduced as the process of finding an instance of an object in a more global scene. While the application may be slightly different, the same methods can be applied to both problems.

3.1.6 Interpolations

When a transformation is applied to an image, it is important to remember that the resulting image will still be stored in a rectangular matrix. There are very few transformations which will keep all the points of the original image in the rectangular grid after applying them (translation of a whole number, reflection and rotation of a multiple of 90°). Most of the time, the points will fall in between four points of the grid (or in the opposite direction, the points of the grid will fall in between the original image grid). To solve that issue, it is needed to have an interpolation method for such points.

The simplest method to estimate the value of such points is to use the nearest neighbor interpolation. As its name says, the principle is to look at the closest neighbor (i.e. the closest point on the grid, which thus has a known value) and to just take the exact same value without considering any other point. As it is easy to implement and fast, it is often a good method when the objective is the rendering. Nonetheless, its discrete nature (a small variation in the transform may not impact at all the result) can be annoying for some applications.

The following method is the bilinear one. This interpolator, very roughly, try to approximate the point by doing a linear interpolation in both dimension. Mathematically it comes as follows.

Considering the four points (x_1, y_1), (x_2, y_1), (x_1, y_2) and (x_2, y_2) surrounding the unknown point (x, y) and the function $f(x, y)$ which is responsible to give the value of a point (x, y) ; the bilinear interpolation can be written as a weighted mean :

$$f(x, y) \approx w_{11} f(x_1, y_1) + w_{12} f(x_1, y_2) + w_{21} f(x_2, y_1) + w_{22} f(x_2, y_2) \tag{3.15}$$

where the four weights w_{11}, w_{12}, w_{21} and w_{22} are found thanks to the following system :

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_1 & x_2 & x_2 \\ y_1 & y_2 & y_1 & y_2 \\ x_1 y_1 & x_1 y_2 & x_2 y_1 & x_2 y_2 \end{bmatrix} \cdot \begin{bmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ y \\ xy \end{bmatrix} \tag{3.16}$$

By solving this system, the following solutions are found for the weights :

$$\begin{cases} w_{11} = \frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} \\ w_{12} = \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} \\ w_{21} = \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} \\ w_{22} = \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} \end{cases} \tag{3.17}$$

Those weights can actually be seen as the area covered by the rectangle "opposite to the point it weights" (over the total area) as shown in Fig 3.14.

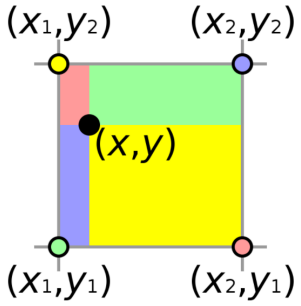


Figure 3.14 – Bilinear interpolation weighting. The areas of the colored rectangles represent the relative weights of the corner point with the same color. [24]

There also exist higher degree interpolators such as the *bicubic*. They produce smoother interpolation but require more points (16 instead of 4 for the bilinear interpolation) and are thus much slower. They will not be covered here but a visual comparison of nearest neighbor, bilinear and bicubic can be seen in Fig 3.15.

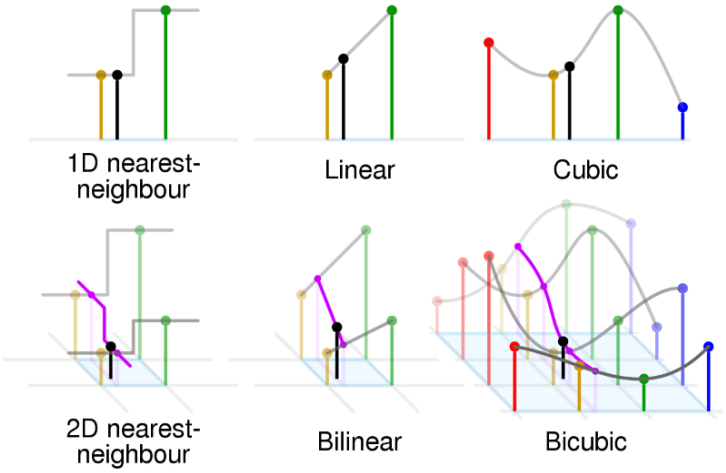


Figure 3.15 – Visual comparison of interpolators (1D and 2D) [24]

The last interpolator that will be covered is the *Lanczos resampling*. It is not exactly an interpolator (even though it could be used as one) in the sense that it is more designed for resizing image. The Lanczos kernel acts as a low pass filter by using the Lanczos window. This window will affect the range and the magnitude at which a sample on the grid will impact the value to interpolate. It is mathematically designed as a normalized *sinc* function (3.18) shown in Fig 3.16 where a is the size of the window.

$$L(x) = \begin{cases} sinc(x)sinc(x/a) & \text{if} -a < x < a \\ 0 & \text{otherwise} \end{cases} \tag{3.18}$$

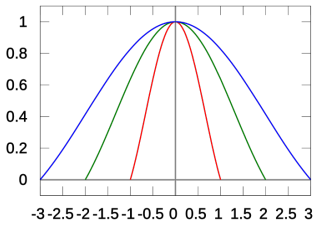


Figure 3.16 – Lanczos window of different sizes ($a = 1, 2$ and 3) [25]

The interpolated/resampled values are then constructed from all neighboring samples s_i (convoluted with the Lanczos window). The equation (3.19) is a simplification as all samples are considered but in practice only the samples close enough to not be zeroed-out by the window are considered for a location x.

$$f(x) = \sum_{s_i} f(s_i) \cdot L(x - s_i) \tag{3.19}$$

The application in two dimensions is straightforward as $L(x, y) = L(x)L(y)$.

3.1.7 Pixel-based registration

The pixel-based approach is an iterative approach to the problem. It consists in finding a registration where the pixels supposed at the same location agreed the most on a given measure. The general framework can be seen in the Fig 3.17.

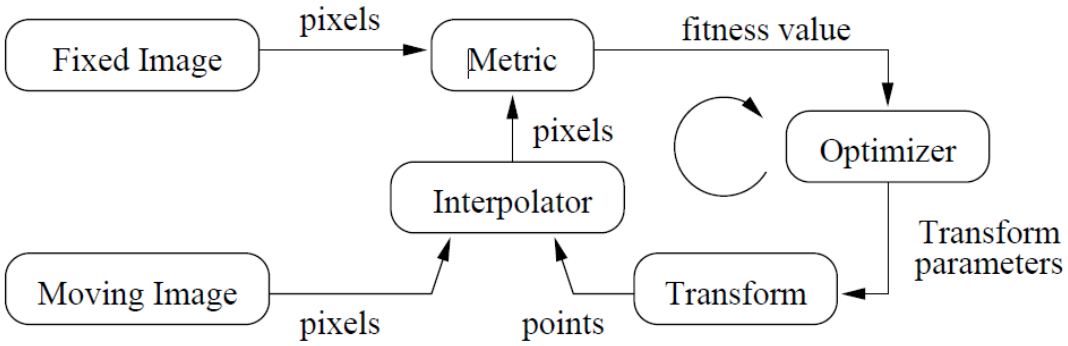


Figure 3.17 – Typical metric-based registration framework. The moving image is transformed with the actual transformation parameters and compared with the fixed image through a similarity/dissimilarity metric. The transform is then optimized according to the result on the metric. [26]

It includes four main components. The transform (Section 3.1.2) and the interpolator (Section 3.1.6) have been reviewed in previous sections. The two remainings are the *metric* and the *optimizer*.

The metric is what will allow to assess the quality of a registration. It can be a pixel-wise comparison or a more complex measure. A simple example of pixel-wise comparison is the *square L2 norm*. In this case, all pixels that are supposed at the same location (after transformation of the moving image) are compared through a square error of their respective intensities, and the sum for all pairs of pixels is computed. There exist two categories of metrics : the similarity and the dissimilarity measures. The only difference between them is that the first should be maximized while the second should be minimized (though the measure can be moved to the other category by multiplying with -1). The L2 square norm is typically a dissimilarity measure as we want this error to be as low as possible such that most pixels at same locations have roughly the same intensity.

The last component is the optimizer, also called search algorithm. It is responsible to generate the next transformation(s) that will help to find the extremum of the metric. The main difference here between algorithms is the properties that will be required from the metric used.

The simplest method, the exhaustive search, does not even care about the value of the metrics to progress in the algorithm. It predetermines at which position it will sample the transformation parameters and then picks the best one. Then there is the 0^{th} order methods that make use of the value of the metrics. There are plenty of methods in this category such as the *coordinate-search* method that samples close values at a given step

distance (within the parameter space), and change to the minimal value if it is smaller than the current one or reduce the step length until it is too short. There is also the *simulated annealing* [27] which mimics the annealing in metallurgy. Here a random neighbor is picked and is accepted with a probability proportional to how better (or worse) is the new value and how long the algorithm has been running (the later, the less a worse value is likely to be accepted).

Then there are higher order search algorithms. the higher the degree, the higher derivative the algorithm will require. In the category of the first order method, one of the most popular is the *gradient descent* which, at each iteration, take a step in the parameter space in the direction of the gradient (of the metric with respect to the parameters) proportional to this gradient and a given constant called *learning rate*. There also exist variants of the gradient descent as well as higher order method such as the Newton method for the second degree.

As most pixel-based methods progress by looking at variation of the metric in the close neighborhood, the choice of the interpolator may be important. The nearest neighbor interpolator may keep the exact same result for small variation which can be problematic for some search algorithms, It is thus preferable to select a bilinear or a bicubic interpolator for those kinds of applications.

To enhance (and speed-up) pixel-based registration, a good practice is to perform it in multi-resolution. First the images are registered at low resolution to focus on coarse details and to compute the metrics for less points. Once it is done, the process of registration restarts at a greater resolution but starting from the transformation obtained at the previous step, up to the original resolution.

A last point, when performing pixel-based registration, is that one must always pay attention that the metrics are evaluated on overlapping parts of the image. This means that a step too large in a search algorithm may completely suppress the overlapping, thus making the algorithm unable to continue and find a solution.

3.1.8 Feature-based registration

The feature-based registration can be decomposed in four main successive steps : the *location*, the *description*, the *matching* and the *fitting*.

The first step, the location, aims at finding points of interest in the image, called *keypoints*. Such interesting points can be edges, corners, blob[1] centers and so on. This operation is performed separately on both images (see Fig 3.18). It is important that the detection is invariant to transformations (as most as possible) to have a higher chance to detect a common keypoint in both images.

[1] A blob is a group of pixels sharing some properties such as a similar intensity value.

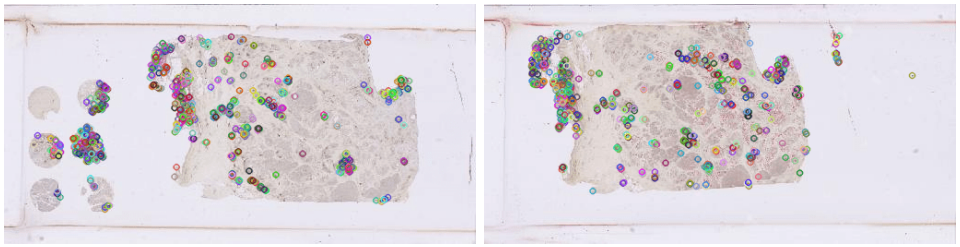


Figure 3.18 – Example of feature detection. The points represent the different keypoints identified by the feature detector in both images, but no link between them is done at this stage.

Once the interesting keypoints are found, they need a descriptor. The descriptor is a vector that will describe the keypoints detected at the previous step. Two main properties are required for these descriptors. They need to be transformation invariant as well, such that similar keypoints in both image are described the same way, but they also need to be as discriminative as possible such that a keypoint in an image does not have many matches in the other.

Once those descriptors are computed, they need to be matched as stated above. This is often done using a brute-force matcher that will simply compare a descriptor from the first image to each descriptor in the other and find the best match(Fig 3.19). An additional option, the *cross-check*, is also possible. It will ensure that, considering descriptor a in first image and descriptor b in second image, b is the best fit from the second image for a but a is the best fit from the first image for b as well. If not, the match is discarded. This method allows to remove a good proportion of false matches while keeping enough true matches for the last steps. Regarding the comparison measure, it may depend on the nature of the descriptor. A binary descriptor will probably use the Hamming distance while a continuous one could use the norm of the vector difference (L1 or L2).

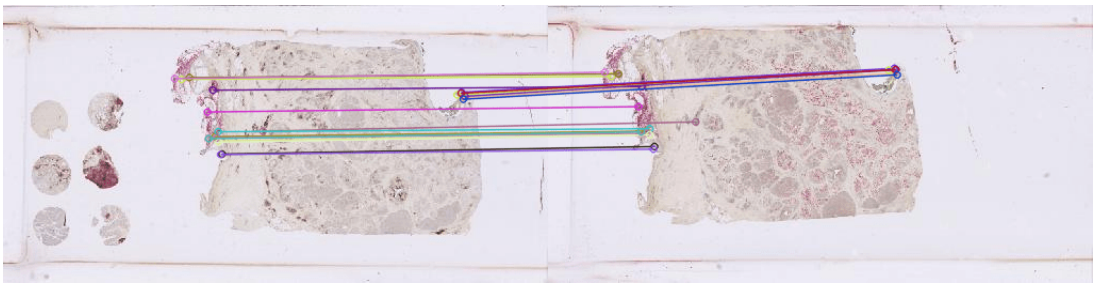


Figure 3.19 – The 20 best matches between keypoints, by comparison of their descriptors, are shown here. In practice, much more than 20 keypoints are matched together.

The last step after matching the keypoints is to find the transformation parameters that explain the more accurately those matches. The technique often used for this task is *Random Sample Consensus* (RANSAC) [28]. The idea is quite simple. The minimal number of matches[2] that allows to compute uniquely a transform is taken randomly from

[2]This number will depend on the kind of transform selected i.e. dependent on its degree of freedom.

the matches pool. This transform is applied to each point of the moving image and then the "agreement" is computed. A pair of points will agree to the transform if they are close enough (given a residual threshold) after registration and will be considered as *inlier*, while pairs that disagree will be considered as *outlier*. This process is repeated as many times as affordable and the transform with the most inliers is kept. The process can also be stopped if it is highly confident that it has encountered a good model. For this confidence, RANSAC looks at how many iterations it has already performed and what is the current ratio of outliers with respect to the number of points. If the probability of selecting only inliers (in the set used to compute uniquely the model) with such outliers ratio and such number of trials is higher than a given probability, often 99%, then the iteration stop and the best model is kept as well. Note that considering a smaller outliers ratio after the same number of iterations gives a even higher probability as it is "easier" to pick only inliers with this hypothesis. So the confidence is not only about the probability to find a model a good as the current one but also any better models.

This algorithm is likely to not consider the remaining false matches. It also highlights the importance of having well placed keypoints in the location step as the final transform will only depend on a few points with RANSAC[3].

[3]Some version, including the OpenCV one, also perform a refinement of the model by doing a few iterations of the Levenberg–Marquardt algorithm.

3.2 Information theory

This small part will introduce the essential components of information theory to understand the notion of mutual information. This notion will be use as a similarity metric for the pixel-based registration (Section 3.1.7).

3.2.1 Entropy

The entropy is the *unknowness* of an event, how uncertain the outcome is. Considering a random variable $\mathcal{X}$ with n possible outcomes and their respective probability $P(x_i)$, the entropy is defined as :

$$H(\mathcal{X}) = -\sum_{i=1}^{n} P(x_i) \log_2 \left(P(x_i)\right) \tag{3.20}$$

As an example, $\mathcal{X}$ can represent a coin toss in which case the probabilities are $P(x = \text{head}) = P(x = \text{tail}) = \frac{1}{2}$. In that case, the entropy is :

$$H(\mathcal{X}) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) \tag{3.21}$$

$$= 2\left(-\frac{1}{2}\cdot(-1)\right) \tag{3.22}$$

$$= 1 \tag{3.23}$$

Actually, it is the highest value that can be obtained with a binary random variable as can be seen in Fig 3.20. It is quite legitimate as the uniform distribution is the one on which we can make the less hypothesis on the outcome. This reasoning is valid as well for random variable with more than two possible outcomes.

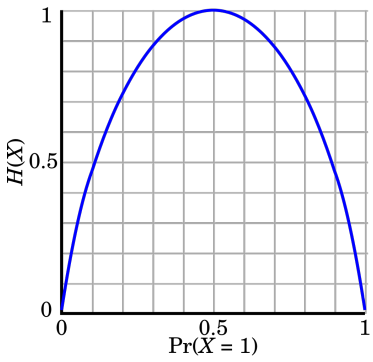


Figure 3.20 – Entropy of a binary event. The entropy is maximum when the event is the more uncertain i.e. when the probability distribution is uniform. [29]

3.2.2 Joint/conditional entropy and mutual information

In probability, there is often more than one random variable involved and they may not be completely independent. It is thus interesting to have a look at the entropy shared by

several random variables. Several notions are then derived from the entropy. First there is the *joint entropy* $H(\mathcal{X}, \mathcal{Y})$ which is the total entropy encompassed by both variables $\mathcal{X}$ and $\mathcal{Y}$. Then there is the *conditional entropy* $H(\mathcal{X}|\mathcal{Y})$ that depicts the portion of the entropy of one of the random $\mathcal{X}$ conditionally independent of the other random variable $\mathcal{Y}$. And finally the *mutual information* $I(\mathcal{X}; \mathcal{Y})$ which is the entropy shared by $\mathcal{X}$ and $\mathcal{Y}$. There are some direct relationships between those notions. For example, the entropy of $\mathcal{X}$ is the sum of the entropy unique to $\mathcal{X}$ (the conditional one) and the entropy that is shared with the other variable $\mathcal{Y}$ (the mutual information). Such relationships can be seen through a Venn diagram shown in Fig 3.21.

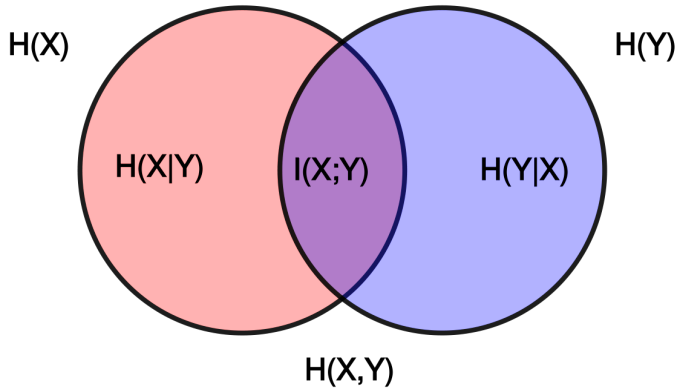


Figure 3.21 – Venn diagram of two random variables. The entropy of an event can be decomposed into the "entropy" shared with another event, the mutual information, plus the entropy that is conditionally independent of that second event. [30]

One way to write the mutual information, which will be interesting for later, is the following :

$$I(\mathcal{X}, \mathcal{Y}) = \sum_{x_i \in \mathcal{X}, y_j \in \mathcal{Y}} P(x_i, y_j) \log_2 \left(\frac{P(x_i, y_j)}{P(x_i) P(y_j)} \right) \tag{3.24}$$

3.3 Deep Learning

This section will first introduce the field of machine learning and the notions associated (training set, bias and variance, ...). Then it will focus on neural networks, deep neural networks and finally convolutional neural networks.

3.3.1 Machine learning

The *machine learning* is a part of the the artificial intelligence field. It encompasses the algorithms that make use of a set of data, the *training* data, to learn a task. The training data contains two components. There are the *inputs/features* which are the part of the data used to make a prediction, and the *output/label/ground truth* which is the prediction that should be made.

The kind of task is also often divided in 2 categories : the *classification* and the *regression*. The former, the classification, aims at determining whether a sample belongs to one class or another (there can be more than two possible classes). For example, one can try to classify a set of pictures to determine if a cat is present on the picture. The latter, the regression, aims at predicting one of several continuous values. A simple instance is the estimation of a function from a set of experimental points. Here, from a set of (x_i, y_i) points where the x_i's are the inputs (which may be multidimensional) and y_i's are the outputs, the algorithm will try to predict a new value y for a given x whose value has not been observed empirically beforehand (Fig 3.22).

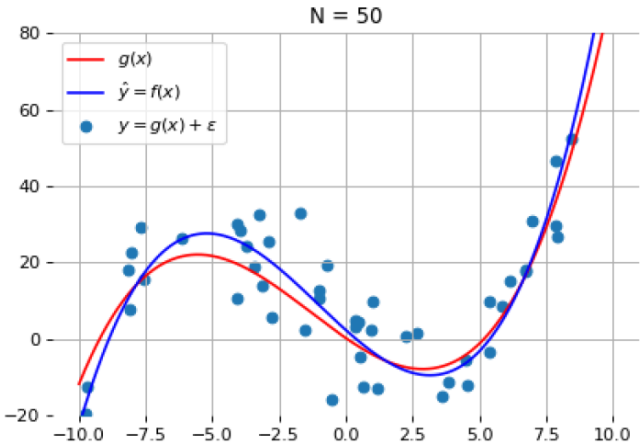
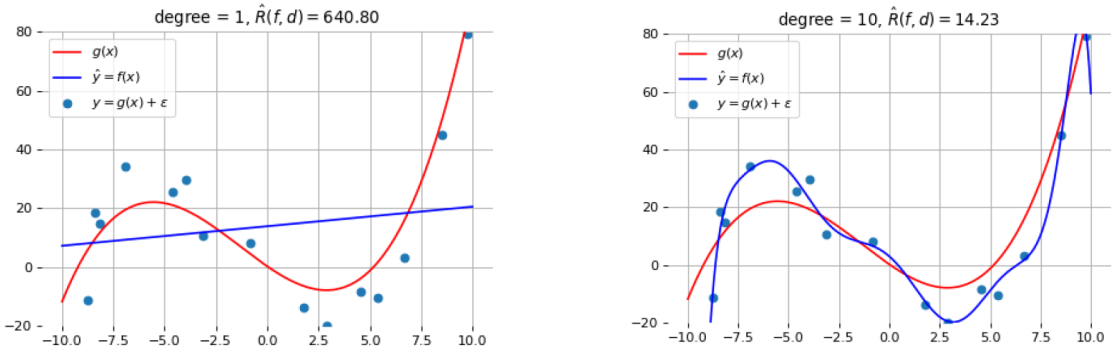


Figure 3.22 – Estimation $\hat{y}$ of a function $g(x)$ from a set of 50 empirical points $g(x) + \epsilon$ [31]

3.3.2 Under-fitting and Over-fitting

Under-fitting and over-fitting often appear when the models used are not appropriate. Under-fitting appears when the model used is too weak compared to the "function" it tries to estimate (Fig 3.23a). On the other hand, over-fitting appears when the model is too complex. In this case, the model will fit the given data very well but the generalization

will be worse as the model has put too many effort in fitting the known samples (Fig 3.23b).



(a) Under-fitting

(b) Overfitting

Figure 3.23 – Estimation $\hat{y}$ of a function $g(x)$ of degree 3 by a function of (a) degree 1 (b) degree 10. The first model can clearly not represent the function while the second gives a too complex solution to explain the empirical points. [31]

This problem can be represented by the bias-variance error tradeoff. The bias can be seen as how far the estimating model is from the true model given any set of training data. A high bias thus represents the under-fitting phenomenon as the chosen model can not encompass the complexity of the training set, whatever it is, because the complexity of the model is too low. The other parameter, the variance, is an indication of how dependent from the data is the model. Indeed, the high complexity of the model will fit extremely well the training data and thus varying very much from one training set to another. This time the phenomenon linked with a high variance is the over-fitting as the model fits too much the data. Those are the two main sources of error while training data[4]. The tradeoff comes from the complexity of the model chosen that will decrease (resp. increase) the bias and increase (resp. decrease) the variance as the complexity increases (resp. decreases) as shown in Fig 3.24. The only way that really allows both bias and variance to decrease is to increase the number of samples in the training set.

[4]The third source of error is the noise of the data, but the only way to reduce it is to collect samples in a more accurate way or to filter out the training set before using it. This is thus not really dependent on the choice of the model.

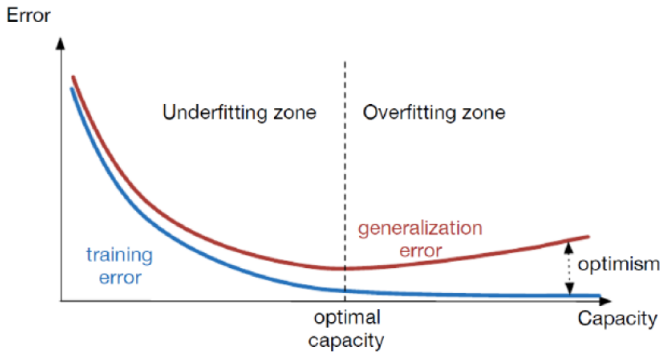


Figure 3.24 – Equilibrium between under-fitting and over-fitting. In the ideal case the algorithm ends in between the two zones where the generalization error is the lowest. [31]

3.3.3 Neural networks

The neural networks are one type of algorithm in the machine learning field. The most inner component of a neural network is the *perceptron*. The principle of the perceptron is actually very close to the behavior of a real neuron (Fig 3.25). Some stimuli enter the neuron and, depending on those inputs, may itself produce a stimulus for the next neurons.

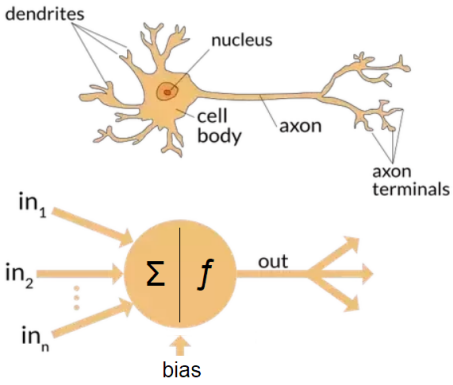


Figure 3.25 – Similarity between a perceptron and a neuron. Several signals enter the neuron and it can react or not by sending signals to other neurons. [32]

The first type of perceptron $\sigma(x)$ was indeed built from a sign function where "enough inputs" leads to a unit signal transferred (3.25). The function σ is called the *activation function*.

$$\sigma\left(b + \sum_i w_i x_i\right) = \begin{cases} 1, & \text{if } (b + \sum_i w_i x_i) \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.25}$$

where b is the bias, x_i the different inputs and w_i their respective contribution to the decision. Among them, the bias b and the weights w_i are the parameters of the perceptron and will dictate the way it reacts to a set of inputs.

The simplest neural network, the multi-layer perceptron (MLP) [33], is just a succession of several layers composed themselves of several perceptrons. The more perceptrons/layers are used, the more complex the model will be. In the case of a function estimation, the size of the input layer is the number of parameters of the function and the output layer is of size one and should not be squeezed through a $sign()$ function.

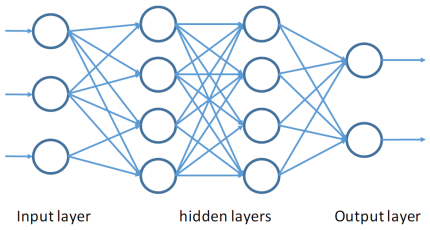


Figure 3.26 – Example of a simple MLP architecture composed of 3 inputs, 2 outputs and 2 hidden layers with 4 neurons each. [34]

3.3.4 Training

With the current structure, the expectation is to inject the input value x at the beginning of the MLP and to have the appropriate y value at the output after computing the value of the different intermediate perceptrons. This is called the *forward propagation*. However, this result depends on all the biases and the weights of those perceptrons. Those weights and biases are indeed the *parameters* ($\boldsymbol{\theta}$) of the MLP and will dictate how it will respond to an input. They unfortunately, and obviously, do not start with the appropriate values for the estimation of an arbitrary function.

There is thus a need of a training procedure to determine the value of those parameters. The first step to do so is to distinguish a good output from a bad one. This is done through a *loss function* $L()$. This function compares the output of the network with the expected output and grows as the output is worse compared to the expectation. The goal then becomes to minimize the value of this function with respect to parameters ($\boldsymbol{\theta}$) of the network.

There are many ways to optimize a function but the one of interest here is *gradient descent*. The idea is to find the best step $\boldsymbol{\epsilon}$ to take (in the parameter space) using the first derivative of the function with respect to those parameters $\boldsymbol{\theta}$ around the current location $\boldsymbol{\theta}_0$. The function $L()$ is thus express locally with $\widehat{L}()$ (3.26) where the last term is a penalty that increases as the size of the step increases. λ is a constant controlling roughly how far the step is allowed to go.

$$\widehat{L}(\boldsymbol{\epsilon};\boldsymbol{\theta}) = L(\boldsymbol{\theta}_0) + \boldsymbol{\epsilon}^T \boldsymbol{\nabla}_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_0) + \frac{1}{2\lambda} \|\boldsymbol{\epsilon}\|^2 \tag{3.26}$$

The best step $\boldsymbol{\epsilon}$ to take is thus the minimum of $\widehat{L}(\boldsymbol{\epsilon};\boldsymbol{\theta})$ with respect to $\boldsymbol{\epsilon}$. From this simple expression, the value of the minimum can be written as a closed expression by finding where the gradient with respect to $\boldsymbol{\epsilon}$ is equal to 0.

$$\begin{aligned} \nabla_{\boldsymbol{\epsilon}} \widehat{L}(\boldsymbol{\epsilon}; \boldsymbol{\theta}) &= 0 \\ &= \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_0) + \frac{1}{\lambda}\boldsymbol{\epsilon} \\ \Rightarrow \qquad \boldsymbol{\epsilon} &= -\lambda \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_0) \end{aligned} \tag{3.27}$$

From (3.27), an iterative formulation to find the values of $\boldsymbol{\theta}$ to minimize $L()$ can be easily derived :

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) \tag{3.28}$$

The constant λ is called the *learning rate* and a low learning rate means that the penalty grows quickly which favors low step, thus leading to slow evolution of $\boldsymbol{\theta}_t$ (i.e. a slow learning).

Now, this algorithm has to be applied to the MLP introduced before. The important property to notice from the gradient descent algorithm is the fact that the loss $L()$ has to be differentiable with respect to the parameters $\boldsymbol{\theta}$. The first condition implied by this is the direct differentiability of the loss $L()$ with respect to the output $\hat{y}$ predicted by the neural network. A very simple example of loss function for regression (e.g. a function estimator) is square error which is differentiable :

$$\begin{aligned} L\left(\hat{y}(x_i; \boldsymbol{\theta}), y_i\right) &= \frac{1}{2}\left(\hat{y}(x_i; \boldsymbol{\theta}) - y_i\right)^2 \\ \Rightarrow \quad \frac{\partial}{\partial \hat{y}} L\left(\hat{y}(x_i; \boldsymbol{\theta}), y_i\right) &= \hat{y}(x_i; \boldsymbol{\theta}) - y_i \end{aligned} \tag{3.29}$$

To continue the differentiation, a simple case where the final output is a single neuron whose computation is $\hat{y} = a^{(l)} = b^{(l)} + \sum_i w_i^{(l-1)} a_i^{(l-1)}$ will be considered for the sake of simplification. In this formula, l is the number of the current layer (the last in this case), $a_i^{(l-1)}$ are the responses of the neurons in the previous layer and $w_i^{(l-1)}$ the weights associated to each of them in the last neuron. The derivative of the loss with respect to those parameters of the network can be easily written with the *chain rule* :

$$\begin{aligned} \frac{\partial L}{\partial w_i^{(l-1)}} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i^{(l-1)}} \\ &= \frac{\partial L}{\partial \hat{y}} a_i^{(l-1)} \end{aligned} \tag{3.30}$$

where $\frac{\partial L}{\partial \hat{y}}$ can be computed as in the example (3.29), considering the condition mentioned above is fulfilled. The value $a_i^{(l-1)}$ was obtained previously in the forward propagation.

The derivatives also have to be computed for all neurons in all previous layers.

In general, the responses of a neuron j in layer k can be written as $a_j^{(k)} = \sigma\left(\sum_i w_{ij}^{(k-1)} a_i^{(k-1)}\right)$ where $\sigma()$ is the activation function as introduced in (3.25) and $w_{ij}^{(k-1)}$ is the weight in

neuron j from the layer k associated with the response $a_i^{(k-1)}$ of the neuron i from the previous layer $k-1$. Actually, the value of the loss $L()$ depends on a weight $w_{ij}^{(l-1)}$ only through $a_j^{(k)}$, the derivative can thus be computed using again the *chain rule* as :

$$\begin{aligned}\frac{\partial L}{\partial w_{ij}^{(k-1)}} &= \frac{\partial L}{\partial a_j^{(k)}} \frac{\partial a_j^{(k)}}{\partial w_{ij}^{(k-1)}} \\ &= \frac{\partial L}{\partial a_j^{(k)}} \sigma' a_i^{(k-1)}\end{aligned} \tag{3.31}$$

The last thing to compute is thus $\frac{\partial L}{\partial a_j^{(k)}}$ whose impact on $L()$ occurs through all the neurons $a_i^{(k+1)}$ in the following layer $k+1$. This is also written with the chain rule as :

$$\begin{aligned}\frac{\partial L}{\partial a_j^{(k)}} &= \sum_i \frac{\partial L}{\partial a_i^{(k+1)}} \frac{\partial a_i^{(k+1)}}{\partial a_j^{(k)}} \\ &= \sum_i \frac{\partial L}{\partial a_i^{(k+1)}} \sigma' w_{ij}^{(k)} \\ &= \sum_i \delta_i^{k+1} w_{ji}^{(k)}\end{aligned} \tag{3.32}$$

where $\delta_i^{(k+1)} = \frac{\partial L}{\partial a_i^{(k+1)}} \sigma'$ which can be computed iteratively from the end of the network as $\delta_i^{(k)} = \sum_j \delta_j^{(k+1)} w_{ij}^{(k)} \sigma'$. The cascading computation of the $\delta_i^{(k)}$ from the end of the network is called the *backpropagation*. The derivatives with respect to any weight can finally be written directly with the δ as :

$$\frac{\partial L}{\partial w_{ij}^{(k-1)}} = \delta_j^{(k)} a_i^{(k-1)} \tag{3.33}$$

This procedure allows to perform gradient descent on a neural network efficiently. For more complex architecture, the computations are summarized in a *computational graph* as in Fig3.27. It represents the architecture as a directed graph (without cycles) where the nodes are the different function applied (such as the σ, the $(b + \sum_i w_i x_i)$ or the loss) and the arc the different variables (inputs, intermediate variables, weights) in the network. The chain rule can then easily applied to this graph to compute the derivative of the loss with respect to any of the variables by performing only local derivation in the functions multiplied by the derivatives computed further in the graph/network. This procedure is called the *automatic differentiation*.

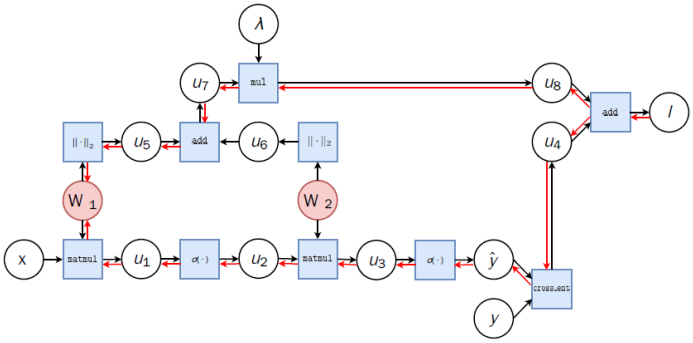


Figure 3.27 – Example of a computational graph. The black arrows show the forward pass while the red ones show the backward pass. The circle represent variables (intermediate or not) and the square the operations that should be differentiable. [34]

A last thing to notice is about the activation function $\sigma(x)$, and more specifically its derivative σ' that appears in the backpropagation as a multiplicator in the chain rule. The first version presented in (3.25) is unadapted as its derivative is 0 everywhere (except in 0 where it is undetermined). A few other activation functions have been proposed since then. The first was the *sigmoid* which is equal to $\frac{1}{1+e^{-x}}$. It is a good one but it suffers from a problem when dealing with deep neural network. The derivative of the sigmoid has a maximal value of $\frac{1}{4}$, but this value acts as a multiplicator at each layer. The propagated gradients thus inevitably tend to 0 as they move backward in the network. This phenomenon is called the *vanishing gradient*. The most common used activation function nowadays is the *Rectified Linear Unit* (ReLU) which solves this problem. It is defined as $\sigma(x) = max(0, x)$. This function has a derivative of 0 in the negative and a derivative of 1 in the positive (the $x = 0$ is undefined but often set to 0).

3.3.5 Optimizers & schedulers

While every deep learning optimization is based on the gradient descent algorithm, there exist some improvements on the initial algorithm. The standard gradient descent with a given training set is to compute the gradient for each sample, then to average them before taking a step. However, this operation depends linearly on the size of the training set which is not a good behavior. Thus the *stochastic gradient descent* (SGD), where an update is taken at each sample, is rather used[5]. All samples are still often process several times, the processing of each sample once is called an *epoch*.

The first parameter to play on is the learning rate λ. As stated in the previous section, it roughly dictates how far the step can be taken. A common practice in artificial intelligence is to favor exploration in a first time and then to be more restrictive. In the same idea, schedulers can be used to control the value of λ. Common schedulings are the step decay (divided after a constant number of epochs) or the exponential decay (modified at each epoch using a negative exponential decrease).

[5]To make use of the parallelization, mini-batch are also often used where the update appears after a fixed number of samples that can be process in parallel.

Other techniques have been applied to improve the step considered at each update. One is the first order *momentum*. The idea comes from the physic of a ball rolling on a surface and consists in keeping part of the previous step in the current one as a ball taking speed. It formulates formally as :

$$\begin{aligned} \mathbf{u}_t &= \alpha \mathbf{u}_{t-1} - \lambda \mathbf{g}_t \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \mathbf{u}_t \end{aligned} \tag{3.34}$$

where α is a constant governing how much impact previous steps have on the current one and $\mathbf{g}_t$ are the gradients computed at this step. This technique helps escaping local minima and allows to speed up in straight descent direction. The momentum can also be improved by computing the gradient $\mathbf{g}_t$ directly in the position where the momentum will bring the parameters (e.g. in $\boldsymbol{\theta}_t + \alpha \mathbf{u}_{t-1}$) in which case it is called *Nesterov momentum*. Similarly, adaptative learning rates such as Adam [35] make used of the second order momentum (in addition to the first[6]). This second order momentum represents the magnitude of the previous steps and divides the learning rate as the friction would do on a ball to keep the same analogy [36].

3.3.6 Regularization

As stated in Section 3.3.2, complex models are prone to over-fitting. Deep neural networks are particularly complex models. A way to limit the complexity of the model is to use *regularization*. The regularization aims at restraining the values that can be taken by the parameters to reduce its potential complexity. In neural networks it can be performed by adding a new term in the loss function (3.35).

$$L_{\text{tot}} = L + \gamma \sum_i w_i^2 \tag{3.35}$$

This new term adds a direct gradient $2\gamma w_i$ to each weight w_i in the backpropagation process.

3.3.7 Data augmentation

Another way to reduce the over-fitting (and under-fitting as well) is to add training samples. While collecting true new samples may be costly, it is possible to generate artificial samples from old ones instead. This is called the *data augmentation*. In the context of image classification for examples, one can generate random transformations (rotation, translation, cropping/brightness changes, ...) to the images in the actual dataset as new samples. Those artificial samples are not as good as true new samples, as they are just altered version of other images already in the dataset, but they still may help the network to generalize the problem.

Even though data augmentation can be performed statically before training a network, the best practice is to used a new more or less altered version of each sample generated dynamically at each epoch.

[6]The momentums used are however implemented as an exponential decaying average instead of a pure accumulation. Bias corrections are also used to stabilize the first steps.

3.3.8 Convolutional neural networks' layers

Fully connected layers as in MLP use many weights and are not appropriate for all tasks. This section will review a few interesting layers when dealing with images as input.

The first type of layer is the *convolutional* layer. It is defined by a kernel that is represented as a 3D tensor sliding across the width and the height of the image. It then produces a 2-dimensional feature map whose elements are the piecewise multiplication of the kernel with the different locations of the image (Fig 3.28). The kernel aggregates neighboring values weighted with weights that have to be learned by the network. Several kernels can be stacked at the same layer to produce different feature maps which will be stacked on the third dimension (as the RGB channels) for the following layers. A standard application of the kernel will reduce the width and the height of the image as the first valid location for a 3x3 kernel will be centered on pixel (2,2)[7], thus reducing the outputted feature map by one pixel at each border. This can be avoided if necessary by adding padding 0's at the border beforehand such that the first valid center is the first pixel of the image. One can also reduce willingly the size of the outputted feature map by moving the kernel of more than one pixel at each application (called the *stride*). A stride of 2, with the image padded, will divide by 2 the width and height of the output compared to the input size.

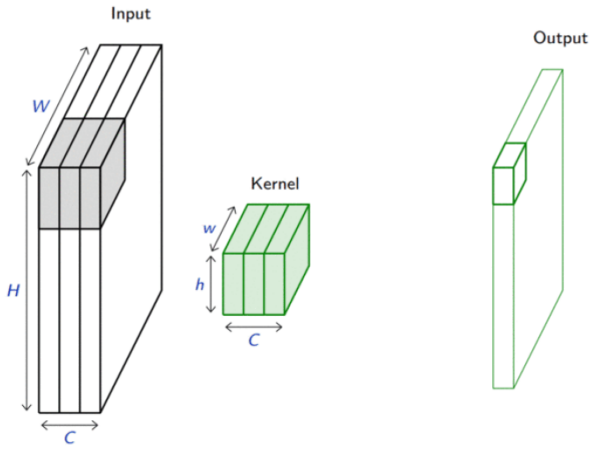


Figure 3.28 – Example of convolutional kernel. Each part of the input (with overlap or not) is multiplied point-wise with the kernel block to produce a single value. Those values are then aggregated in a new channel. [37]

The next type of layer is the *pooling* layer. It performs a sort of downscaling by also aggregating groups of neighboring values into a single value. However, the pooling layers have some particularities compared to the convolution. It does not aggregate values across channels, the aggregated groups do not overlap (as a convolutional layer with a stride equal to its kernel size) and they do not have any parameters to learn as the type of aggregation is predetermined. Two common types of pooling layer are the max-pooling,

[7]Considering an indexation starting at (1,1)

taking the maximum value of the group, or the average-pooling, taking the average as its name suggests.

Then there is the upsampling often achieved with *transposed convolution*. The approach is similar to the convolution layer in the sense that a kernel is also learned by the network. But instead of using the kernel as an aggregator, it will spread each value from the input in the outputs. If the goal is to double the width and height, the kernel will spread the values in the output with a stride of 2. Note that upsampling can also be performed with more classical interpolation methods like the bilinear upsampling in Section 3.1.6 which does not have to learn parameters. Less parameters means less complexity which may be a good or bad thing depending on the dataset used and the objective of the neural network.

The final type of layer, used nowadays in many other types of network, is the *batch normalization* [38]. The goal of this layer is to normalize the data at a given layer to stabilize the behavior of the model[8]. It is performed by computing a channel-wise computation of the mean and the variance (across the width, height and various samples in the mini-batch). The values are then re-distributed on a normal distribution of mean 0 and variance 1. This distribution can then be rescaled and shifted to a given mean β and a variance γ^2 which are both parameters to be learned by the network. Concerning the location of such layers, it was initially recommended to use them before the non-linearity (e.g. the ReLU activation function), but it is now yet another part of the debates around the batch normalization.

[8]There are actually motivation from the weights' initialization to perform batch normalization linked to the internal covariate shift. The true impact of that layer is however still not completely understood even though its effectiveness seems to not let any doubt [39].

3.4 State of the art

The last section of this chapter is where the state of the art techniques used later are described, using notions presented in the first three sections. Those techniques regroup specific feature-based registration techniques, a pixel-based registration metric and the neural network used to perform segmentation.

3.4.1 Differentiable mutual information

The mutual information (Section 3.2.2) for image registration is defined over the joint histogram (Section 3.1.4). Whereas intensity values between stains can be quite different, it is expected that similar tissues/cells always take the same range of intensities within a stain. Instead of trying to align pixel with similar values, the idea here is to look for a registration where pixels in a given range for the first image are mostly paired with value within another single range in the second image. This would means that, knowing the pixels from the first image, one would be able to predict what would be the value on the second one i.e. the mutual information is high.

In this context, the joint probability $P(\mathcal{M}, \mathcal{N})$ is defined as the number of samples in the bin b_{mn} over the total number of samples, where m is the id of the range of values for the pixel in the first image and n the id of the range for the second one. The marginal probability of the variable $\mathcal{M}$ is directly derived by summation of the joint probabilities over the variable $\mathcal{N}$ (and vice versa).

While this would be enough to apply 0^{th} order search algorithm, the discrete nature of this joint probability makes it impossible to differentiate, and thus to apply to higher order method.

Many papers have been published on the subject [40][41][42]. The common idea in those papers is to compute an estimate of the continuous joint probability through Parzen windows. This method allows to estimate a distribution from a set of N samples x_i using a positive kernel $w(t) \geq 0$ with unit integral ($\int_{-\infty}^{\infty} w(t)dt = 1$) with the following formula :

$$p_{es}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{w\left(\frac{x - x_i}{\epsilon(N)}\right)}{\epsilon(N)} \tag{3.36}$$

where $\epsilon(N)$ is a scaling factor for the width of the window.

With the same methodology, the continuous joint histogram can be estimated :

$$h(m, n; \boldsymbol{\mu}) = \frac{1}{\epsilon_M \epsilon_F} \sum_{i=1}^{N} w\left(\frac{m - f_M\left(g(\mathbf{x_i}; \boldsymbol{\mu})\right)}{\epsilon_M}\right) \cdot w\left(\frac{n - f_F\left(\mathbf{x_i}\right)}{\epsilon_F}\right) \tag{3.37}$$

where M and F represent the moving and the fixed image, ϵ_M and ϵ_F their respective scaling factor, $f_M()$ and $f_F()$ the intensity of a pixel at a given location $\mathbf{x}$ for the moving/fixed image, $\boldsymbol{\mu}$ are the transformation parameters and $g(\mathbf{x}; \boldsymbol{\mu})$ is the transformation function.

The continuous joint distribution and the marginal distribution are directly derived from it :

$$p(m,n;\boldsymbol{\mu}) = \frac{h(m,n;\boldsymbol{\mu})}{\sum_m \sum_n h(m,n;\boldsymbol{\mu})} \tag{3.38}$$

$$p_M(m;\boldsymbol{\mu}) = \sum_n p(m,n;\boldsymbol{\mu}) \tag{3.39}$$

$$p_F(n;\boldsymbol{\mu}) = \sum_m p(m,n;\boldsymbol{\mu}) \tag{3.40}$$

Then the mutual information can be formulated as in (3.24), or rather the opposite of it as a gradient descent will be applied, thus solving a minimization problem (whereas the initial mutual information should be maximized). The detail of the differentiation can be found in the paper linked above. However, a point to notice in the development is the fact that the marginal distribution of the fixed image $p_F(n;\boldsymbol{\mu})$ depends on $\boldsymbol{\mu}$ from its construction (3.40) although it should not be impacted by it. But fortunately, an additional constraint on the Parzen window[9] allows to remove that effect of coupling. It is called the *partition of unity* and states that :

$$\sum_{x\in\mathbb{Z}} w(x+a) = 1, \qquad \forall a \in \mathbb{R} \tag{3.41}$$

Knowing these, the last part is to choose a Parzen window $w()$. A popular choice satisfying all the conditions mentioned above is to take one of the normalized B-spline functions $\beta^n(x)$[43]. The B-spline function of degree 0 $\beta^0(x)$ is defined as the unit square impulse (3.42) and the higher orders $\beta^n(x)$ are defined as a convolution with the previous order (3.43). The four first orders can be seen in Fig 3.29.

$$\beta^0(x) = \begin{cases} 1, & \text{if } -\frac{1}{2} < x < \frac{1}{2} \\ \frac{1}{2}, & \text{if } x = \pm\frac{1}{2} \\ 0, & \text{otherwise} \end{cases} \tag{3.42}$$

$$\beta^n(x) = \int_{-\infty}^{\infty} \beta^{n-1}(x)\beta^0(x-t)dt \tag{3.43}$$

[9]It actually also impose a condition on the ϵ_M and ϵ_F stating that $\frac{m}{\epsilon_M}$ and $\frac{n}{\epsilon_F}$ have to be in $\mathbb{Z}$ for all value of m and n. See the papers for more details.

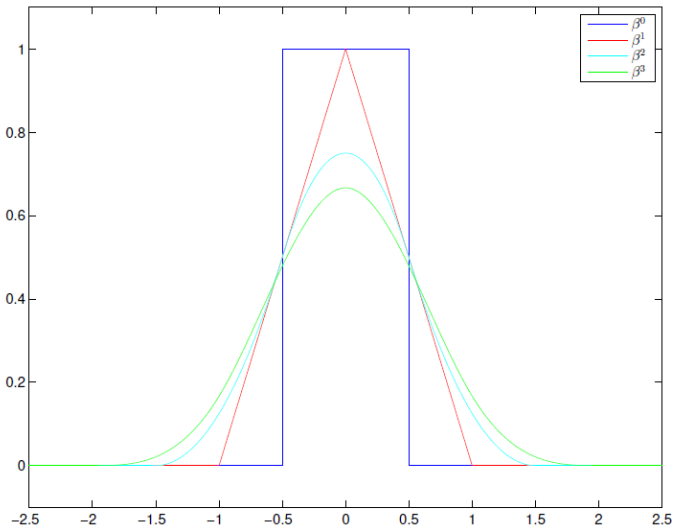
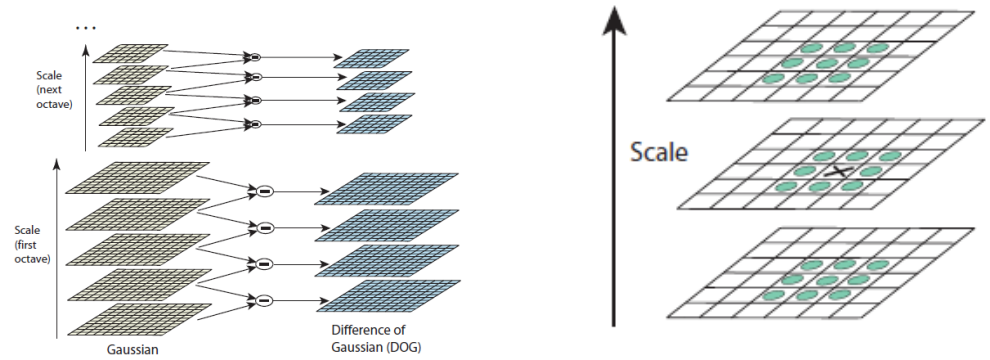


Figure 3.29 – Normalized B-spline function up to order 3
Credits [43]

3.4.2 SIFT feature detector

SIFT (Scale Invariant Feature Transform) [44] is a state of the art feature detector used to perform the two first tasks mentioned in Section 3.1.8, namely the location (finding keypoints) and the description (finding descriptors).

The location of keypoints is performed through three main steps. First Gaussian blurs are applied with increasing "magnitude" (more blurred) five times. Then the difference of successive blurred images is computed (Difference of Gaussians) as in Fig 3.30a. This DoG is actually an approximation of the Laplacian of Gaussian. The idea here is to detect sharp changes in intensity that will describe interesting points such as corners. The DoG has moreover several advantages over the LoG. It is much faster to compute, it is less sensitive to noise and it is already scale invariant. For the second step, local extrema are found by taking points where the 8 neighboring pixels in the blurred image and the 9 neighboring pixels at adjacent blur magnitude are all greater or smaller than the point considered as in Fig 3.30b. The exact location of the point is refined by using a second order Taylor expansion.
This whole process is repeated for several resolutions (called *octave* in the paper) to sample more accurately the scale-space.



(a) Difference of gausians across scales

(b) Finding local extrema

Figure 3.30 – Sift feature detection. The difference of Gaussians is performed between the different successive blurring magnitudes to find sharp changes in intensities (on the right). The local maximums of those changes are then found across the DoG.[44]

However, this detection mechanism may also detect uninteresting points. The first category of removed points is those with low contrast, which is easy to filter as they are too sensitive to noise. The seconds are the edges as those are often not so interesting for feature matching because there may have several successive points on a long edge that would look similar. The idea to remove those ones is to analyze the curvature in two perpendicular directions and see if the value is high for both (which is expected for a corner, whereas an edge will only have one high value). The exact procedure followed by SIFT is to estimate the Hessian matrix (for second derivatives) with neighboring pixels and assess that the ratio between the eigenvalues is small enough (as it is possible to do it faster than computing the eigenvalues themselves).

The last step is to compute the feature descriptor. As stated above, we want it to be scale and rotation invariant. The first thing to do is actually to assign a scale and an orientation to each keypoint, such that in the end we can "normalize" our descriptor knowing these special properties of the actual keypoint. For the scale factor, it is assigned at the detection step and is proportional to the level of blurring and the resolution (which octave). Now an orientation have to be assigned. To do so, a window around the keypoint is taken (sized and blurred according to the keypoint scale factor) and the gradient magnitude and orientation is computed for all points in the windows (estimated from the 4 neighbor points). The angular space (360°) is then decomposed in 36 bins covering 10 degrees each. Then, according to the orientation computed, each gradient point magnitude is weighted by a Gaussian circular window around the keypoint (with σ equals to 1.5 times the scale factor), and summed in the appropriate bin. Once the histogram representing degree range is computed, the maximum bin is taken and the orientation factor of the keypoint is estimated by a parabola fitted by this maximum and its 2 neighbors. If it appears that there is another local maximum (peak) in the histogram that is not the global maximum but of at least 80% of the maximum found, a new keypoint is created at the same location and with the same scale factor but another orientation corresponding

to this other peak. This means that several keypoints can share the same location but different orientation factors, and this is also the case for the same location and different scale factors as maximum can be found at the same location but at another DoG[10] or at another octave in the first step.

Finally, the feature vector is computed with a procedure quite similar to the orientation assignment mentioned above. A 16x16 sample array around the keypoint is taken and divided in 16 subsquares of 4x4 samples (from the image blurred according to the scale factor). This sample array is taken of size according to the scale factor as above and oriented according to the orientation factor (then resized to fit the 16x16 size). Afterwards, the angular space is divided in 8 bins and a histogram is built for each of the 16 subsquares by summing the magnitude of the samples[11] within them (weighted by a Gaussian circular window centered on the keypoint, and with σ equals half of the window size used above to determine the orientation factor). An example with a smaller sample array is shown in Fig 3.31.

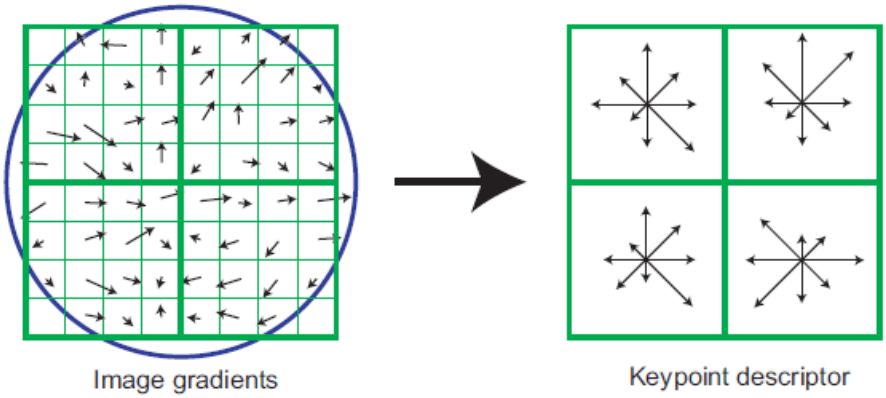


Figure 3.31 – Example of SIFT feature descriptor with a 8x8 sample array divided in 4 subsquares. The gradients are then combined in each subsquare to describe the magnitude in the eight main directions.[44]

In the end, the 8 values of the 16 subsquares are aggregated in a feature descriptor vector of 128 values. This vector is also slightly post-processed by normalization to unit length (to reduce the effect of constant/affine illumination changes), then the values above 0.2 are reduced to 0.2 (to reduce the effect of non-linear illumination changes) and normalized again to unit vector.

[10] If 5 scales are used per octave, then two maximum at the same location and from the same resolution should be at scale 2/5 and 4/5 as they cannot be neighbors in scale according to the way maximum are found in DoG (and knowing that no maximum are taken from the greatest and lowest scale, as they only have a single neighbor in scale while the procedure requires a greater and a lower one).

[11] The magnitude of a sample is actually also weighted by its difference in orientation with the central value of the bin it is putted in.

3.4.3 ORB feature detector

ORB (Oriented FAST and Rotated BRIEF)[45] is another feature detector developed by the OpenCV team [46] which first goal was to provide a feature detector that was not under patent in opposite to SIFT[12][44] and SURF[47]. The ORB detector is actually a mixing between the FAST keypoints detector [48] and the BRIEF descriptor [49] with some additional features.

About the FAST detector, the principle is to take a circle of radius 3 around a potential keypoint and check the 16 pixels on this circle (Fig 3.32), if among them there are at least 12 contiguous points with intensities all greater (resp. smaller) than the keypoint value plus a treshold value t (resp. minus that threshold value t), the central point is considered as a corner. To speed up the process, it starts by looking at 4 extreme points on the circle and check if 3 of them can be grouped as greater or smaller.

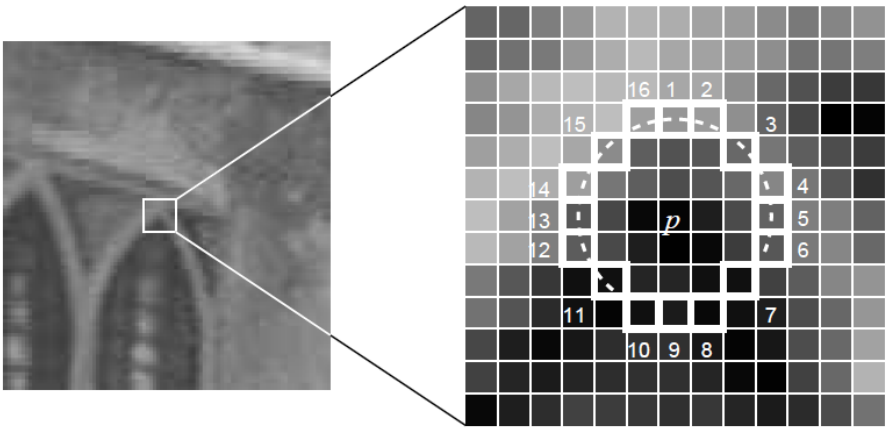


Figure 3.32 – Example of FAST circle used to evaluate a point p. p will be considered as a keypoint if 12 consecutive pixels among the enumerated ones are all greater or smaller in intensity than p. [48]

In the case of ORB, FAST-9 is used (looking for 9 contiguous value greater/smaller) and is applied at multiple scale (to detect features in a scale invariant manner). Then the features are ranked using the Harris score [50] where the idea is very roughly to check if a small translation in every 8 directions induces a high change in intensity value (which is expected from a corner). Only the top N features are kept afterwards. The first main addition from ORB is the implementation of a rotation compensation. To do so, they compute the *moments* of the patch (3.44) and with them they compute the centroid C (3.45) and the orientation θ (3.46) of the patch. To improve the rotation-invariant property, they only consider the points within a circle of radius r equal to the patch size

[12]SIFT patent has expired a bit more than a year ago (March 2020).

for the computation.

$$m_{pq} = \sum_{(x,y)\in \text{patch}} x^p y^q I(x,y) \tag{3.44}$$

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{3.45}$$

$$\theta = \text{atan2}(m_{01}, m_{10}) \tag{3.46}$$

The final step is the descriptor and for ORB, it is a binary vector. To determine one of the value of the descriptor, a simple binary test τ is performed. A pair of points is taken within the patch (at fixed locations for a given element of the descriptor to be consistent) and if the first point of the pair as a higher (or equal) value the result is a 0, while a first point with a lower value will give a 1 (3.47).

$$\tau(\mathbf{x_1}, \mathbf{x_2}) = \begin{cases} 0, & \text{if } f(\mathbf{x_1}) \geq f(\mathbf{x_2}) \\ 1, & \text{if } f(\mathbf{x_1}) < f(\mathbf{x_2}) \end{cases} \tag{3.47}$$

Such choice of pair has been precomputed by randomly taking a first pixel in a Gaussian distribution around the keypoint, then a second random point in a Gaussian distribution around the first point with a σ twice the first one. In the case of ORB, 256 features are computed so this procedure has been done 256 times. Moreover, ORB computes an orientation for the keypoints, thus the coordinates of the pixel to consider for a test are rotated to match the orientation of the keypoint (*steered BRIEF*). To be more precise, the rotations of the binary test are also precomputed, to make it faster, and are stored in a lookup table aggregated by range of 12°.

However, while the BRIEF original binary tests had a nice property that the tests were not correlated (i.e. with a mean of 0.5 and a large variance), it was not really the case anymore for the steered BRIEF. They design an algorithm of greedy search to find an uncorrelated binary tests set, for all orientations, to recover that property. This final version is called rBRIEF.

3.4.4 U-net & losses

U-net [51] is a state of the art deep neural network architecture used for segmentation (e.g. producing binary mask). Its name comes from its U-shape (Fig 3.33).

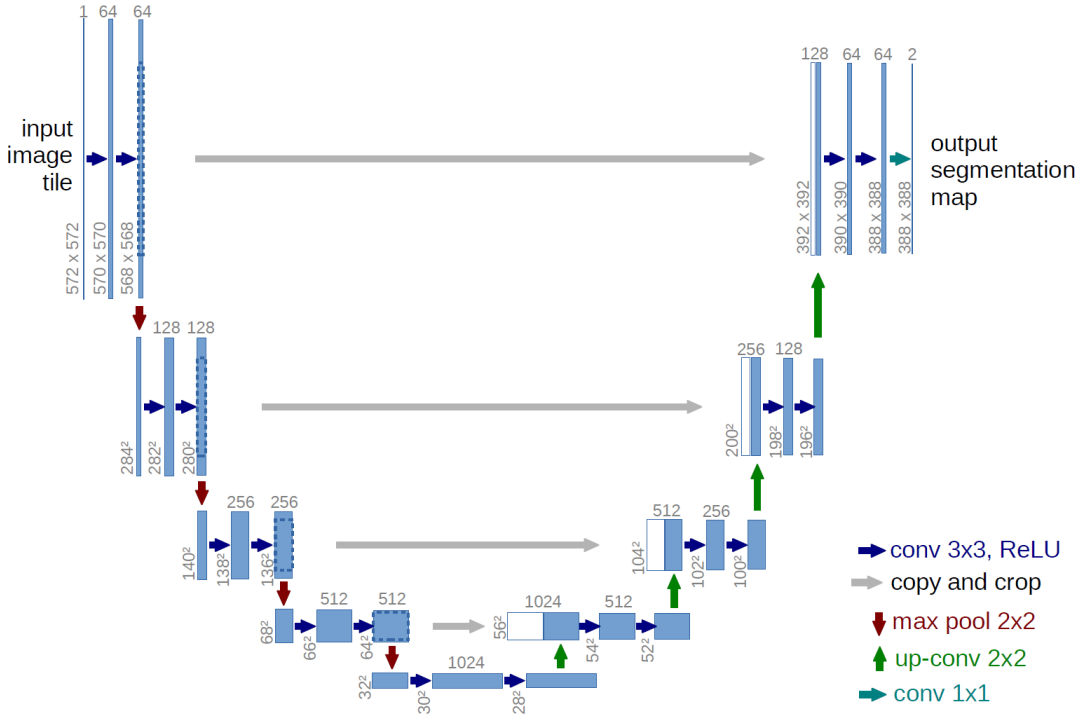


Figure 3.33 – U-net architecture with the input image on the left and the outputted segmentation mask on the right. [51]

The left part is the *contracting path* reducing the width and height into a feature map of $\frac{1}{16}$ of the original size with 1024 channels, and the right part is the *expansive path* going back to the initial width and height and number of channels. Each stage of the contracting path starts by performing two successive convolutions with a kernel of size 3x3 without padding that double the number of channels (except the first one where the number of channels is set to 64). Each of the 2 convolutions is followed by a ReLU activation function. Afterwards, a max-pooling with a 2x2 kernel is used to downsample the image by a factor of 2 to the lower stage. For the expansive path, it starts with the same double convolution but the number of channels is divided by 2 instead of being multiplied. Then a upsampling convolution with a kernel of size two is performed to double the size and the number of channel is again divided by 2. The output is however stacked with the output from the end of the "symmetric" contracting path such that the number of channels is doubled (thus only divided once in the double convolution). The part of the contracting path concatenated also has to be cropped due to the border pixels lost within unpadded convolutions. The very last step is a 1x1 convolution used to readjust the number of channels to 1.

In addition to this architecture, two loss functions will be used. The first one is the *binary cross entropy* (BCE). This loss function considers labels y (equal to 0 or 1) and compares them with a predicted probability p_1 to be from the class labelled as 1. The BCE loss is written as follow :

$$L_{BCE}(p_1(y), y) = -\left[y \cdot \log\left(p_1(y)\right) + (1-y) \cdot \log\left(1 - p_1(y)\right)\right] \tag{3.48}$$

In this loss, only one of the two term is effective at a time. If the true class is 0, the first term is always equal to 0 and the second term becomes $-\log(1-p_1(y))$. But as there are only two classes ; $p_0 = 1 - p_1$ and the loss is $-\log p_0(y)$. Similarly, the loss for a sample with true label of 1 is $-\log p_1(y)$. In both cases, the loss is equal to 0 when the probability associated to the true label of the sample is maximal and it increases exponentially as it goes down. In the case of a binary mask, this is performed on every pixel prediction.

The second loss is the *intersection over union* (IoU), which will be also used as a metric to assess performance of the different methods. Here the label 0 is associated to the background and is not considered as interesting. The important parts are the pixels with label 1 in both mask and predicted mask. First, the number of pixels labelled as 1 in both masks are computed as the *intersection* of the masks. This can be performed by a pixel-wise multiplication of the labels then summing all values obtained. Second, the number of pixels labelled as 1 in at least one of the masks is computed as the *union* of the masks. This step can be done by summing all the label values in both images and retrieving the union computed at the step before (as those pixels as been counted twice). The IoU can finally be computed as $\frac{\text{intersection}}{\text{union}}$. As this metric should be maximized, $1 - \frac{\text{intersection}}{\text{union}}$ is taken for the loss.

Using $c(x, y)$ as the true class label and $\hat{c}(x, y)$ as the predicted label, the loss is written :

$$L_{IoU}\left(c(x,y), \hat{c}(x,y)\right) = 1 - \frac{\sum_{(x_i,y_i)\in \text{mask}} c(x_i, y_i) \cdot \hat{c}(x_i, y_i)}{\left(\sum_{(x_i,y_i)\in \text{mask}} c(x_i, y_i) + \hat{c}(x_i, y_i)\right) - \left(\sum_{(x_i,y_i)\in \text{mask}} c(x_i, y_i) \cdot \hat{c}(x_i, y_i)\right)} \tag{3.49}$$

However, the network will output prediction as a probability to be labelled as 1 (as in the previous loss). The final prediction is done by thresholding the probabilities that are above or below $\frac{1}{2}$. This thresholding is nevertheless not a differentiable operation. The loss will thus use directly the probability in $\hat{c}(x, y)$ instead of exact label predictions.

Chapter 4

Protocols & metrics

The *Protocols and metrics* chapter will present the exact protocols that will be deployed in order to solve the problem described in Chapter 2, i.e. the registration of two linked annotations within a same image group. This will be done using either only global registration, with feature-based and pixel-based methods (Section 4.1), or a global registration followed by a local one, with again feature-based registrations (Section 4.2) or deep neural network segmentation (Section 4.3). The metrics that will be used to assess the performances are also described in Section 4.4.

4.1 Single step registration

The first protocols consist in a single registration to align both images. Those registrations are performed on the images fetched with a maximal size of 512 pixels[1]. Working on the original images would take too much time and, additionally, the very local feature may more disturb the registration due to the highly non-rigid aspect of the slicing. To reduce the impact of the various stains and speed up the processes, the images are registered in grayscale. For the degree of freedom of the registration, two cases are tested. For the different techniques, *full* affine transform (translation + rotation + scale + shear) and *partial* affine transform (withouth shear) are used. As registration procedures, three techniques are considered.

The first two are feature-based registrations, namely the SIFT detector (Section 3.4.2) and the ORB detector (Section 3.4.3), with a maximum of 5k features for the latter. The feature descriptors are then matched using respectively the L2 norm and the Hamming distance (as the SIFT detector produces continuous descriptors while the ORB detector produces binary ones) using cross-checking. The 10% worse matching are then removed to exclude some outliers[2]. The remaining points are finally used to find a transformation (with the chosen degree of freedom) with RANSAC. Those operations are performed

[1]Higher or smaller resolution may be tried for future works, the size of 512 pixels was a good tradeoff to have a sufficient resolution while removing too local features. The resolution used also often affects the speed of the method (finding features, computing mutual information) such that a too high resolution would have made the assessment of the performance much longer.

[2]The value of 10% should encompass the worse outliers while not removing too many good matches.

thanks to the OpenCV library [46].

The last technique is pixel-based. As the starting point of the process is important, the centers of both images are aligned if their resolutions are different (and the center of rotation of the transform is set to this transform). The images are then registered on their mutual information (Section 3.4.1) with gradient descent using a learning rate of $\frac{1}{2}$. This registration is also done in a multi-resolution fashion with 3 scales (from coarse to fine) and is performed with the SimpleITK library [52].

Once the registration in the downscaled fetched image is found, the original annotation in the moving image is transformed with the following successive transformations : rescaled from the moving image original size to its fetched size, reversed on the y axis with the fetched height (as the annotations are in cartesian system while the transformation are computed in the computer referential), registered according to the registration found, turned back to the cartesian system (from the fixed image referential this time) and finally resized to the fixed image original size.

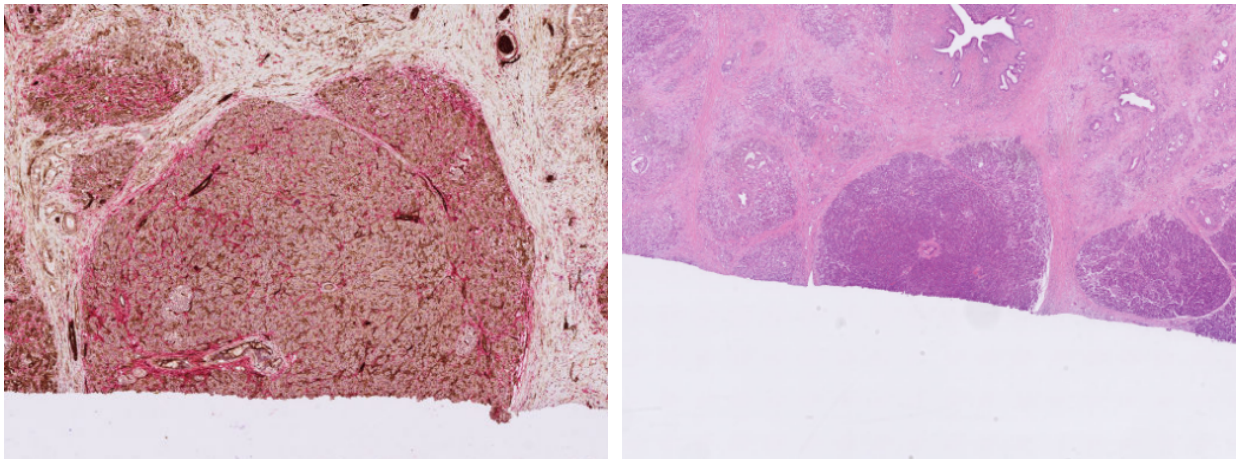
4.2 Two steps registration

To enhance the registration, a second step is added. The first registration is still performed as above but only the centroid of the annotation (in the moving image) is registered with the procedure explained at the end of the previous section. Then two cases are considered : either a pattern-matching is performed or a "large window" registration.

In the first case, the annotation in the moving image is fetched from the Cytomine API with a maximum resolution of 512 pixels[3]. The window fetched in the image is based on the minimum and maximum values in both axes of the polygon of the annotation (with a small increment of 30% in both dimensions to have a slight margin from the border of the annotation). A window is then fetched in the fixed images around the centroid registered. This time, the window is taken as twice the dimension used for the first window. Additionally, if this second window is too small, i.e. smaller than 10% of the original image's size[4], 10% of the original size is chosen (for the dimension that is too small, both if both are too small). An example of such windows in shown is Fig 4.1.

[3]The whole images have been fetched in 512 pixels, extracting the annotation directly from this image would give an image with only a few decades of pixels as width and height which is too small.

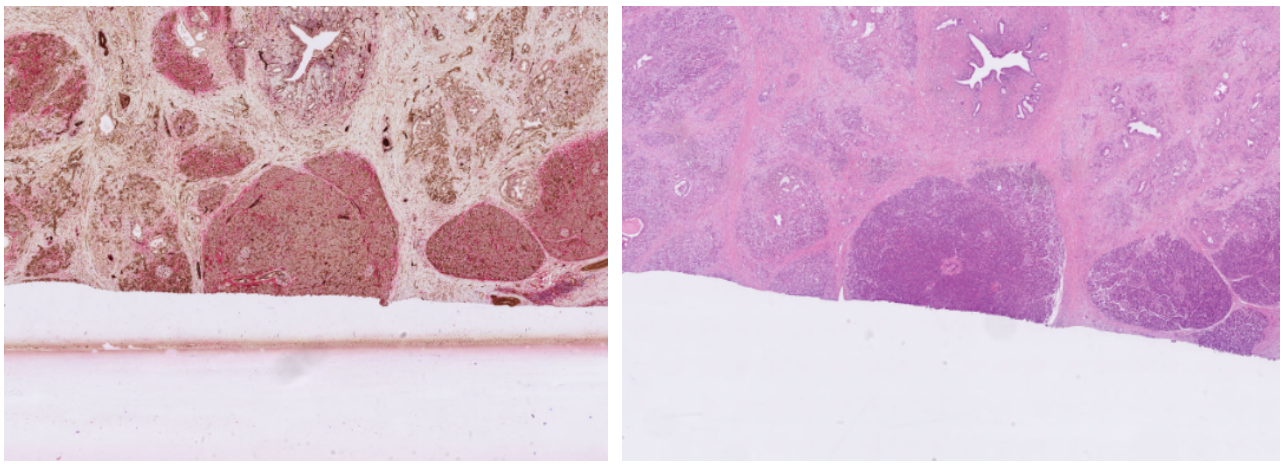
[4]The value of 10% has been empirically found such that most of the annotation whose average size is smaller than 0.1% of the image size are still visible in the window, while remaining the more local possible.



(a) Window moving image (b) Window fixed image

Figure 4.1 – Example of the two windows extracted after the first registration in the pattern-matching case. The first is restricted to the known annotation in the moving image while the second is a more global glimpse around the expected location of the annotation in the fixed image.

In the second case, the same width and height is taken for both fetched windows as 3 times the annotation size (with the same minimum size of 10% of the image size). The windows are respectively centered on the annotation for the moving image and on the registered centroid for the second. An example of those windows is presented in Fig 4.2. The size has been chosen slightly bigger, 3 times instead of 2.6 times[5], than in the previous case. While only the annotation needs to be strictly present in the fixed window for the pattern-matching case, this method rather expects the greatest overlap between the two windows in order to obtain good results, a slightly greatest window is thus taken.



(a) Window moving image (b) Window fixed image

Figure 4.2 – Example of the two windows extracted after the first registration in the "large window" registration case. Both contain the surroundings (a) of the known annotation in the moving image (b) of the expected location of the annotation in the fixed image.

[5]In the pattern-matching case, it is the annotation size after being increased by 30% which is doubled.

The initial annotation is then translated to the referential of the window in the moving image (i.e. with the moving window bottom left corner), then registered with the same procedure as in the previous section (but using the fetched and original sizes of the windows fetched instead of the whole images) then translated again from the referential relative to the window in the fixed image to the fixed image referential (i.e. with the opposite of the fixed window bottom left corner).

Those registration can fail within the process. For example, the first registration can give a transform such that the annotation center in the moving image is registered out of the fixed image. When this happens, a window can not be downloaded to perform the second registration and the process must abort. In such cases where the registration can not be performed entirely, a failure is counted and the worse metric values are taken (i.e. 0 for the IoU and 1 for the RCM).

4.3 Deep learning approach

Lastly, a deep learning approach is applied for the second local registration instead of using a second time the same method as it has been done in Section 4.2. The idea is to used the grayscale square windows in both images (Fig 4.3a and 4.3c) and a mask of the known annotation in the moving image (Fig 4.3b) to predict a mask of the annotation in the fixed image (Fig 4.3d).

The dataset is fetched the following way. First the original size of the windows is chosen as twice the maximum dimension of the annotation size (or 10% of the maximum dimension of the image if it is smaller). A transformation between both images is first computed with the single step ORB detector. Then, the square window in the moving image is fetched with the size discussed above multiplied by $\frac{3}{2}$ with a max effective size of 1024 pixels[6]. This value of $\frac{3}{2}$ comes from the worse case rotation where the angle is $45° + k \cdot 90°$. In this case, the rotated square should be $\sqrt{2} \approx 1.41$ larger to be able to crop a square (whose borders are parallel to the image's borders) that is inscribed in the rotated one. The value of $\frac{3}{2}$ should thus gives windows without black padding in cases where the global scaling is not too extreme.

This moving window (and the moving annotation) finally undergoes the following transformations :

1. The window is resized to its original size and translated to its location in the moving image, such that the window and the annotation in the moving image are aligned.
2. They are both scaled down according to the scale in which the global registration has been performed. In other words, if the whole moving image was fetched and registered with a resolution of 512 pixels while its original size was 2048, the image was scaled down by a factor of 4 and the same shrink have to be applied to the window and the annotation before using the global registration found previously.
3. They can finally be registered using the global registration found previously. Note

[6] A greater resolution is used to avoid the upscaling when the final window will be cropped back a smaller version and rescaled to 512 pixels.

that the registration preformed by OpenCV considers the computer representation (with the (0,0) coordinate at the top left). The annotation, which is in cartesian coordinates, have to undergo a change of coordinate (relative to the height of the fetched moving image) beforehand. It must also be turned back to cartesian coordinates as well afterwards. However, as the moving annotation is expected to be aligned with the fixed annotation in the fixed image coordinate system, the height of the fixed image have to be used for the second transformation.

4. Both are then rescaled to the fixed image original size. The center of the moving window is now aligned with the center of the fixed window that will be fetched (with the original size of the windows determined earlier). The moving window is also cropped[7] such that it recovers the original windows size with its borders parallel to the image borders. This way the four borders of the windows are aligned as well.

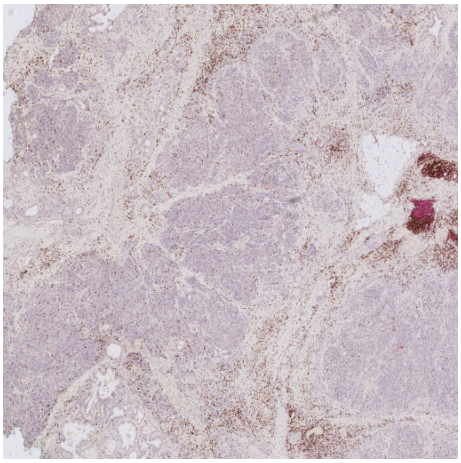
5. When the fixed window is fetched, the image obtained is initially already align with the (0,0)[8] at the corner and scaled down to the 512 pixels size. To simulate the same behavior for the moving window/annotation (and the fixed annotation used as ground truth as well), they are translated such that the corner is aligned with the (0,0) as well then they are also scaled down to the 512 pixels size.

If padding is required, in the case where the square window is fetched partially out of the image or if the registering transformation is such that the cropping is not fully covered by the window fetched, the image is padded with 0 (black pixels). The fixed and moving polygon annotations, already aligned in the window relative coordinate system, are turned to mask. To do so, all pixels whose centers are included in the polygon are set to 255 (white) while all the others are set to 0 (black).

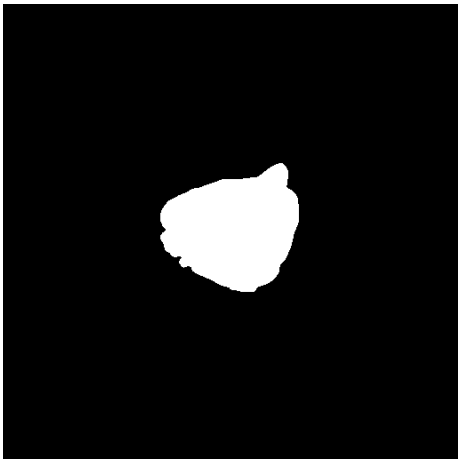
To speed up the process, the different transformation matrices are pre-multiplied together then applied to the image/annotation.

[7]The cropping actually appears at the end as it is not an affine transformation, but the result is the same as the cropping is done in the final scale of the original window size (i.e. 512 pixels) instead of the the original windows size as described here.

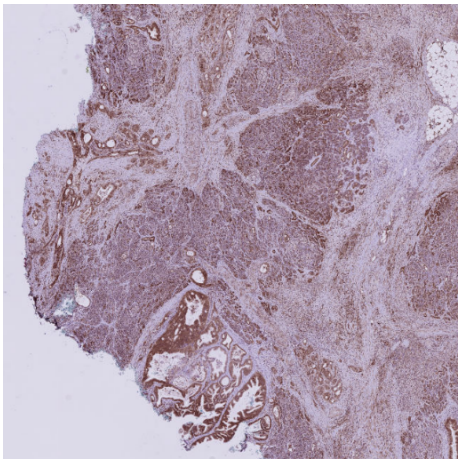
[8]The translation and the alignement with the left corner (either top or bottom) is performed in the coordinate system of the object considered, so the window is treated using the computer coordinate system while the annotations are treated using the cartesian coordinate system.



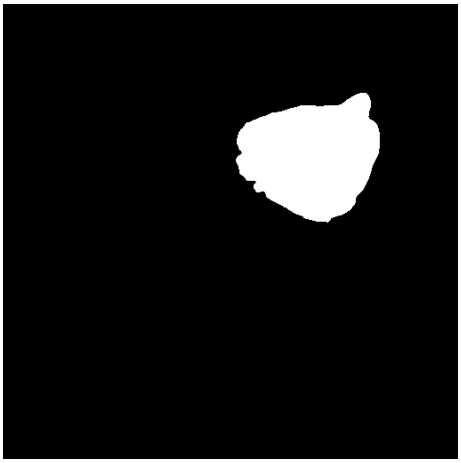
(a) Window moving image (input)



(b) Mask moving image (input)



(c) Window fixed image (input)



(d) Mask fixed image (ground truth)

Figure 4.3 – Example of the four components of a sample. The input is composed of (a) a window including the known annotation in the moving image and its surrounding, (b) the mask of this annotation, and (c) a window of the expected location of the annotation in the fixed image. The two windows are converted to grayscale such that the three inputs are stacked as a custom three channels image. The expected output is (d) the mask of the annotation in the second window i.e. in the fixed image.

Some pairs can not be recovered from this procedure as their registration failed so much that the window does not exist. In the end, 3948 pairs were fetched from the 4160 possible ones (considering that a pair of stains can be used twice by swapping the stain for the fixed and the moving image). Each pair will then be considered as a single sample for the deep learning algorithm. Note that among them, some still have an empty mask for the ground truth as the registration was not good enough. To split the dataset between the train, evaluation and test set, it was mandatory to make a split based on the image groups and not on the annotation groups[9]. It has been decided to split it as presented in

[9]A split for the staining as well would have been ideal but this could not be afforded. Many images would have been removed to satisfy this additional constraint and the dataset is already not very big.

Table 4.1 to balance as much as possible the number of annotations in the different image groups within the same set.

Set	Images	# of annotation groups	# of annotations	# of samples
Train	PKR-1 PKR-2 PKR-7 PKR-10	11	158	2506
Eval	PKR-3 PKR-4	3	41	671
Test	PKR-5 PKR-6 PKR-8 PKR-9	5	59	771

Table 4.1 – Set splitting between train set, evaluation set and testing set.

The network U-net (Section 3.4.4) is then used with the PyTorch library[10] [53]. The implementation differs slightly in some place from the original U-net. All the convolutions are padded to recover a mask of the same size at the end and batch normalization is used between the convolution and the ReLU layers. Predictions at the end of the network are turned to probabilities by applying a sigmoid function, then thresholding around the probability $\frac{1}{2}$ to determine if the class should be 0 or 1 (this last step is not applied in the training as the loss would not be differentiable). Two alternatives for the upsampling are explored, either the transposed convolution or the bilinear upsampling.

Two losses are also evaluated ; the BCE and the IoU losses (Section 3.4.4). The network is then trained with the Adam optimizer[11] using an initial learning rate of 0.001 (multiplied by 0.9 every 5 epoch) and a weight decay of 10^{-8} over 20 epochs with a batch size[12] of 1. Additionally, data augmentation is performed, dynamically at each epoch, with random affine transform (rotation up to 45°, translation up to 30% of the size, scaling from 0.8 to 1.2 and shear with factors between -10 and 10)[13]. The augmentation is performed separately on the image/mask from the moving image and from the fixed image (an image and its associated mask however undergo the same transformation).

Depending on the quality of the masks, closing and/or opening (see Section 3.1.3) could be applied to remove noise or to fill holes within the predicted mask.

[10]The implementation used is based on the Github Pytorch-UNet.

[11]As optimizer, some tests have also been done with SGD using different learning rates but the results were not significantly different. The same goes for the weight decay which has been tried at higher value up to 10^{-4} leading to results which were either similar or worse.

[12]The limitation of Cuda on the computer used did not allow a batch size greater than one.

[13]The value for the transformations have been empirically designed such that it should simulate several average registrations for the pairs where the initial registration is pretty good.

4.4 Metrics

Two metrics are used to assess the performances. The first is the *intersection over union* presented as a loss in (Section 3.4.4) where the ratio of the overlap of both polygons with respect to the total area covered by at least one of them is computed (Fig 4.4). This measure is commonly used in computer vision tasks such as segmentation.

$$\mathrm{IoU}(B, \hat{B}) = \frac{\mathrm{area}(B \cap \hat{B})}{\mathrm{area}(B \cup \hat{B})}.$$

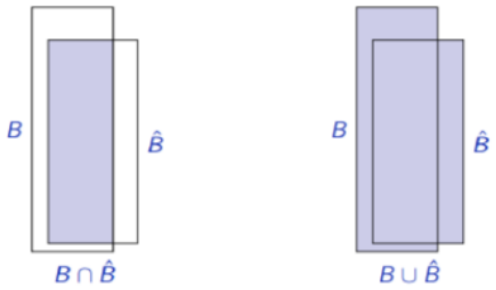


Figure 4.4 – IoU representation. The left part shows the intersection of the two polygons while the right part shows their unions. The IoU is then computed as the ratio between the two found areas. [31][37]

The second metric is a custom-made metric used to discriminate slightly bad registrations from very bad registrations. It is called *relative centroid misplacement* (RCM) and evaluate the distance between the centroid of both polygons with respect to the diagonal size of the fixed image. When two polygons fall one next to each other or very far away, the IoU is equal to 0 in both cases, but the RCM makes a difference between both.

Chapter 5

Results

This chapter will present the different results obtained using the protocols described in Chapter 4.

The codes used are available in this Github :
`https://github.com/Asefy/Annotation-multimodal-biomedical`.

5.1 Single step results

Here below the IoU and RCM for the different annotation groups can be seen in Tables 5.1 and 5.2. The statistics for each group were described at Section 2.2 and an example for each of them in the appendix at Section 8.1.

In those tables the three methods are presented (ORB detector, SIFT detector and mutual information) either with a full affine transform (FA) or a partial affine transform (PA). Those can be compared with the inital values (Init) of the metrics as they were before any registration. The number of pairs corresponds to the possible association $(annotation_1; annotation_2) \longrightarrow (moving\ annotation; fixed\ annotation)$, meaning that a given pair will be counted twice for both registration directions.

Group ID	# of pairs	Init	ORB PA	ORB FA	SIFT PA	SIFT FA	MI PA	MI FA
526756968	600	0.028	**0.820**	0.757	0.724	0.651	0.325	0.025
527083083	600	0.004	**0.597**	0.530	0.539	0.481	0.181	0.004
527268885	210	0.044	0.299	**0.330**	0.243	0.216	0.102	0.040
529103480	210	0.121	**0.770**	0.743	0.701	0.680	0.261	0.118
529104012	210	0.029	**0.609**	0.551	0.504	0.454	0.169	0.028
529108168	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
529118038	306	0.042	**0.801**	0.772	0.797	0.754	0.306	0.023
529119861	6	0.0	0.877	**0.894**	0.817	0.776	0.053	0.0
529121751	380	0.028	**0.688**	0.637	0.619	0.557	0.386	0.028
529123859	380	0.046	**0.864**	0.835	0.832	0.743	0.231	0.044
529125794	110	0.158	0.836	**0.864**	0.824	0.844	0.332	0.127
529129437	132	0.327	0.755	**0.773**	0.710	0.644	0.439	0.281
529830333	210	0.011	**0.427**	0.328	0.405	0.360	0.029	0.014
529832314	110	0.059	0.809	0.808	**0.865**	0.864	0.177	0.051
529836154	2	0.0	0.813	0.881	**0.918**	0.755	0.0	0.0
529837579	240	0.001	0.415	**0.441**	0.412	0.347	0.035	0.001
529839089	240	0.0	0.415	**0.441**	0.433	0.396	0.044	0.0
529840812	210	0.003	0.532	**0.546**	0.483	0.430	0.092	0.005
529842978	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Global	4160	0.041	**0.655**	0.624	0.609	0.555	0.217	0.036

Table 5.1 – IoU per annotation groups for one step registration

Group ID	# of pairs	Init	ORB PA	ORB FA	SIFT PA	SIFT FA	MI PA	MI FA
526756968	600	0.123	**0.004**	0.007	0.027	0.044	0.086	0.124
527083083	600	0.126	**0.004**	0.008	0.030	0.050	0.087	0.127
527268885	210	0.044	**0.016**	0.022	0.069	0.147	0.038	0.047
529103480	210	0.145	**0.004**	0.005	0.013	0.018	0.116	0.139
529104012	210	0.146	**0.007**	0.009	0.014	0.021	0.117	0.143
529108168	2	0.089	**0.023**	0.033	0.027	0.026	0.015	0.100
529118038	306	0.248	**0.010**	0.017	0.012	0.033	0.217	0.242
529119861	6	0.135	**0.001**	**0.001**	0.003	0.004	0.101	0.130
529121751	380	0.094	**0.019**	0.020	0.024	0.051	0.058	0.092
529123859	380	0.089	**0.002**	0.003	0.017	0.064	0.079	0.105
529125794	110	0.034	**0.002**	**0.002**	0.003	**0.002**	0.030	0.044
529129437	132	0.040	**0.008**	0.011	0.014	0.075	0.033	0.045
529830333	210	0.045	**0.033**	0.062	0.187	0.110	0.041	0.050
529832314	110	0.198	0.018	0.017	0.012	**0.011**	0.219	1.396
529836154	2	0.368	0.004	**0.002**	**0.002**	0.008	0.416	0.418
529837579	240	0.146	0.057	**0.056**	0.062	0.069	0.113	0.170
529839089	240	0.148	0.057	**0.056**	0.062	0.071	0.119	0.271
529840812	210	0.140	0.033	**0.031**	0.034	0.049	0.110	0.182
529842978	2	0.290	0.203	0.200	**0.197**	0.209	0.322	0.350
Global	4160	0.122	**0.016**	0.019	0.037	0.055	0.096	0.166

Table 5.2 – RCM per annotation groups for one step registration

The results show first that the feature detectors outperform the pixel-based method used, with an advantage for the ORB detector. The pixel-based method is showing awful results and is actually very unstable (with the full affine version being even worse than the initial metrics). On the other hand, the performances of the feature detectors are quite impressive considering that they work on the pixel value despite the different stains used (even though the images are processed in grayscale). Moreover, they are about 5 times faster than the pixel-based method, with a maximum of 5000 keypoints kept for the ORB detector which is already quite a high setting. The value of 5000 has been chosen as keeping all keypoints tend to give the best result. In this resolution and those kind of images, the number of features often range between 2000 and 4000. The registration process with partial affine has been also tested with a maximum of 2000 keypoints and the results were slightly lower with a global IoU score of 0.633 and a global RCM of 0.018.

For the degrees of freedom of the transform, it seems that restricting to partial affine leads to better results than with the shearing. Both of the above observations are supported by the IoU and the RCM obtained, consistently higher for the IoU and lower for the RCM

for the three methods (on the global average).

About the annotation groups, the annotations with an average size below 0.1% (529108168, 529830333, 529837579, 529839089, 529842978) tend to have worse results for the IoU but also for the RCM which is more surprising. In particular, the last group (529842978) with two annotations has a particularly high RCM (0.2). This can however be explained by the fact that the tissue is duplicated thrice on one of the two images as can be seen in Fig 5.1. While the annotation on the first image is only on one of the three duplicates, the registration process aligns one of the two others with the single tissue in the second image (obviously containing the linked annotation). The IoU is thus of 0 and the RCM depend on the distance between the "good" tissue, with the annotation, and the duplicate whose registered with the tissue in the second image.

Figure 5.1 – Images in last annotation group. The organ is duplicated thrice in the first image such that a successful registration has only one chance out of three to be done with the duplicate containing the annotation.

Another factor that may impact the registration processes is the fact that the fetching of images from Cytomine replaces some transparent areas with black pixels as in the pictures above (Fig 5.1). Features are detected on the corners of the frontier between black and white pixels. As they are not very discriminating, many should be filtered by the cross-checking but the ones that are kept are among the top matches as can be seen in Fig 5.2. Hopefully, the worse matches are removed with such sorting but the scoring does not impact the RANSAC algorithm. If enough features are used, those false features will be removed by it. The fact that those borders get such keypoints have been discovered late as most annotations used at the early stage of this project did not suffer from those black pixel effects. They do not seem to alter that much the registration process thanks to RANSAC. If further works tend to show a more significant impact, one may try to replace black pixels by white pixels. This may help the SIFT detector. On the other hand, the ORB detector will likely rank them lower (as the Harris score is based on the derivatives) but they will keep the same descriptors (or rather the "opposite" descriptor in the binary sense) as the actual white already present in the image is likely not a perfect white. The binary tests will then always give the same kind of results, and thus the matching will remain as perfect as before. The maximum number of features should probably be restricted to a lower value in ORB to have them removed before the matching process. However, it has been said earlier that lowering the number of features kept within ORB also decreases the performances. This would suggest that the cross-checking associated with RANSAC is efficient to filter those keypoints.

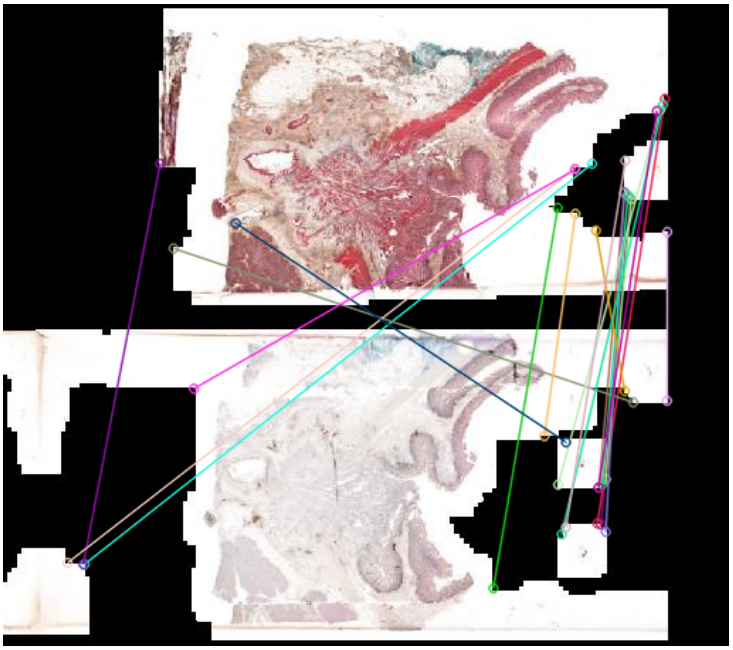
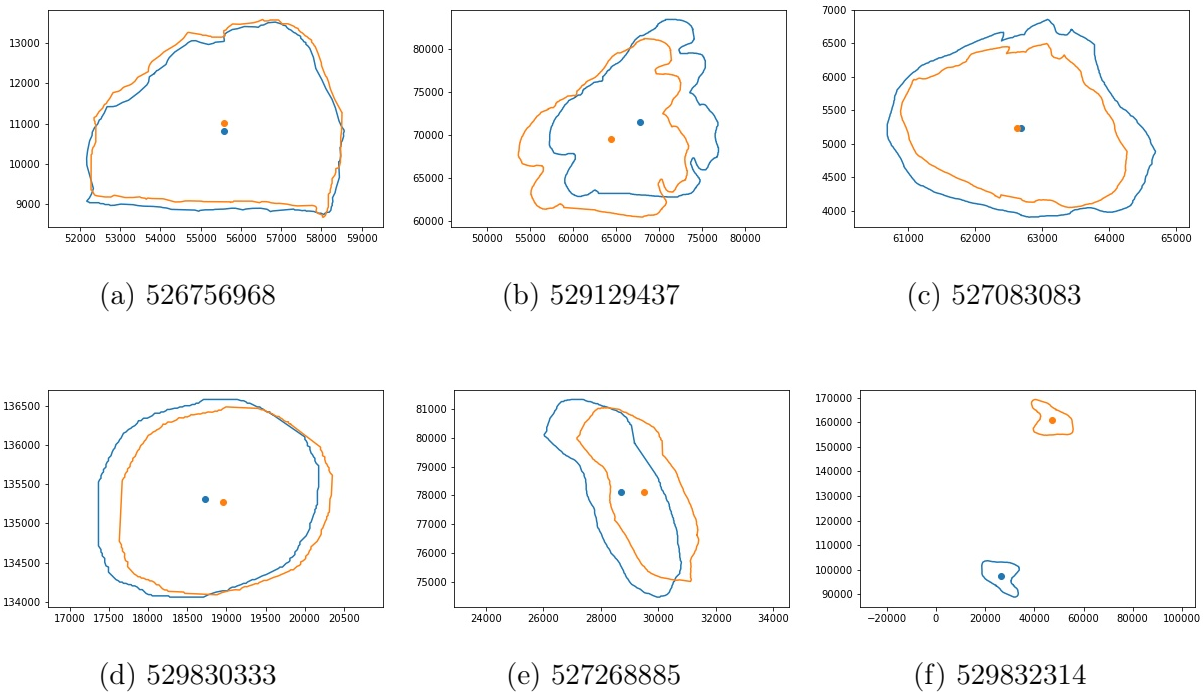


Figure 5.2 – Features matched at black borders (top 20 matches are shown). None of those matches corresponds to the true transformation.

In Fig 5.3, a few examples of registration with partial affine transforms found with the ORB detector are shown. Even though some are really well registered (a), most have a good overlap but suffer from the local variation with respect to the whole tissue (b-c-d-e). These are the annotations expected to be improved by the second local registration. Lastly, some are already very poorly registered (f) and will obviously not be enhanced by a second local registration as the true annotation will not be present in the locality encompassed by the window taken in the fixed image.



(a) 526756968 (b) 529129437 (c) 527083083

(d) 529830333 (e) 527268885 (f) 529832314

Figure 5.3 – Examples of annotation registrations with the ORB detector and partial affine transform. The plots are captioned with their annotation group such that the corresponding subtype of tissue can be seen in Section 8.1.

5.2 Two steps results

Tables 5.3 and 5.4 show the results of the IoU and the RCM for the ORB and SIFT detector (with a partial affine transform) used in two steps, either for pattern-matching methodology (2P) or the "large window" one (2L). Those can be compared with the initial value for the metrics as well as with the results for the ORB detector using a partial affine transform, the best one at the previous step.

Group ID	# of pairs	Init	ORB PA	ORB 2P	ORB 2L	SIFT 2P	SIFT 2L
526756968	600	0.028	**0.820**	0.290	0.673	0.222	0.523
527083083	600	0.004	**0.597**	0.107	0.508	0.120	0.305
527268885	210	0.044	0.299	0.320	**0.408**	0.186	0.203
529103480	210	0.121	**0.770**	0.485	0.757	0.487	0.769
529104012	210	0.029	**0.609**	0.125	0.531	0.147	0.298
529108168	2	0.0	0.0	0.0	0.0	0.0	**0.221**
529118038	306	0.042	**0.801**	0.304	0.738	0.348	0.580
529119861	6	0.0	**0.877**	0.759	0.545	0.396	0.614
529121751	380	0.028	**0.688**	0.109	0.379	0.160	0.262
529123859	380	0.046	**0.864**	0.132	0.649	0.220	0.548
529125794	110	0.158	0.836	0.603	**0.859**	0.517	0.774
529129437	132	0.327	0.755	0.480	**0.833**	0.472	0.618
529830333	210	0.011	0.427	0.041	**0.454**	0.026	0.309
529832314	110	0.059	0.809	0.435	**0.875**	0.431	0.831
529836154	2	0.0	0.813	0.452	0.819	0.791	**0.897**
529837579	240	0.0	0.415	0.277	**0.530**	0.226	0.400
529839089	240	0.0	0.415	0.062	**0.553**	0.105	0.355
529840812	210	0.003	0.532	0.454	**0.735**	0.365	0.618
529842978	2	0.0	0.0	0.0	0.0	0.0	0.0
Global	4160	0.041	**0.655**	0.237	0.600	0.231	0.454

Table 5.3 – IoU per annotation groups for two steps registration

Group ID	# of pairs	Init	ORB PA	ORB 2P	ORB 2L	SIFT 2P	SIFT 2L
526756968	600	0.123	**0.004**	0.030	0.013	0.067	0.049
527083083	600	0.126	**0.004**	0.018	0.008	0.056	0.052
527268885	210	0.044	0.016	0.013	**0.012**	0.080	0.078
529103480	210	0.145	**0.004**	0.025	0.006	0.041	0.012
529104012	210	0.146	**0.007**	0.023	0.012	0.030	0.024
529108168	2	0.089	**0.023**	0.030	0.025	0.034	0.006
529118038	306	0.248	**0.010**	0.037	0.018	0.043	0.028
529119861	6	0.135	**0.001**	0.007	0.019	0.016	0.018
529121751	380	0.094	**0.019**	0.041	0.032	0.054	0.058
529123859	380	0.089	**0.002**	0.044	0.015	0.068	0.047
529125794	110	0.034	**0.002**	0.010	**0.002**	0.015	0.007
529129437	132	0.040	0.008	0.032	**0.004**	0.044	0.058
529830333	210	0.045	**0.033**	0.052	0.048	0.124	0.120
529832314	110	0.198	0.018	0.038	**0.015**	0.039	0.016
529836154	2	0.368	0.004	0.075	0.004	0.005	**0.002**
529837579	240	0.146	**0.057**	0.215	0.207	0.218	0.217
529839089	240	0.148	**0.057**	0.213	0.207	0.205	0.193
529840812	210	0.140	**0.033**	0.164	0.105	0.111	0.088
529842978	2	0.290	**0.203**	0.605	0.601	0.603	0.603
Global	4160	0.122	**0.016**	0.059	0.043	0.079	0.069

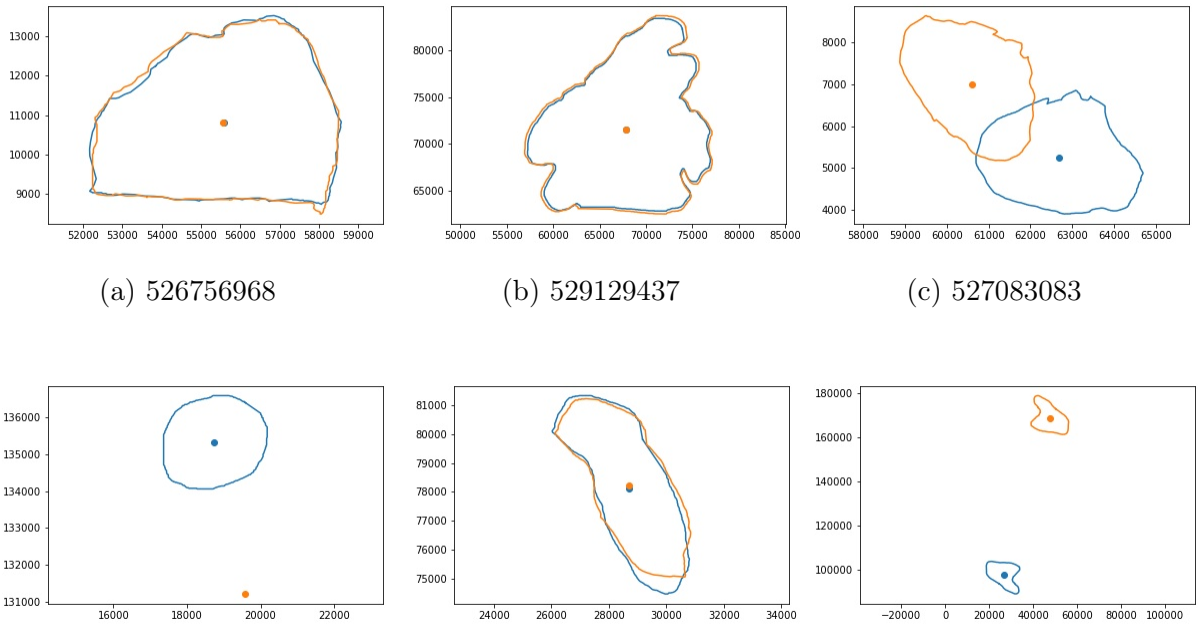
Table 5.4 – RCM per annotation groups for two steps registration

The results for those two steps registrations are quite disappointing, especially for the pattern patching where the performances fall abruptly. The results for the large windows are worse but more comparable to the initial ones. Nevertheless, this account for the IoU while the RCM is better for ORB than for SIFT with both methods. This can be explained by the fact that the second registration is limited to an area around the first registration. The worse RCM achievable is thus dependent of the RCM of the first method which was better for the ORB detector than for the SIFT one.

Some examples, for the same pairs as in the previous Section 5.1, using the two steps (large window) partial affine ORB detector are shown in Fig 5.4. Several cases are derived from them. The annotation that was already well registered (a) is still very well registered (probably even better). Among the ones that were averagely registered, some got a significant improvement (b-e) while some other became worse (c-d). How much worse will vary from slightly (c) to a unexpected behavior where the window is registered at a kind of random place by being hugely squeezed (d), but the reasons causing such strange registration have not been determined precisely. However, this may be due to the fact that the local windows suffer more from the difference in staining. When performing the

global registration, many interesting keypoints can be found at the border of the organs where the external white points are likely to be brighter than any points within the organ whatever the stain used, leading to more common descriptors across modalities. On the other hand, the local windows mainly (or only) see tissues whose colors/intensities depend on the stain used, thus leading to more different descriptors if the stains have very different aspects in grayscale. This behavior actually happens more often with the pattern-matching methodology. Finally, the last annotation (f) which was already very far away is still as far as expected from the method used. Even though there is no case of "perfect" registration in one step that became terrible after the second step, this behavior likely exists as a very good first registration or an averagely good first registration does not make much difference for the large window methodology using a feature detector.

A last remark can be made about the maximal potential of the methods. As can be seen in (e) and in a lower measure in (a), some non-rigid transformations can not be done through an affine transform (and maybe even not with an homography). If such details are needed, which may be debatable, other techniques would be needed (either in addition or in place of the presented category of methods).



(a) 526756968 (b) 529129437 (c) 527083083

(d) 529830333 (e) 527268885 (f) 529832314

Figure 5.4 – Examples of annotation registrations with the two steps ORB detector and partial affine transform for "large patch". The plots are captioned with their annotation group such that the corresponding subtype of tissue can be seen in Section 8.1.

5.3 Stain results

The main idea of this work was to have to annotate only one image (with as many annotations as needed), and then to have all of the annotations done transferred automatically to the other images in the same image group. It is interesting to see which stains (as

moving image where the known annotation is) give the best results to know which images are the best candidates to be the one to be manually annotated. Table 5.5 gives the average results over all the pairings where the stain is used in the moving image for the IoU and the RCM with the best registration method found i.e. ORB 1 step with partial affine). Those tables are for *single stain* meaning that a single image with several stains is counted for all its stains. All stains with less than 10 annotations are filtered out from the table.

Considering the number of different annotations that are used for each, it is hard to determine without any doubt that a stain is clearly better or worse as there IoU's are all roughly between 0.55 and 0.75 which probably does not give differences high enough to consider a classification. The only one that slightly more underperforms is the *he* stain but it is also the one with the worse initial IoU and RCM. The tables for the full stains (mixing of stains considered as a unique one) is available in the appendix (Section 8.3) but the results are similar such that no conclusion can either be made.

Stain name	# of annots	# of pairing	Init IoU	Reg Iou	Init RCM	Reg RCM
muc5ac	13	209	0.031	0.761	0.121	0.007
muc6	13	209	0.032	0.766	0.124	0.007
wt1	15	234	0.047	0.740	0.113	0.005
ca125	15	234	0.047	0.740	0.113	0.00
ki67	14	247	0.026	0.664	0.120	0.015
smad4	10	172	0.043	0.637	0.106	0.022
vim	12	199	0.027	0.641	0.119	0.018
p53	31	424	0.053	0.705	0.120	0.008
cd34	12	190	0.052	0.717	0.114	0.007
cald	31	424	0.053	0.705	0.120	0.008
ck19	26	437	0.038	0.729	0.118	0.006
pdl1	10	181	0.040	0.636	0.165	0.015
d240	20	253	0.056	0.711	0.121	0.009
maspin	20	253	0.056	0.710	0.121	0.009
muc2	11	165	0.040	0.628	0.102	0.017
muc1	11	165	0.040	0.628	0.102	0.017
ck17	23	364	0.027	0.645	0.138	0.017
he	13	175	0.019	0.518	0.205	0.045
ck20	11	189	0.047	0.647	0.108	0.015
ck7	11	189	0.047	0.647	0.108	0.015
ceam	17	282	0.035	0.593	0.128	0.018
cdx2	11	185	0.048	0.548	0.113	0.033
ca199	19	321	0.053	0.593	0.111	0.027
cd146	15	234	0.056	0.567	0.132	0.027
ngfr	15	234	0.056	0.567	0.132	0.027
cd68	10	161	0.064	0.642	0.100	0.016
hmga2	11	180	0.024	0.736	0.131	0.008
cd3	12	209	0.057	0.631	0.098	0.009

Table 5.5 – IoU and RCM per single stain. Both the initial values of the metric (Init) and the the values after using the best registration (Reg) i.e. the global ORB registration with a partial affine transformation are shown.

5.4 Deep learning results

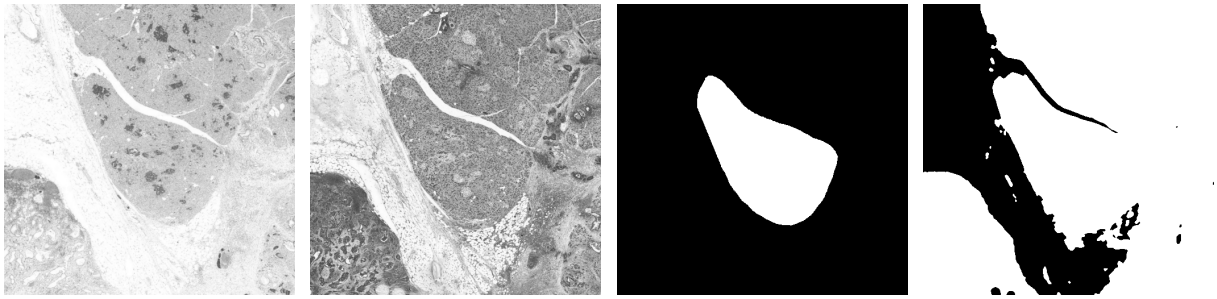
The last part is the U-net training. The different versions are evaluated using the IoU score of the images obtained as described in Section 4.3 and compared with the results from Section 5.1 using the ORB method for a global registration as for the creation of the

deep learning dataset (and the best as well). Note that the measurement is not evaluated exactly in the same way for both. In the deep learning case, the performances are assessed such that only the prediction within the window matters. It is thus slightly in the favor of the deep learning in the case where the window obtained does not encompass the whole annotation. In particular, for the few cases from the test set where the annotation is completely out of the window (38), a prediction of an empty mask gives an IoU of 1 for the deep learning method whereas it is a failure from the viewpoint of the first global registration (and is thus a score of 0 for the ORB method). There are also the pairs that have been removed from the test dataset (41) because it was so bad that a proper window could not be extracted, which likely account for a score of 0 in the ORB case while it is simply not counted in the deep learning score. Unfortunately, the results obtained are still much worse as can be seen in Table 5.6. The score for the ORB is an average of the 5 annotation groups from the images used in the test set weighted with the number of pairs involved (as it is implicitly done in the deep learning case).

Upsample	Loss	Test set IoU score
Convolution	IoU Loss	0.157
Bilinear	IoU Loss	0.226
Convolution	BCE Loss	0.049
Bilinear	BCE Loss	0.142
ORB one step		0.740

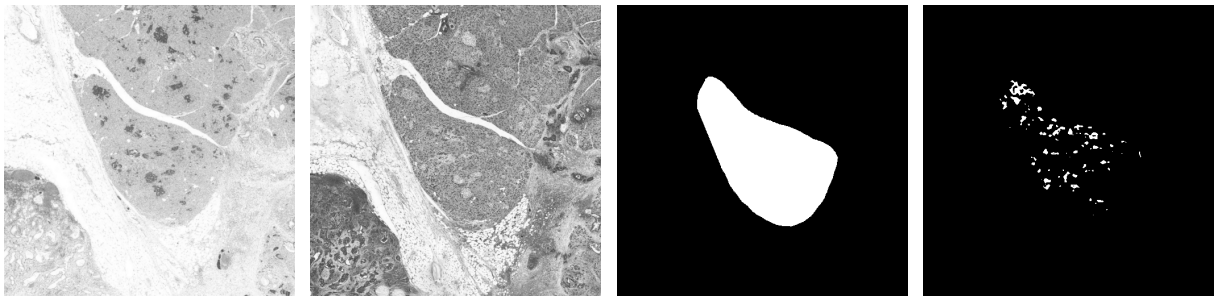
Table 5.6 – IoU score on the test set for the different cases considered as well as the best registration found before.

Concerning the upsampling methods, the bilinear tends to give better IoU scores. For the losses, very different behaviors have been observed. The IoU loss tends to only perform a thresholding of the initial values in the grayscale image (Fig 5.5). On the other hand, the BCE loss often predicts very few (Fig 5.6) to no (Fig 5.7) foreground pixel (annotation segmented) especially in bright grayscale images. The prediction of empty mask is indeed more expected from the BCE loss as it does not favor background from foreground. As most images have much more background than foreground, it is not so surprising that, for a task too hard, it ends up predicting only background. On the other hand, the IoU loss is already trained to have the best IoU score. So even if it finds a way to have a better IoU score than with the BCE loss, the required task is probably as far as for the BCE to be accomplished. It may even be further as, in some rare cases, the BCE loss was able to predict pretty good mask (Fig 5.8) that could probably be corrected by applying a closing. Nevertheless, those good predictions do not seem to be fitted for local registration. The same image has been tested but with a small transformation of the fixed window (3^{nd} input) and its mask (ground truth) to simulate the fact that such a local registration is supposed to be performed. The mask predicted was now a kind of weird intersection between the area of interest in the fixed window (3^{nd} input) and the mask from the first image (2^{nd} input) as can be seen in Fig(5.9).



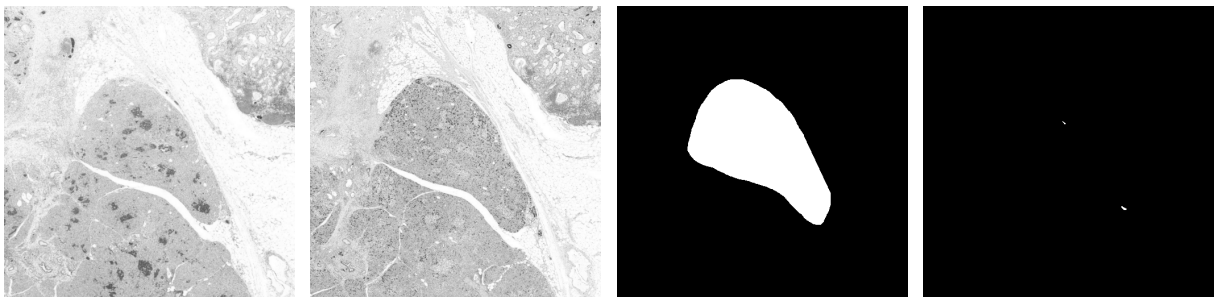
(a) Window from moving image (1^{st} input)
(b) Window from fixed image (3^{rd} input)
(c) Mask to predict in fixed image (ground truth)
(d) Mask predicted in fixed image (output)

Figure 5.5 – Typical prediction with bilinear upsampling and IoU loss assigning foreground pixels to pixels with low intensities. Roughly all low-intensity pixels are classified as foreground.



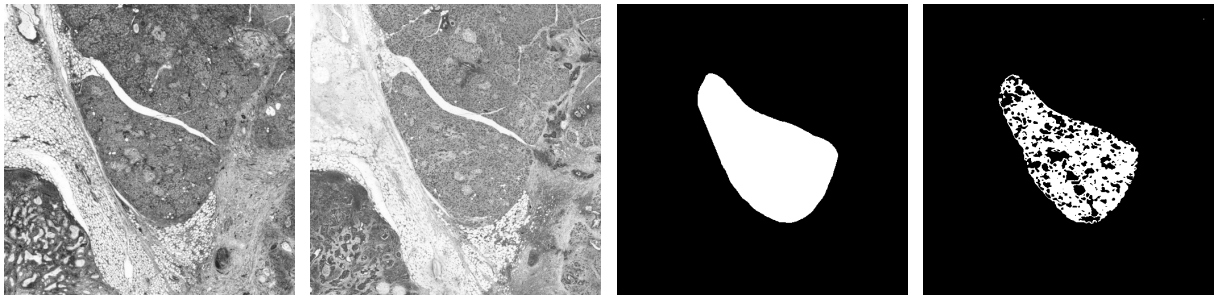
(a) Window from moving image (1^{st} input)
(b) Window from fixed image (3^{rd} input)
(c) Mask to predict in fixed image (ground truth)
(d) Mask predicted in fixed image (output)

Figure 5.6 – Average prediction with bilinear upsampling and BCE loss giving few foreground pixel. Only a few points within the region of interest are classified as foreground.



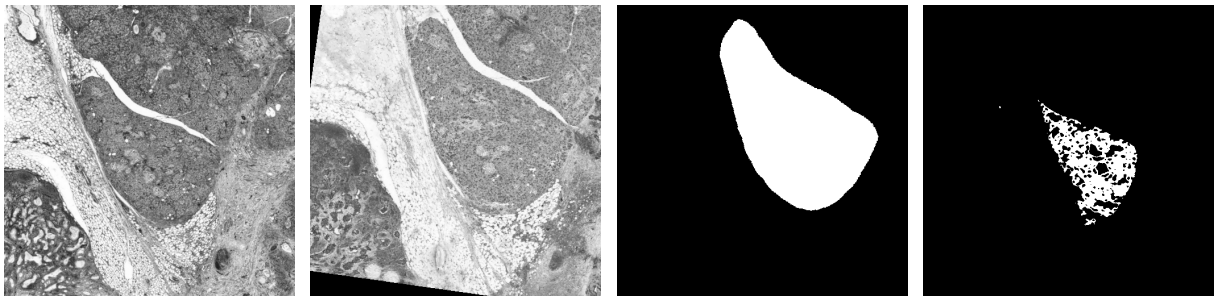
(a) Window from moving image (1^{st} input)
(b) Window from fixed image (3^{rd} input)
(c) Mask to predict in fixed image (ground truth)
(d) Mask predicted in fixed image (output)

Figure 5.7 – Bad prediction with bilinear upsampling and BCE loss giving nearly no foreground pixel. Nearly no points are predicted as foreground.



(a) Window from moving image (1^{st} input)
(b) Window from fixed image (3^{rd} input)
(c) Mask to predict in fixed image (ground truth)
(d) Mask predicted in fixed image (output)

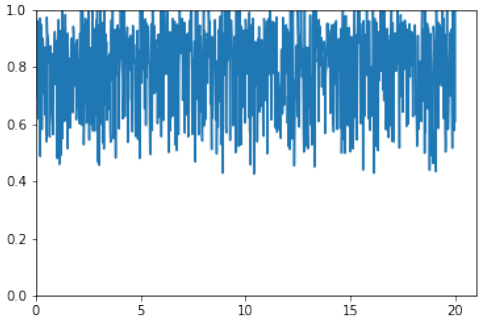
Figure 5.8 – Rare good prediction with bilinear upsampling and BCE loss that could be enhanced by using morphological closing. The shape seems predicted correctly even though it is filled with small holes.



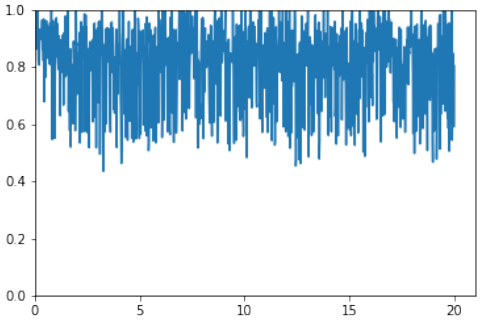
(a) Window from moving image (1^{st} input)
(b) Modified window from fixed image (3^{rd} input)
(c) Modified mask to predict in fixed image (ground truth)
(d) Mask predicted in fixed image (output)

Figure 5.9 – What a good prediction with bilinear upsampling and BCE loss becomes if local registration have to be accomplished by the network. It looks like the intersection of the known region of interest in (a) with the region of interest in (b) that is expected as an output.

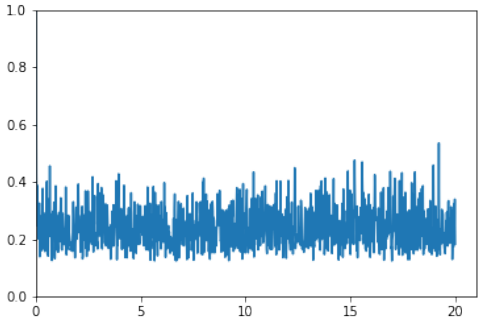
About the number of epochs performed, the training losses (Fig 5.10) and the evaluation scores (Fig 5.11) show that a longer training would more than likely not give better results. The losses for the BCE are much lower than for the IoU loss as it gives more weight to the prediction of background as background.



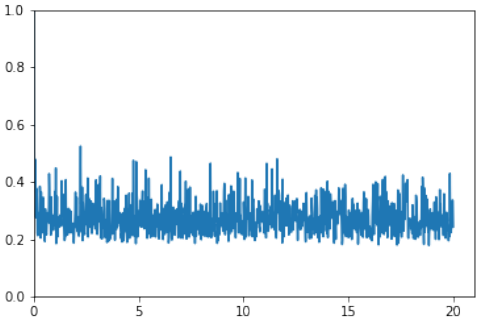
(a) Convolution upsampling & IoU Loss



(b) Bilinear upsampling & IoU Loss

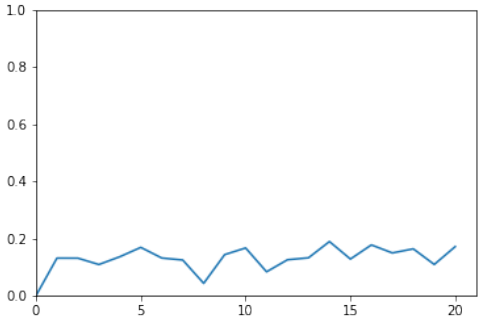


(c) Convolution upsampling & BCE Loss

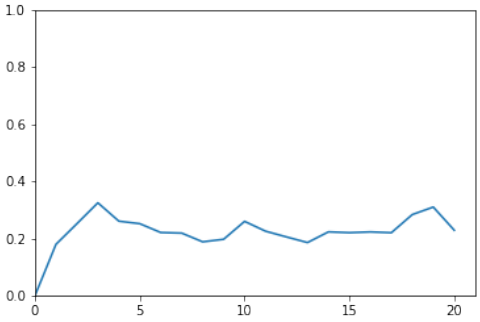


(d) Bilinear upsampling & BCE Loss

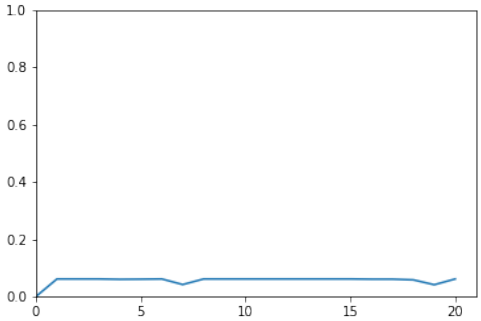
Figure 5.10 – Evolution of the training loss evolution through epochs. The value varies through samples but no evolution is made through time. The BCE losses have a lower value as they are rewarded for good prediction of the background while the IoU loss only focuses on foreground prediction.



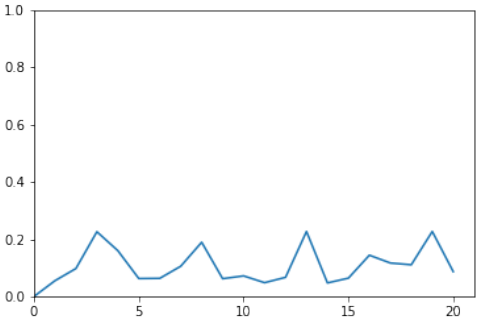
(a) Convolution upsampling & IoU Loss



(b) Bilinear upsampling & IoU Loss



(c) Convolution upsampling & BCE Loss



(d) Bilinear upsampling & BCE Loss

Figure 5.11 – Evolution of the IoU score for the evaluation set through epochs. No progress seems to appear after the second epoch in any of the model trainings.

Given the poor results obtained, the masks could not be converted to polygons to compare with the other methods. Even though closing and opening may help, they would obviously not be enough to recover a proper polygon.

Chapter 6

Limitations & perspectives

The main difficulty of this work was the general task it is trying to solve. Many recent papers on the registration of biomedical images relies on deep learning to perform task more accurately, but only with two given type of stains [9]. Some of them use network to virtually stain one modality to the other [4][6][54] or to convert both modalities to a new more representative one [3]. Working on many stains without distinction makes those kind of tasks much harder. Some others work on more stains but for specific type of organs [16][55] or with "easier" annotation such as landmarks [17]. The problem tackled here more or less combines the different difficulties encountered in those papers.

The second main limitation of this work was the dataset used. As it has been stated in the presentation (Section 2.2), the dataset is still quite small and was smaller at the beginning of this project. Despite a good variety in staining, it probably still lacks diversity in annotation groups which may be particularly limiting for machine learning approaches (such as deep learning). Additionally, the transformations between images were not known (as ground truth) and the reference points (the annotations) were too few per image and often too small, knowing the highly non-rigid local behaviour of the slicing procedure, to be trusted to build such a reference. Deep learning techniques to find a transformation were then not really applicable.

There are still improvements that can be made on this work. First the usage of another colorspace than grayscale could be tried as the *Haematoxylin* [16]. For data augmentation, *channel swapping* or *color transfer* from the same paper may be tried for deep learning approaches.

Considering the kind of failure seen in the second registration, it may be improved by disabling the scaling for the second transform research. This may need to extract the scale factor from the first registration to start from the more likely scaling between both annotations.

Another way of improvement may be to combine methods and/or annotation. As it has been seen in the Chapter 5, the one step method often had average results while the two steps often had perfect or terrible results. One could decide that the first one is a good approximation and accept the second if it overlaps enough the first, or if the area covered by the two steps is not too small or too big compared to the one step. A quick try using

an overlap threshold[1] of $\frac{1}{2}$ was able to perform an higher IoU with a score of 0.692 (while the best was 0.655), the RCM remains at 0.016.

Combining annotations may also be possible, even though it relaxes the initial constraint which was to only have one image to manually annotate. Grouping methods such as polling could then be used. A registration would be performed with, for example, the annotations from three different images manually annotated and their results would be combined into a single polygon. Annotating three images instead of one is annoying but still much less than to annotate 10 to 30 images.

Finally a last word could be made on the deep learning approach. It seems clear that it does not work to perform local registration, and it may even still had bad performances with a greater number of annotation groups. Nevertheless, it may be still an idea to make very local non-rigid correction due to slicing if a very good first registration is performed. In this case data augmentation would be applied commonly on the 2 images (and their respective masks) instead of separately for both images. It may be interesting to modify the data augmentation process in such case. Color augmentation could also be applied, as mentioned above, and one may forbid translation such that the annotations are always centered.

If one would use the methods presented in this work, he probably should at least add some translations and rotations to the annotation transferred in most case. Allowing the second step registration with the IoU agreement (as described above) may increases the number of times the transferred annotation is directly placed accurately. The time needed to perform the annotations would definitely not be divided by several factors compared to the current system, but some times could probably still be saved. However, none of these methods performs the non-rigid local correction as the deep learning strategy, the only one susceptible to do so, is currently not convincing. In the cases those non-rigid local transformation were to be mandatory for the following tasks, the current techniques would not be of any help.

[1] A threshold of $\frac{3}{4}$ has also been tried but leads to a smaller improvement with an IoU of 0.673 and again the same RCM of 0.016.

Chapter 7

Conclusion

Annotating histological images is a very time-consuming task which can not be done by anybody. In the context of the multimodality, the aim of this work was to alleviate the issue by allowing to transpose automatically polygon annotations from one modality to another. To do so, several techniques of global registration have been tried on images from many stains converted to their grayscale counterpart.

The feature-based detectors ORB and SIFT and a pixel-based method performing a gradient descent on a differentiable mutual information (based on the joint histogram of the images) approaches are tried. From this comparison the feature-based detectors outperform the gradient descent and among them the ORB detector was the best. The usage of a full affine transform or an affine transform without shearing has been discussed and it has been shown that allowing more degrees of freedom was not always the best choice.

A second local registration, using a second time the same technique, has been tried either by registering the annotation alone to the second image surrounding or the surrounding in both images. In this framework, the registration with both surroundings was the more stable but neither of them gives solely a better performance than the registration in one step as the non-rigid modifications due to the slicing were more present locally.

In the same purpose of second local registration, a deep learning approach has been tried based on the well-known segmentation network U-net. The polygon, represented by a mask, for the second image was being predicted by combining the surrounding in both images and the mask of the annotation in the first. Two losses and two upsampling techniques has been tried for the network but none gives satisfactory results.

In conclusion, the transposition of annotation for any stains, any tissue and any cell with a single framework is a difficult task. Among the techniques used, the feature detectors have shown surprisingly good results despite the different stains. They moreover tend to be quite stable within the different stains used. Nevertheless, those results are still to be improved to have a real potential to replace human workforce in a blind manner.

Bibliography

[1] D. Lahat, T. Adali, and C. Jutten, "Multimodal data fusion: An overview of methods, challenges, and prospects," *Proceedings of the IEEE*, vol. 103, no. 9, pp. 1449–1477, 2015.

[2] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," 2017.

[3] N. Pielawski, E. Wetzer, J. Ofverstedt, J. Lu, C. Wahlby, J. Lindblad, and N. Sladoje, "Comir: Contrastive multimodal image representation for registration," *ArXiv*, vol. abs/2006.06325, 2020.

[4] Y. Rivenson, H. Wang, Z. Wei, K. de Haan, Y. Zhang, Y. Wu, H. Gunaydin, J. Zuckerman, T. Chong, A. Sisk, L. Westbrook, W. Wallace, and A. Ozcan, "Virtual histological staining of unlabelled tissue-autofluorescence images via deep learning," *Nature Biomedical Engineering*, vol. 3, 06 2019.

[5] Y. Rivenson, K. de Haan, W. Wallace, and A. Ozcan, "Emerging advances to transform histopathology using virtual staining," *BME Frontiers*, vol. 2020, pp. 1–11, 08 2020.

[6] C. Mercan, G. Mooij, D. Tellez, J. Lotz, N. Weiss, M. van Gerven, and F. Ciompi, "Virtual staining for mitosis detection in breast histopathology," in *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*, 2020, pp. 1770–1774.

[7] D. Hong, L. Gao, N. Yokoya, J. Yao, J. Chanussot, Q. Du, and B. Zhang, "More diverse means better: Multimodal deep learning meets remote-sensing imagery classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 5, p. 4340–4354, May 2021. [Online]. Available: http://dx.doi.org/10.1109/TGRS.2020.3016820

[8] M. Dalla Mura, S. Prasad, F. Pacifici, P. Gamba, J. Chanussot, and J. A. Benediktsson, "Challenges and opportunities of multimodality and data fusion in remote sensing," *Proceedings of the IEEE*, vol. 103, no. 9, pp. 1585–1601, 2015.

[9] P. Paul-Gilloteaux and M. Schorb, "Correlating data from imaging modalities," in *Correlative Imaging*. John Wiley & Sons, Ltd, 2019, ch. 11, pp. 191–210.

[10] H. McGurk and J. MacDonald, "Hearing lips and seeing voices," *Nature*, vol. 264, no. 5588, pp. 746–748, Dec. 1976.

[11] The royal college of pathologists. What is histopathology? Accessed: 2021-07. [Online]. Available: https://www.rcpath.org/discover-pathology/news/fact-sheets/histopathology.html

[12] Polysciences. Staining - histology/cytology. Accessed: 2021-07. [Online]. Available: https://www.polysciences.com/german/catalog-products/histology-cytology-pathology/staining-histology-cytology

[13] R. Marée, L. Rollus, B. Stevens, R. Hoyoux, G. Louppe, R. Vandaele, J.-M. Begon, P. Kainz, P. Geurts, and L. Wehenkel, "Collaborative analysis of multi-gigapixel imaging data using cytomine," *Bioinformatics*, vol. 32, pp. 1395 – 1401, 2016.

[14] Cytomine website. [Online]. Available: https://cytomine.be/

[15] Cytomine api documentaion. [Online]. Available: https://doc.uliege.cytomine.org/dev-guide/api/

[16] T. Lampert, O. Merveille, J. Schmitz, G. Forestier, F. Feuerhake, and C. Wemmert, "Strategies for training stain invariant cnns," in *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, 2019, pp. 905–909.

[17] J. Borovec, J. Kybic, I. Arganda-Carreras, D. V. Sorokin, G. Bueno, A. V. Khvostikov, S. Bakas, E. I.-C. Chang, S. Heldmann, K. Kartasalo, L. Latonen, J. Lotz, M. Noga, S. Pati, K. Punithakumar, P. Ruusuvuori, A. Skalski, N. Tahmasebi, M. Valkonen, L. Venet, Y. Wang, N. Weiss, M. Wodzinski, Y. Xiang, Y. Xu, Y. Yan, P. Yushkevich, S. Zhao, and A. Muñoz-Barrutia, "Anhir: Automatic non-rigid histological image registration challenge," *IEEE Transactions on Medical Imaging*, vol. 39, no. 10, pp. 3042–3052, 2020.

[18] Karolinska institute website. [Online]. Available: https://ki.se/en

[19] L. Simonot and M. Hébert, "Between additive and subtractive color mixings: Intermediate mixing models," *Journal of the Optical Society of America. A, Optics, image science, and vision*, vol. 31, pp. 58–66, 02 2014.

[20] Red. The bayer sensor strategy. Accessed: 2021-07. [Online]. Available: https://www.red.com/red-101/bayer-sensor-strategy

[21] Wikipedia. Gamma correction. Accessed: 2021-07. [Online]. Available: https://en.wikipedia.org/wiki/Gamma_correction

[22] ——. Mathematical morphology. Accessed: 2021-07. [Online]. Available: https://en.wikipedia.org/wiki/Mathematical_morphology

[23] M. V. Droogenbroeck and P. Latour, "Computer vision lecture notes uliège," 2019.

[24] Wikipedia. Bilinear interpolation. Accessed: 2021-07. [Online]. Available: https://en.wikipedia.org/wiki/Bilinear_interpolation

[25] ——. Lanczos resampling. Accessed: 2021-07. [Online]. Available: https://en.wikipedia.org/wiki/Lanczos_resampling

[26] L. Ibanez, W. Schroeder, L. Ng, J. Cates, and I. Consortium, "The itk software guide, second edition, updated for itk version 2.4," 11 2005.

[27] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science (New York, N.Y.)*, vol. 220, pp. 671–80, 06 1983.

[28] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, p. 381–395, Jun. 1981. [Online]. Available: https://doi.org/10.1145/358669.358692

[29] Wikipedia. Binary entropy. Accessed: 2021-07. [Online]. Available: https://en.wikipedia.org/wiki/Binary_entropy_function

[30] ——. Mutual information. Accessed: 2021-07. [Online]. Available: https://en.wikipedia.org/wiki/Mutual_information

[31] G. Louppe, "Deep learning lecture notes uliège," 2020.

[32] R. Nagyfi. The differences between artificial and biological neural networks. Accessed: 2021-07. [Online]. Available: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

[33] J. Nazzal, I. El-Emary, and S. Najim, "Multilayer perceptron neural network (mlps) for analyzing the properties of jordan oil shale," *World Applied Sciences Journal*, vol. 5, 01 2008.

[34] P. Geurts and L. Wehenkel, "Introduction to machine learning lecture notes uliège," 2018.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[36] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a nash equilibrium," *CoRR*, vol. abs/1706.08500, 2017. [Online]. Available: http://arxiv.org/abs/1706.08500

[37] F. Fleuret, "Deep learning lecture notes unige."

[38] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[39] S. Santurkar, D. Tsipras, A. Ilyas, and A. Mądry, "How does batch normalization help optimization?" in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 2488–2498.

[40] P. Thevenaz and M. Unser, "Optimization of mutual information for multiresolution image registration," *IEEE Transactions on Image Processing*, vol. 9, no. 12, pp. 2083–2099, 2000.

[41] R. Xu, Y.-W. Chen, S. Tang, S. Morikawa, and Y. Kurumi, "Parzen-window based normalized mutual information for medical image registration," *IEICE Transactions*, vol. 91-D, pp. 132–144, 01 2008.

[42] D. Mattes, D. Haynor, H. Vesselle, T. Lewellyn, and W. Eubank, "Nonrigid multimodality image registration," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 4322, 07 2001.

[43] T. Briand and P. Monasse, "Theory and practice of image b-spline interpolation," *Image Processing On Line*, vol. 8, pp. 99–141, 07 2018.

[44] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–, 11 2004.

[45] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571, 11 2011.

[46] Opencv website. [Online]. Available: https://opencv.org/

[47] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417.

[48] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443.

[49] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792.

[50] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, 1988, pp. 147–151.

[51] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[52] Simpleitk website. [Online]. Available: https://simpleitk.org/

[53] Pytorch website. [Online]. Available: https://pytorch.org/

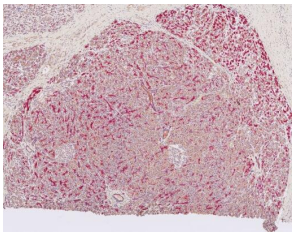
[54] Y. Rivenson, K. de Haan, W. Wallace, and A. Ozcan, "Emerging advances to transform histopathology using virtual staining," *BME Frontiers*, vol. 2020, pp. 1–11, 08 2020.

[55] O. Merveille, T. Lampert, J. Schmitz, G. Forestier, F. Feuerhake, and C. Wemmert, "An automatic framework for fusing information from differently stained consecutive digital whole slide images: A case study in renal histology," *Computer Methods and Programs in Biomedicine*, vol. 208, p. 106157, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169260721002315

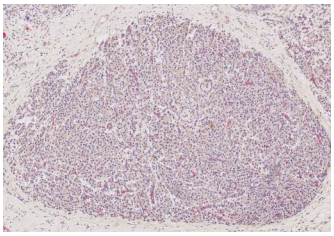
Chapter 8

Appendix

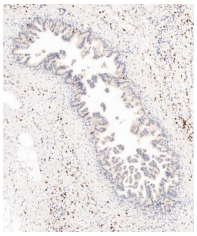
8.1 Example from each annotation group



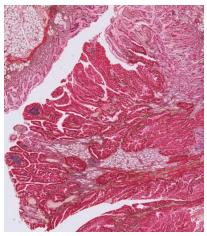
(a) 526756968 - 0.436%



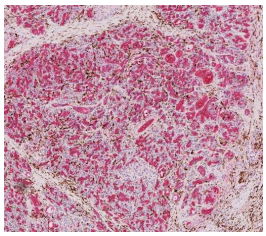
(b) 527083083 - 0.113%



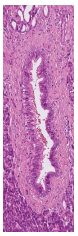
(c) 527268885 - 0.078%



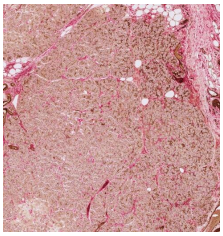
(d) 529103480 - 1.106%



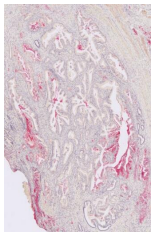
(e) 529104012 - 0.124%



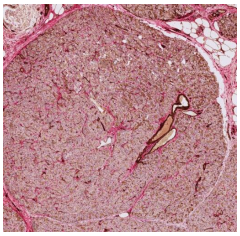
(f) 529108168 - 0.019%



(g) 529118038 - 0.398%

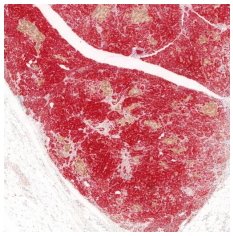


(h) 529119861 - 0.559%

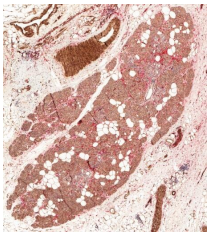


(i) 529121751 - 0.299%

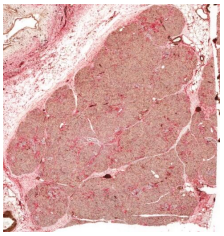
Figure 8.1 – First annotation of each AnnotationLinkCollection (and the proportion of the area cover by the annotation) [1/2]



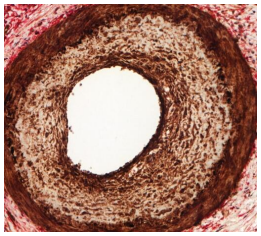
(a) 529123859 - 0.547%



(b) 529125794 - 0.453%

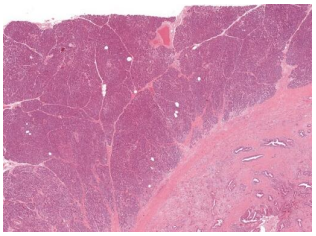


(c) 529129437 - 1.439%

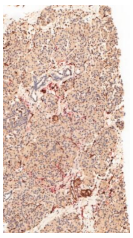


(d) 529830333 - 0.038%

(e) 529832314 - 0.761%



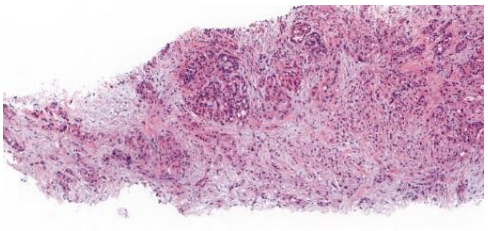
(f) 529836154 - 1.841%



(g) 529837579 - 0.048%

(h) 529839089 - 0.038%

(i) 529840812 - 0.224%

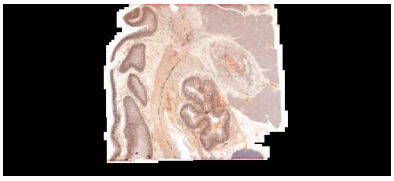


(j) 529842978 - 0.058%

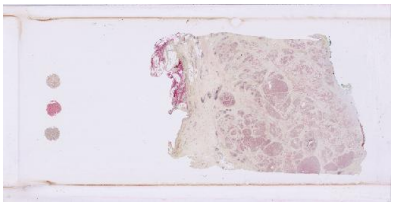
Figure 8.2 – First annotation of each AnnotationLinkCollection (and the proportion of the area cover by the annotation) [2/2]

8.2 Stains example

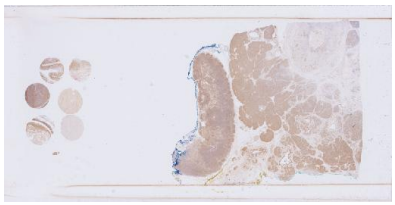
(a) muc5ac_muc6 : PKR-10-muc5ac_muc6



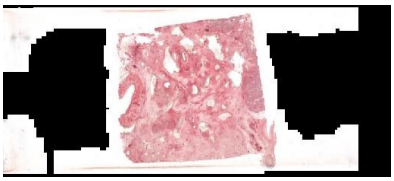
(b) wt1_ca125 : PKR-9-wt1_ca125



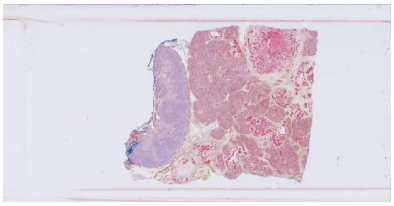
(c) ki67_p16 : PKR-2-ki67_p16



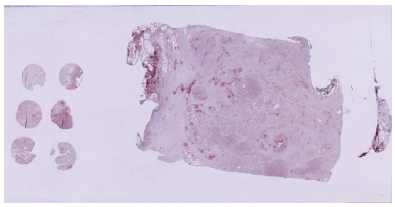
(d) smad4 : PKR-4-smad4



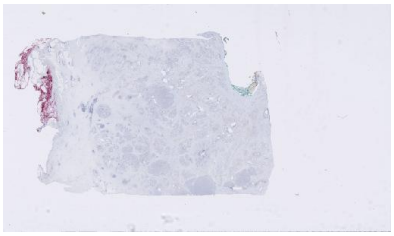
(e) ki67_vim : PKR-8-ki67_vim



(f) p53_cd34_cald_ck19 : PKR-4-p53_cd34_cald_ck19

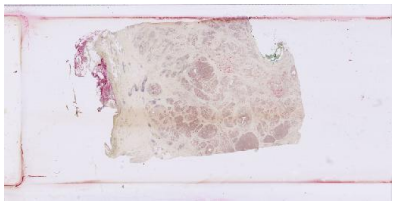


(g) pdl1_cd8 : PKR-2-pdl1_cd8

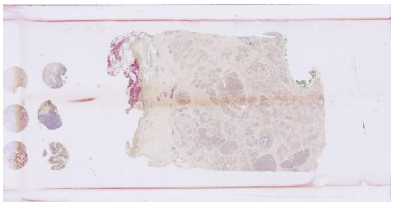


(h) pdl1 : PKR-2-pdl1

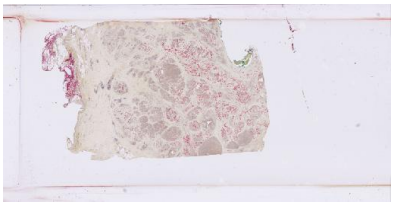
(i) p53_d240_cald_maspin : PKR-9-p53_d240_cald_maspin



(j) p40_ck5 : PKR-2-p40_ck5

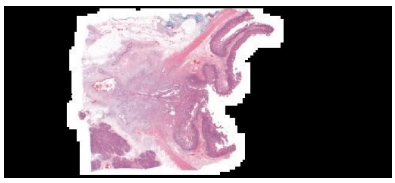


(k) muc2_muc1 : PKR-2-muc2_muc1

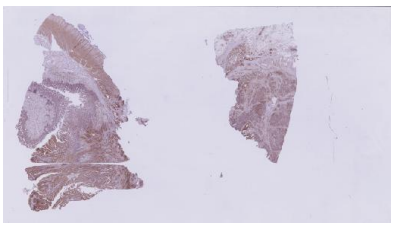


(l) p63_ck17 : PKR-2-p63_ck17

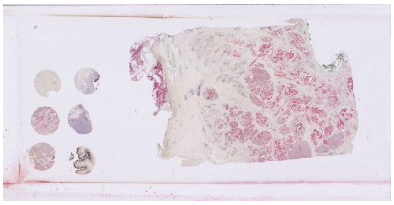
Figure 8.3 – Example of each stain [1/4]



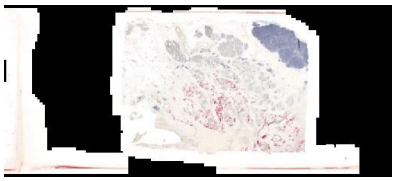
(a) he : PKR-7-he



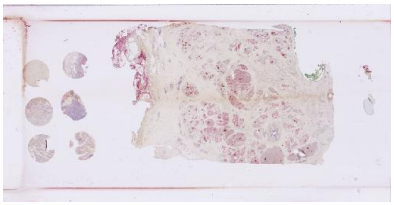
(b) ent1 : PKR-1-ent1



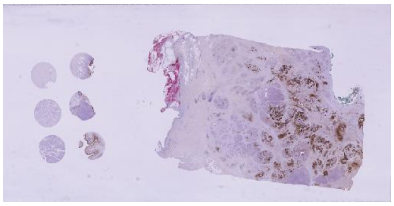
(c) ck20_ck7 : PKR-2-ck20_ck7



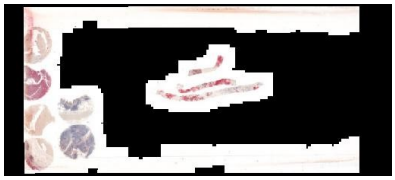
(d) ck17_ceam : PKR-6-ck17_ceam



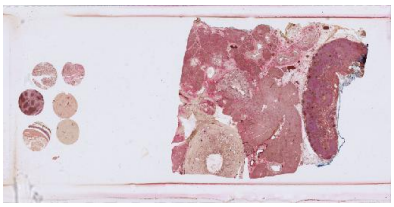
(e) chra_cd56 : PKR-2-chra_cd56



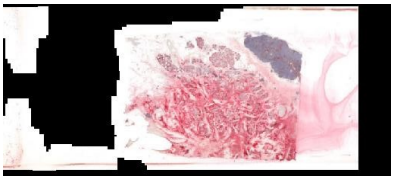
(f) ceam : PKR-2-ceam



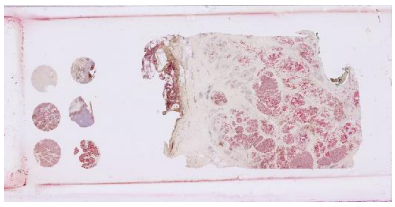
(g) cdx2_ca199 : PKR-10-cdx2_ca199



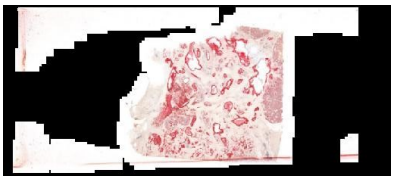
(h) cd146_ngfr : PKR-4-cd146_ngfr



(i) cd68_ca199 : PKR-6-cd68_ca199



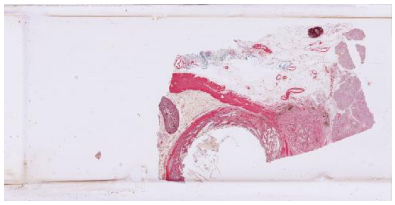
(j) cd15_ck19 : PKR-2-cd15_ck19



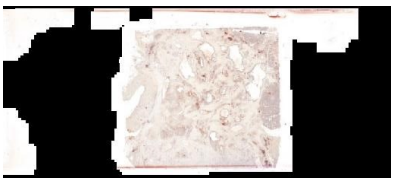
(k) hmga2_ck19 : PKR-8-hmga2_ck19

(l) cd3_cd20 : PKR-2-cd3_cd20

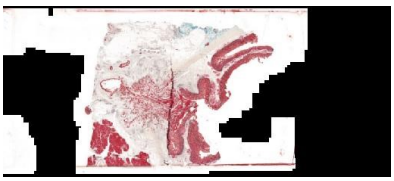
Figure 8.4 – Example of each stain [2/4]



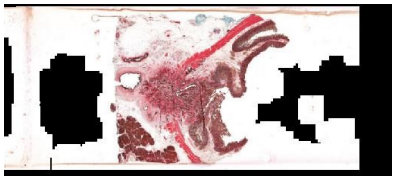
(a) cd3_actinsm : PKR-3-cd3_actinsm



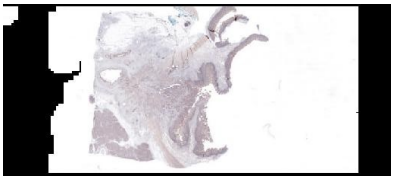
(b) cd4_cd8 : PKR-8-cd4_cd8



(c) cd68_ck18 : PKR-7-cd68_ck18

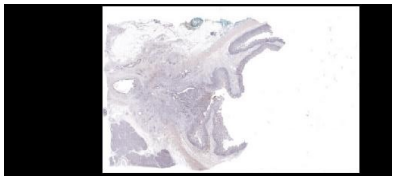


(d) ck18_actinsm : PKR-7-ck18_actinsm



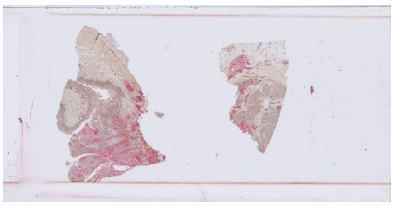
(e) foxp3 : PKR-7-foxp3

(f) m30 : PKR-7-m30

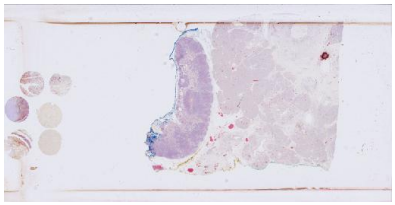


(g) mpo : PKR-7-mpo

(h) muc2_muc1 : PKR-1-muc2_muc1

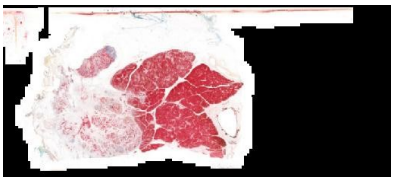


(i) 1wt1_ca125 : PKR-1wt1_ca125



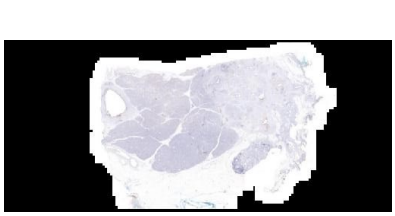
(j) chroma_cd56 : PKR-4-chroma_cd56

(k) ck5_cd10 : PKR-4-ck5_cd10

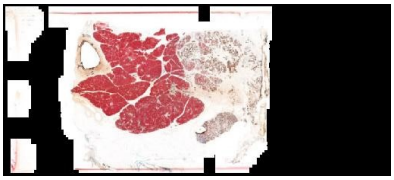


(l) berep4_ema : PKR-5-berep4_ema

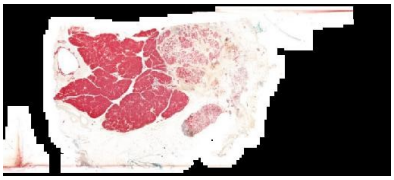
Figure 8.5 – Example of each stain [3/4]



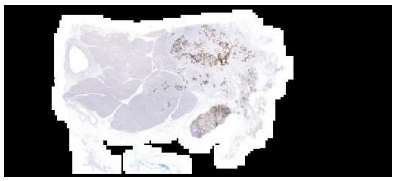
(a) cd10 : PKR-5-cd10



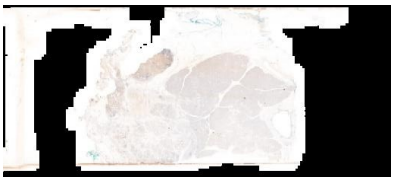
(b) ck19_ck18 : PKR-5-ck19_ck18



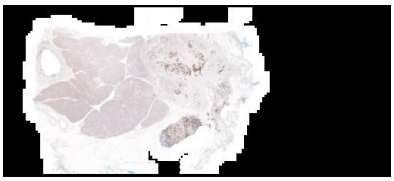
(c) d240_ck18 : PKR-5-d240_ck18



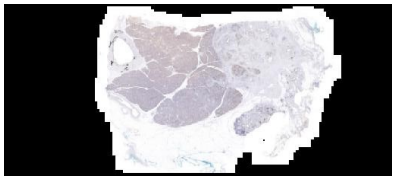
(d) maspin : PKR-5-maspin



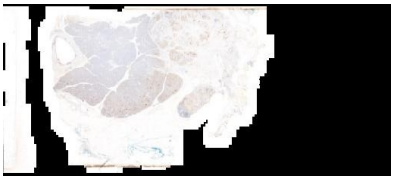
(e) mesoth : PKR-5-mesoth



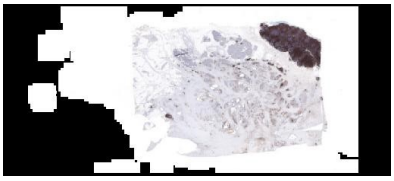
(f) muc5ac : PKR-5-muc5ac



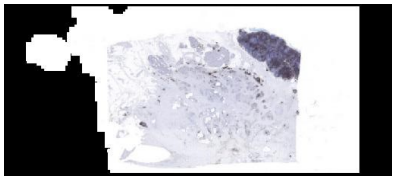
(g) muc6 : PKR-5-muc6



(h) pdx1 : PKR-5-pdx1



(i) cd4 : PKR-6-cd4



(j) cd8 : PKR-6-cd8

(k) trypsin : PKR-10-trypsin

Figure 8.6 – Example of each stain [4/4]

8.3 Stain results (full names)

Stain name	# of annots	# of pairing	Init	ORB PA
muc5ac_muc6	12	190	0.033	0.752
wt1_ca125	15	234	0.047	0.740
smad4	10	172	0.043	0.637
ki67_vim	12	199	0.027	0.641
p53_cd34_cald_ck19	12	190	0.052	0.717
pdl1	8	133	0.041	0.597
p53_d240_cald_maspin	19	234	0.055	0.696
muc2_muc1	11	165	0.040	0.628
he	13	175	0.019	0.518
ck20_ck7	11	189	0.047	0.647
ck17_ceam	15	234	0.038	0.575
cdx2_ca199	11	185	0.048	0.548
cd146_ngfr	15	234	0.056	0.567
cd68_ca199	8	136	0.060	0.655
hmga2_ck19	11	180	0.024	0.736
cd3_cd20	9	144	0.077	0.647

Table 8.1 – IoU per stain

Stain name	# of annots	# of pairing	Init	ORB PA
muc5ac_muc6	12	190	0.124	0.008
wt1_ca125	15	234	0.113	0.005
smad4	10	172	0.106	0.022
ki67_vim	12	199	0.119	0.018
p53_cd34_cald_ck19	12	190	0.114	0.007
pdl1	8	133	0.192	0.020
p53_d240_cald_maspin	19	234	0.126	0.010
muc2_muc1	11	165	0.102	0.017
he	13	175	0.205	0.045
ck20_ck7	11	189	0.108	0.015
ck17_ceam	15	234	0.130	0.021
cdx2_ca199	11	185	0.113	0.033
cd146_ngfr	15	234	0.132	0.027
cd68_ca199	8	136	0.109	0.017
hmga2_ck19	11	180	0.131	0.008
cd3_cd20	9	144	0.083	0.011

Table 8.2 – RCM per stain