Master's Thesis : Cytomine modules for multispectral data analysis

Auteur : Mathy, Lionel
Promoteur(s) : Maree, Raphael
Faculté : Faculté des Sciences appliquées
Diplôme : Master en science des données, à finalité spécialisée
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/11241

Master's Thesis

Cytomine modules for multispectral data analysis

Author:
Lionel Mathy

Supervisor:
Raphaël Marée

Thesis performed in order to obtain the grade of master in data science

University of Liège
Faculty of Applied Science

Academic year 2020-2021

Abstract

Computer vision has always been a challenging fields. Many success have been obtained with sophisticated techniques. However, the vast majority of the algorithms developed last decades only uses images with 1 channel (black and white image) or 3 channels (colored images). However, the light spectrum is larger than this and all these other frequencies may contain very useful information for some machine learning applications. The goal of this master thesis is to exploit machine learning algorithms applied to multispectral images. Some random forests algorithms have been successfully applied for histology classification. We show that some pre-processing steps are required, but others are not. Also, some hyper-parameters were tested in a grid search way in order to find the optimal values of these parameters. We also show that having a lot of frequencies is good, but for some applications, a good selection of frequencies provided to machine learning algorithm allows to obtain high accuracy, which allows for improving the throughput of the whole process in the future.

Acknowledgements

First, I would like to thank the supervisor of this master's thesis, Raphaël Marée. He gave me lots of good advice and support throughout this thesis. He was always encouraging me, and was always comprehensive when difficulties arose.

Also, I do not forget all the professors and assistants thanks to whom I learned many subjects in a wide variety of fields in computer and data science throughout my studies. Without the skills they taught to me, I would have been unable to create this master's thesis.

Contents

Chapter 1

Motivations of the thesis

1.1 Introduction

In data science, and more generally in computer science, the field of computer vision has always been considered as a highly challenging field but also one of the most promising in term of potential applications. The main difficulty comes from the fact that even if we, as humans, consider ourselves generally as good at observing and analysing many situations, we are however unable to explain how we achieve that. For example, in object recognition, we are most of the times perfect at differentiating different objects, without being able to explain deeply why we are so confident in our choice. Therefore, it is highly difficult to create algorithms that are good at computer vision tasks.

Many techniques have been developed throughout the years for different computer vision tasks, with some success. However, the development of machine learning techniques applied to some highly challenging tasks has revolutionised the field of computer vision, improving significantly the accuracy at such tasks. The most famous example of this revolution is the image classification task. Thanks to huge datasets, such as the ImageNet dataset, and many sophisticated deep neural networks trained for these images, the accuracy of such deep learning algorithms have equalised and even outperformed human performance according to some metrics. For example, the best NASNet architecture network is reported to have a top-1 accuracy of 82.7 % and a top-5 accuracy of 96.2 % (source: [1]).

1.2 Multispectral Data

When processing images through computer vision algorithms, one of the key aspect is the way the images are stored. In classical computer vision, images are

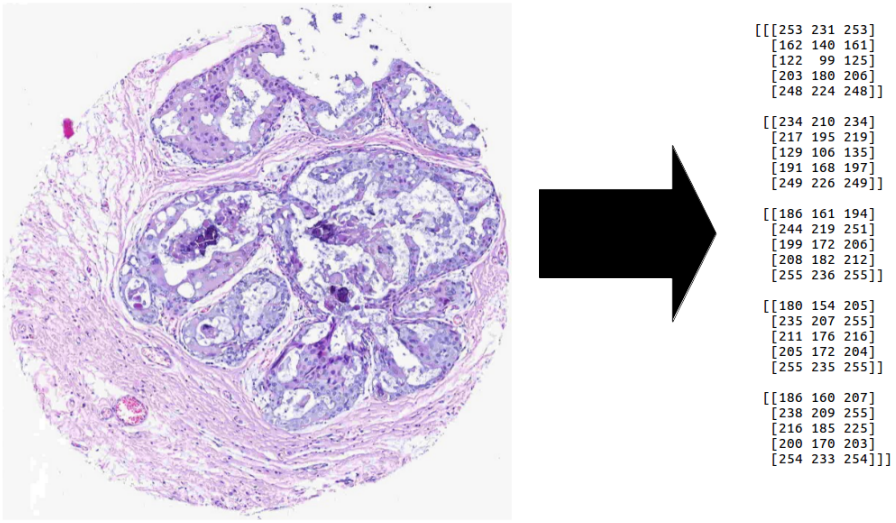

Figure 1.1: An RGB (red-green-blue) image with its representation in a computer memory. (Source of the left image: [2])

often stored in RGB format (for red, green and blue). A picture can be found in figure 1.1 showing an example of such RGB image with its representation in a computer memory. An RGB picture is therefore a third-dimension matrix where two axes represent the abscissa and ordinate coordinates. The third axis is the channels axis, each channel representing one of the red, green and blue colour. For each pixel coordinate, there are therefore 3 values representing the intensities of the red, green and blue colours.

One can argue that humans can perceive more colours than these 3 colours. Indeed, the range of visible light and associated colours is more widespread, as shown in figure 1.2. In fact, only red, green, and blue, called primary colours, are sufficient to generate all colours perceived by the human eye thanks to the principle of additive colour mixing, as shown in figure 1.3. Actually, each human eye is constituted of 3 types of cones. Each type of cones is sensitive to one of the three primary colours. It means that each type of cones is good at capturing photons whose wavelengths are near the wavelength of their respective colours. The human brain is able to create colours with the level of excitation of each type of cone. One of the consequences of that is that superposing two colours can be

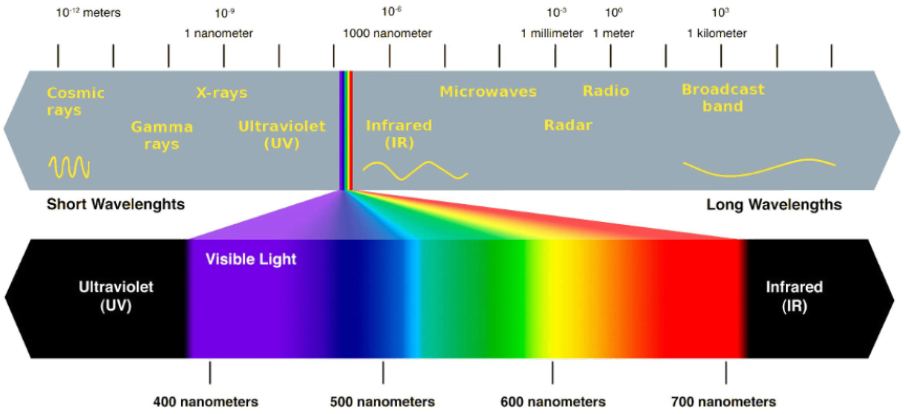

Figure 1.2: The light spectrum. The range of visible light and the associated wavelength and colours are emphasised in the bottom of the picture. (source: [3])

processed by the brain as a third colour. For example, the wavelength of yellow is typically considered as 580nm. Therefore, emitting a light at such wavelength will be perceived as yellow by the brain. But, another way to see a yellow colour is to combine a red and a green light. This is due to the fact that the cone cells in the eyes will be excited in the same way in both cases, resulting in the same mental representation. The consequence is that perceived colours is a mental representation made by the brain that is not always representative of the real emitted wavelengths.

As a consequence, we can see that the information provided by RGB images is quite limited compared to the full range of light spectrum. There are two types of limitations. The first is that RGB images are targeted at representing image as the human eyes perceive them, not as the physical wavelengths reflected by the objects, which is a more precise and measurable feature. Furthermore, the second type of limitations comes from the fact that the light spectrum is more widespread than visible light, which is a small range of the full spectrum (see figure 1.2). Exploiting other ranges of light spectrum (e.g.: infrared light, ultraviolet light, ...) can possibly give some critical information that could increase the performance of machine learning algorithms. It is thus interesting to consider multispectral images in order to see which possibilities these images could bring. This master thesis is intended at exploring some potential machine learning applications of multispectral images.

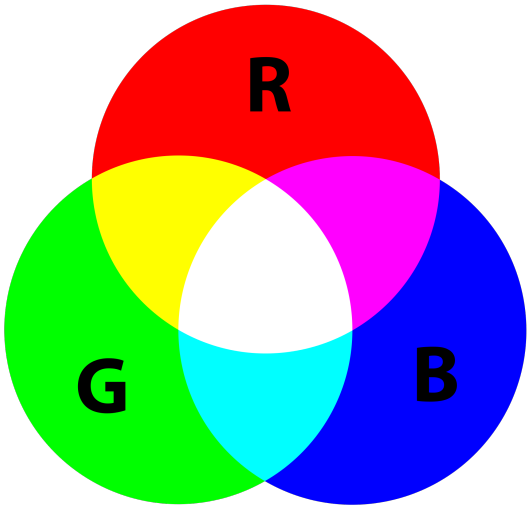

Figure 1.3: Additive colour mixing. (source: [4])

1.3 Cytomine

Cytomine is a web platform that allows to manipulate high-dimensional images, and especially medical images. It is intended as a collaborative tool that helps pathologists and teachers in research, diagnostics, and academic activities.

The Cytomine platform can be very useful in the context of this master thesis. Indeed, when developing modules for multispectral data analysis, it is very useful to have a platform accessible from a nice user interface that allows to make rapidly and easily many tasks related to multispectral data manipulation. This ergonomic platform will also allow specialists of different domains (and in particular in medicine and biology) to use the machine learning algorithms developed in the context of this master thesis without being required to have an extended knowledge in computer science. Moreover, Cytomine is a web platform and thus the data can be shared, accessed, and manipulated very easily. Furthermore, Cytomine is designed such as to be able to support high-dimensional images, which allows to manipulate images with a resolution that allows to see and process very detailed images. A picture illustrating some of the numerous tools accessible from the Cytomine platform can be seen in figure 1.4.

For this master thesis, two tools are particularly useful. The first is the annotation tool which could allow to train rapidly and easily new machine learning models thanks to the annotations made by specialists of different domains. The second tool is the ability of the Cytomine platform to manage multispectral data.

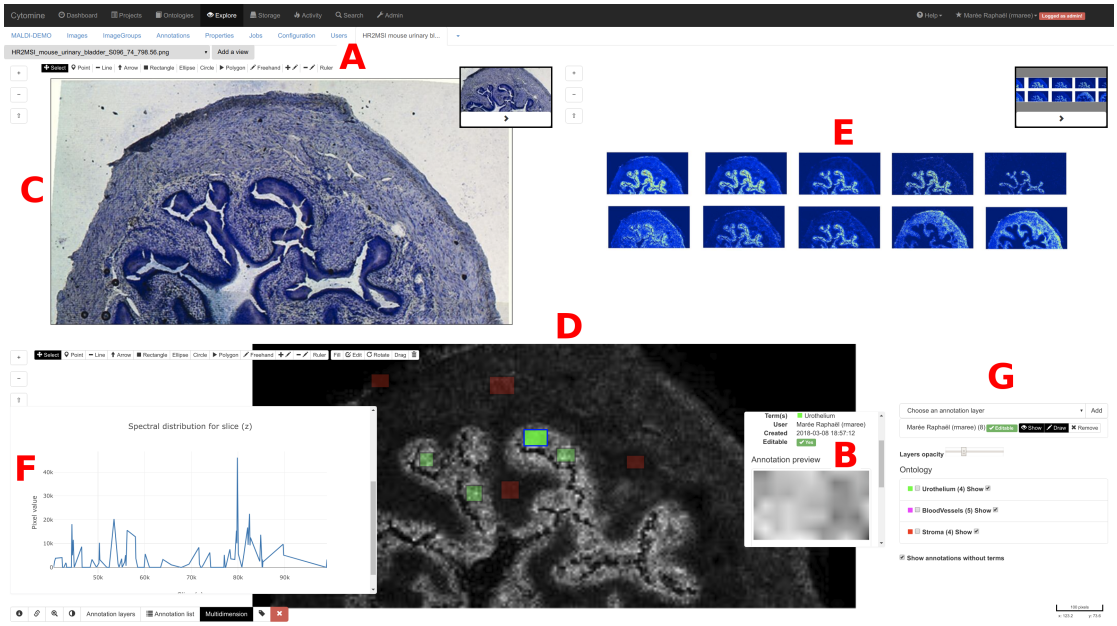

Figure 1.4: Some tools accessible from the Cytomine user-interface. A) A drawing tool allowing to select and manipulate some area of an image; B) A panel showing the information related to the annotation selected in part D of the figure; C) An optical tissue image; D) A multispectral image showing the intensities of the pixels at a specific wavelength. Some annotations are also visible; E) 10 images of the same region as image in D but with different wavelengths and with normalised jet color maps; F) a graphic showing spectral data for a selected pixel in image D; G) Some information attached to the images. (source: [5])

This ability allows to see and manipulate in a very intuitive way multispectral images, permitting either to see an image at a given wavelength or to see the spectral data at a pixel selected by the user. The new modules developed in the context of this thesis can allow to automatise some tasks normally dedicated to highly skilled specialists, allowing to make their work more efficient. These modules can be integrated in the future to the Cytomine platform to give access very easily to the classifiers developed in this master's thesis.

Chapter 2

Theoretical Part

2.1 Fundamentals of machine learning

Machine learning is a field of artificial intelligence focused at creating algorithms allowing computers to improve on a task based on experience. In machine learning, experience is acquired thanks to sample data provided to some algorithms in order for the machine to improve at a given task. Machine learning tasks can be divided in 3 categories:

- **supervised learning**: In this setting, you provide to the algorithm a dataset of samples considered as the input set. These samples consists in a vector giving information about a certain number of features. You also provide for each sample in the input set a corresponding output, which corresponds to the value associated with the sample. The goal of the machine learning algorithm is to be able to find a good predictive model for which, given an input sample, predicts well the output associated with the sample thanks to the features associated with the samples. Formally, given a set of input-output pair $(x_i, y_i), i = 1, ...N$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, the goal of supervised learning is to find a function $f : \mathcal{X} \to \mathcal{Y}$ that minimise the expectation of a chosen loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ over the joint distribution of input-output pair. In mathematical notations, we have to minimise $\mathbb{E}_{x,y}\{L(f(x), y)\}$.

- **Unsupervised learning**: In unsupervised learning, the algorithm is only provided with an input set of samples where each sample corresponds to a vector of features. The goal is to detect interesting relationship between variable in order to better understand the underlying process from which the data were generated.

- **semi-supervised learning**: In semi-supervised learning, we have an input set of samples where some samples have a corresponding output and other

samples were not provided with their corresponding output, even if in fact, this output exists, but was not determined in any way and thus not available in the input set. The idea is first to create a model from the input-output pairs that are fully provided. Then, thanks to the trained model, samples with no corresponding output will be predicted. A confidence level has to be associated with each prediction. Then, the predictions on which the model is the most confident will be added to the input set at the corresponding sample, hoping that the predicted values are correct or close to be correct. The model will then be trained again with the original input-output pairs and the new input-output pairs on which the model is most confident. It is expected than the obtained model will be better thanks to the new data given to the machine learning algorithm. Consequently, new predictions can be made on the remaining set of samples without output. And then the predictions in which the model is confident are added and the process is repeated a given amount of times to get a final model.

In this master's thesis, we will not encounter unsupervised learning. This thesis focuses only on supervised and semi-supervised learning. Supervised learning can be divided in two kind of tasks:

- **classification**: In classification tasks, the output of each sample is a class, i.e. the output of each sample is a category. The goal for the classifier model is to find a model that determines the right class for a sample according to the sample features.

- **regression**: In regression, the output for each sample is a real value. The goal of the regressor model is to find a model that approximates at best the output for a sample according to the sample features.

In this master's thesis, only classification tasks are done.

2.2 Assessing a classifier performance

When you have trained a classifier thanks to data provided to machine learning algorithm, the performance of the model must be evaluated in order to see if the algorithm models correctly the relationships between the input samples and the corresponding outputs. In order to make that, we have to create a set of input to provide to the algorithm for making predictions. The real classes of these inputs must be known but are not provided to the algorithm for making predictions. The performance is evaluated by comparing the vector of real outputs with the vector of predicted outputs. In practice, from the original set of input-output pairs, two

True class \ Predicted class	0	1
0	TN	FP
1	FN	TP

Table 2.1: Confusion matrix for a binary classification task. Class 0 is called the negative class. Class 1 is named the positive class. Meaning of acronyms: TN = true negative, TP = true positive, FN = false negative, FP = false positive

separate set are created: a training set (also named learning set) and a test set. Most of the data are put in the training set. The rest of the samples is put on the test set and are not used in any step of the learning phase. In this way, the model is tested on samples that it never sees before. In this way, we can evaluate the generalisation performance of the classifier. Indeed, The learned model can suffer of overfitting, i.e. using some relationships in the training data useful for making predictions on this same training data but not in general. This is due to the fact that the model is learning on noise in the samples provided to the learning algorithm. Concretely, the consequence of overfitting is that the model makes really good predictions on samples present in the training set but performs very badly on new samples not used for training. However, what we want is an algorithm that works well on new samples not provided during training. It is thus really important to create a test set for assessing a classifier, even if it implies to provide fewer samples for the training phase.

When we have the vector of predicted outputs and the corresponding vector of real outputs, there exists many ways to evaluate the classifier. Often, to have a good general view of the classifier performance per class, a confusion matrix is created. The table 2.1 gives a general view of a confusion matrix for a binary classification task. The two class are named 0 and 1. 0 is considered as the negative class and 1 is considered as the positive class. Each cell of the confusion matrix contains the number of times a given pair of real output and predicted class appears for the test set. Note that the confusion matrix can be easily used for more than two classes. Its is only required to make a square matrix of the dimension given by the number of classes used for the task. Thereby, each pair of real class and predicted class has a cell in the confusion matrix.

From the confusion matrix for binary tasks as given by the table 2.1, many metrics can be computed to assess the classifier performance under different aspects. Let's first define accuracy:

$$accuracy = \frac{TN + TP}{TN + FP + FN + TP}$$

Accuracy computes the proportion of samples in the test set that are well predicted. However, this metrics is not always appropriate. Indeed, consider a test set with 99% of samples being negative (class 0) and the remaining 1% being positive (class 1). If we create a classifier that always predict a sample as being a negative sample, we have 99% accuracy. But then the classifier is completely useless. Accuracy does not give any details on the performance of the algorithm at detecting correctly each class. For this reason, another metrics have been defined. Let's now define precision and recall:

$$\text{precision for the positive class} = \frac{TP}{TP + FP}$$
$$\text{recall for the positive class} = \frac{TP}{TP + FN}$$

Precision computes the proportion of samples predicted as positive that are really positive and well predicted. Recall computes the proportion of samples whose real class is positive that are correctly predicted by the classifier. Both metrics can be useful. Depending on the application, one of the two metrics can be more important. For example, When classifying web pages as useful for a user or not, we are often interested in giving useful pages to the user, but not necessarily all useful pages. In this case, precision is more important because the most important is that provided pages are really useful to the user. On the contrary, for cancer classification, we want to find all cancerous patients, even if it implies to have some false positives (i.e. detect patients as having cancer while it is not the case). In this case, recall is more important. There exists also situation for which both precision and recall are important. For these situation, the F1-score can be used. First, let's define the F-score:

$$\begin{aligned} \text{F-score} &= \frac{1}{\alpha\frac{1}{\text{precision}} + (1-\alpha)\frac{1}{\text{recall}}} \text{ with } \alpha \in [0,1] \\ &= \frac{(\beta+1) * \text{precision} * \text{recall}}{\beta * \text{precision} + \text{recall}} \text{ with } \beta \text{ defined as } \beta = \frac{1-\alpha}{\alpha} \end{aligned} \tag{2.1}$$

We can see that the F-score is the weighted harmonic mean between precision and recall. In order to get the F1-score, we have to equalise the weights of precision and recall. It implies to set $\alpha = \frac{1}{2}$, which means that $\beta = 1$. Finally, the formula for the F1-score is:

$$\text{F1-score} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

When we have a classification task where the number of different classes is greater than 2, then precision and recall can be computed in a one-vs-rest fashion. It means that we consider a given class as the positive class and the rest as the negative class. We tell that we compute a metric (precision, recall or f1-score) for a given class if we consider this class as the positive one. After having computed precision, recall or f1-score for each class, then aggregated precision, recall and f1-score can be computed. Let's note L, the set of classes and NTS_l, the number of true samples for the class $l \in L$. Then, the macro and weighted average for a metric (either precision, recall or f1-score) is:

$$\text{macro average metric} = \frac{1}{|L|} * \left(\sum_{l \in L} \text{metric for class } l \right)$$

$$\text{weighted average metric} = \frac{1}{\sum_{l \in L} NTS_l} * \left(\sum_{l \in L} NTS_l * (\text{metric for class } l) \right)$$

2.3 Tree models

2.3.1 Decision trees

This part will explain the machine learning algorithm called decision tree. A decision tree is made of 3 elements:

- interior nodes where a test is made on the training set;
- branches which are the different decision based on the test made in an interior node
- leaf node where a class is attributed and no new tests are done.

An example of such decision tree is shown in figure 2.1. In order to make a prediction given an input sample, the decision tree starts from the root node and makes the test at this node on the input sample. A decision is taken from the test, which corresponds to following one of the branch below the node. Following a brach will lead to another node, either internal or leaf node. If an internal node is reached, then repeat the process of making a test and follow the appropriate

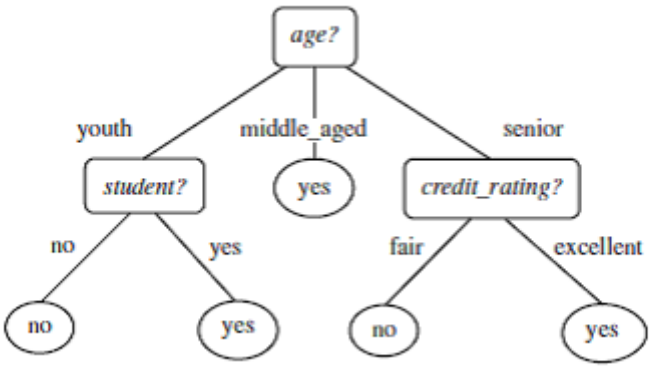

Figure 2.1: An example of a decision tree. (source: [6])

Algorithm 1: learn_dt(LS)

if *all samples from LS have the same class or if all the samples have the same value for every attribute* **then**
  create a leaf.
  Assign to that leaf the majority class of the samples reaching that leaf.
else
  Use LS to find the best splitting attribute noted A^*.
  Create a test node for that attribute.
  forall *different values a of* A^* **do**
    Build $LS_a = \{o \in LS | A^*(o) \text{ is a } \}$
    Call learn_dt(LS_a) to grow a subtree from LS_a

Figure 2.2: Decision tree induction algorithm (source: [7])

branch. When a leaf node is reached, the predicted class is the one associated with the leaf node.

The goal of training a decision tree is to find a tree structure that allows to give good predictions based on the features in the training set. In an ideal world, every possible tree structure would be tested and the one reaching the best performance would be kept. However, this is intractable most of the times given the number of possible trees. Some heuristics must be chosen. A top-down induction of the trees is chosen as heuristics. A pseudo-code explaining the strategy is given in algorithm 1. Note that LS refers to the learning set, also called training set. The notation TS is reserved for the test set.

The algorithm requires to find a way to create a split rule in a given node. In order to choose a best splitting rule, we have to define a way to assess the quality of a split. In order to make that, we first define an impurity measure of a node. From information theory, we have the notion of Shannon's entropy which gives a mathematical definition of the uncertainty. Let's define $p_j, \forall j = 1, ..., J$ the proportion of sample belonging to class j. Then, Shannon's entropy is defined as:

$$H(LS) = -\sum_{j=1}^{J} p_j \log_2 p_j$$

However, this metric can be slow to compute because of the logarithm computation. Another impurity measure , called the Gini index, is often used for its higher speed of computation. It is defined as:

$$I_{Gini} = \sum_{j=1}^{J} p_j(1 - p_j)$$

Let's define a given impurity measure by I. Then, the best split is the one that maximises the expected reduction of impurity. Mathematically, if we define LS_a the subset of samples s from LS such that $A(s) = a$ and $A(LS)$ the set of the different values of the attribute A in LS, then the expected reduction of impurity is defined by:

$$\Delta I(LS, A) = I(LS) - \sum_{a \in A(LS)} \frac{|LS_a|}{|LS|} I(LS_a)$$

The chosen split at a node is the one that maximises the expected reduction of impurity. We compute the best split at each node until the leafs contains only samples of the same class and/or the leafs contain identical samples. However, fully developing the tree can lead to overfitting. That is due to the fact that the tree at the bottom nodes might end up learning on spurious correlations between features and the output classes in the learning set. The consequence is that the tree is good at predicting on samples of the learning set, but not on new samples never seen by the classifier. In order to avoid that, pre-pruning can be used, i.e. stop growing the tree before having completely pure nodes. One common criterion for stopping the tree growing earlier is a threshold giving a minimum number of samples required for a leaf node. It means that when considering a split for a node, if at least one of the child nodes have less samples that the minimum required for a leaf node, then the node is not split and becomes instead a leaf node. The predicted class for this leaf is the majority among the classes of all samples in the learning set reaching that new leaf.

2.3.2 Random Forests

Random forest is an ensemble method combining several tree predictions. This algorithm belongs to what is called bagging method. Bagging is an averaging technique consisting in combining the output of several models to get better predictions. The idea is to build N training sets from the data and to train a classifier for each of these training sets. The different models are trained independently of each other, allowing to parallelise the process. When making a prediction, each model in the ensemble makes a prediction and these predictions are then averaged to get the final prediction. In order to create the N learning set from the data, bootstrap sampling is used. Bootstrap sampling consists in sampling with replacement from the whole training set in order to create the N learning sets to provide to the different models in the ensemble. Generally, bagging reduces a lot the variance but increases a little bit the bias because the different models created by bootstrap sampling have an effective size smaller than the size of the original training set.

Random forests is a method that combines bagging and feature selection. It means that when searching for the best split of a node, only a subset of the features are analysed. This allows to decrease significantly the computing time of the model training but also to decrease the variance.

2.3.3 Extremely randomised trees

The extremely randomised trees (also known as extra-trees) is another ensemble methods that is based on the same ideas as random forest but with a main difference in the way trees are constructed. As in random forest, a given amount of features to use to assess splits is drawn. But instead of looking for the most discriminative threshold as random forest do, extremely randomised trees draw some threshold at random for each of the selected feature. The quality of the split with each threshold drawn is assessed and the best split found is kept. This process adds more randomness than random forest. This algorithm generally reduces more the variance than random forests but the bias can increase a little bit.

2.3.4 Extension of tree models to multi-output tasks

A multi-output problem is a task where the training set input remains a set of samples feature vectors but where the corresponding output is not just a single value as in single-output classification but an array of outputs. Thus, to each input sample is associated a vector of several outputs. The goal consists in making the best machine learning algorithm for all outputs.

A simple approach for dealing with multi-output problems is to train a separate model for each output. However, we can assume that if several outputs are associated to the same input sample, they must be in some way correlated as there must be relations between the input sample and the output for a classifier to predict correctly the class associated with the input sample. Consequently, it is interesting in that setting to train only one classifier which can handle multi-output tasks and exploit correlations between the outputs to build a better classifier.

Decision trees can be extended so as to handle multi-output tasks. For that, we have to define a new impurity measure for assessing the quality of a split and taking the best split found. For multi-output problem, the quality of a split can be assessed by computing the average reduction of impurity across all outputs. Mathematically, for $k = 1, ..., K$ the different outputs of the machine learning task, the expected reduction of impurity for a learning set LS and an attribute A is:

$$\Delta I(LS, A) = \frac{1}{K} \sum_{k=1}^{K} \left(I(LS, k) - \sum_{a \in A(LS)} \left(\frac{|LS_a|}{|LS|} I(LS_a, k) \right) \right)$$

The best split is the one that maximises the expected reduction of impurity.

When creating a leaf node, we must store a vector of classes for the associated outputs rather than a single class. The classes associated with the leaf node are the majority class for each output taken separately.

2.4 Feature importance

When we have a model with good performances at predicting for a given task, we are often interested about what the model has discovered in order to correctly classify new samples. Then, the goal is to create procedure for model inspection. Model inspection is really important to give an explanation about the predictions in order to better understand the underlying process generating the data. Feature importance are a way of computing a given importance to a variable. It allows to see which variables contribute the most to the model predictions.

2.4.1 Reduction of impurity importance

Reduction of impurity is a way of computing feature importance for tree-based models (single decision tree, random forests or extremely randomised trees). It consists in computing the total reduction of impurity brought by each feature on a trained tree-model. Let's consider single decision tree. The importance of a feature A is computed as:

$$importance(A) = \sum_{\text{nodes where } A \text{ is tested}} |LS_{node}| \Delta I(LS_{node}, A)$$

For random forest or extremely randomised trees, computing feature importance is done by computing feature importance on each individual tree in the ensemble and then averaging over all trees in the ensemble model.

2.4.2 Permutation importance

Permutation importance is a way of estimating the feature importance that is model-agnostic. First, we compute the performance of a model by computing a metric on a given set. Then, for a given feature of this set, the idea is to shuffle randomly the values of that feature between samples in order to break the potential relationship between the feature values and the predictions. Then, we compute again the same metric on the same dataset. If the model performances dropped a lot, it means that the feature is important. Otherwise, it means that the feature is not important. The difference between the metric on the original set and the same metric on this set with a feature shuffled gives the permutation importance score. However, as shuffling is a random process, the shuffling and evaluation procedure on shuffled dataset is repeated several times and the performance after each shuffling procedure is evaluated. The permutation importance is computed as the difference between the score on the original input set and the average performance score on shuffled dataset for a given feature. The feature importance computation is repeated for each feature in order to see which variables are the most important to keep good prediction performances. Note that if a feature is really not important, it can occur by chance that the permutation importance computed for that feature is negative, i.e. the model has better performance when shuffling randomly that feature. Such feature permutation importance is set to 0 to show that the feature is not important. If we define m the model to assess, D the data on which to assess the model, s a score and i a feature importance, then a pseudo-code is given in algorithm 2.

Permutation importance has several advantages over reduction of impurity importance. The first is that the permutation importance can be computed on any

Algorithm 2: compute_permutation_importance(m, D)

compute the reference score s of the model m on data D
foreach *feature j (column of D)* **do**
  foreach *k in 1,...,K* **do**
    randomly shuffle column j of dataset D to generate a corrupted version of the data named $\tilde{D}_{k,j}$
    compute the score $s_{k,j}$ of model m on corrupted data $\tilde{D}_{k,j}$
  compute importance i_j for feature f_j defined as $i_j = s - \frac{1}{K}\sum_{k=1}^{K} s_{k,j}$

Figure 2.3: Permutation importance algorithm (adapted from [8])

model, not just tree-based methods. Furthermore, one problem of decrease in impurity method is that these features are computed on the training set. However, if the model is overfitting, reduction of impurity importance can give high importance to data that are irrelevant to predict on new data not used for training. Permutation importance allows to compute the importance on an independent test set not provided for training. But permutation importance can also be computed on the training set and compared to permutation importance of the test set to evaluate at which features the model is possibly overfitting.

2.5 Classes equalisation

In machine learning, one of the biggest problem is dataset class imbalance. Indeed, nearly all machine learning algorithms will be biased towards majority class(es) when fitting because of the higher proportion of samples of this (these) class(es) in the training set. Unfortunately, tree-based algorithms exhibit such a bias. Several strategies exists in order to solve the problem of data imbalance. Some of them are explained in the subsequent sections.

2.5.1 Under-sampling strategies

In this section, sub-sampling strategies will be explained. These are algorithms that try to reduce the number of samples of some or all classes. The purpose is to keep samples that brings the most information to the learning algorithm and to reject others. Each resampling algorithm has its own way to determine the most important samples to keep in the training set. When under-sampling is applied only to the majority classes, we can reduce imbalance and in some cases obtain completely balanced dataset. The ability to create completely balanced dataset is dependent upon the resampling algorithm used.

Random under sampler

The random under algorithm consists in separating the training set given the class of the samples and then to select randomly a given number of samples of each class. A balanced dataset can be obtained by selecting from each class a number of samples equal to the minority class. Consequently, all samples from the minority class is added to the resampled training set. In the rest of this thesis, random under sampler is always used to balance the dataset to the number of samples of the minority class.

Edited nearest neighbours

In edited nearest neighbours algorithm, the goal is to apply a nearest neighbours algorithm in order to exclude samples that do not agree enough with their neighbours samples. This evaluation is done for all the samples from classes to be under sampled. The number of neighbours is a parameter to choose. Two selection criteria can be used. One of them consists in determining the majority classes in the neighbours samples in order to see if the considered sample agrees with its neighbourhood. Another strategy consists in rejecting the sample if at least one of the neighbours samples have not the same class as the evaluated samples.

Near Miss

The near miss algorithm is an under sampling algorithm that uses some heuristics based on the nearest neighbours algorithms to select samples from majority classes. The minority class samples are always selected. There exists 3 versions of the near miss algorithm, each one implementing a different heuristics for selecting samples.

In near miss version 1, selected samples from the class to be under-sampled are those for which the average distance to the N closest samples of the minority class is the smallest. N is a parameter to choose.

In near miss version 2, selected samples from the class to be under-sampled are those for which the average distance to the N farthest samples of the minority class is the smallest. N is a parameter to choose.

In near miss version 3, there are 2 steps. In the first step, for each minority class sample, its M nearest neighbours of the other classes will be selected. Then, the samples selected from the classes to be under sampled are those for which the average distance to the N nearest neighbour of the minority class is the largest. M and N are parameters to choose.

Tomek's Links

In this algorithm, the goal is to remove Tomek's Links. A Tomek's Link between two samples a and b of different classes exists if for all samples c and for a distance function d, we have:

$$d(a, b) < d(a, c) \text{ and } d(a, b) < d(b, c)$$

In other words, two samples from different classes form a Tomek's link if they are the nearest neighbour of each other. A parameter to choose allows to tell the algorithm if we want to remove both samples in the Tomek's link or only the sample from the majority class. For dealing with multi class classification problem, a one-vs-rest scheme is used.

2.5.2 Over-sampling strategies

In this section, over-sampling strategies are presented. They consists in creating new data samples from minority classes in order to obtain a balanced dataset with a number of samples for each class equal to the number of samples of the majority class.

Random over sampler

In over sampling algorithm, the samples are separated according to their class. Then, a given number of samples of each class taken with replacement. Most of the times, we want to over sample the minority classes so as to get a number of samples for each class equal to the number of samples of the majority class. Consequently, some samples from minority classes are copied several times in the over sampled training set.

SMOTE

SMOTE stands for synthetic minority oversampling technique. An illustration of how this algorithm generates new samples is shown in figure 2.4. In order to create a new sample x_{new}, a sample x_i is chosen at random. Then, the k nearest neighbours of x_i will be computed. k is a parameter to choose. In figure 2.4, the 3 nearest neighbours are computed and represented by the blue circle. After that, one of the k nearest neighbours is selected at random. This neighbour sample is noted x_{zi}. Then, a new sample is created by applying the following formula:

$$x_{new} = x_i + \lambda * (x_{zi} - x_i) \text{ with } \lambda \text{ chosen randomly in } [0, 1]$$

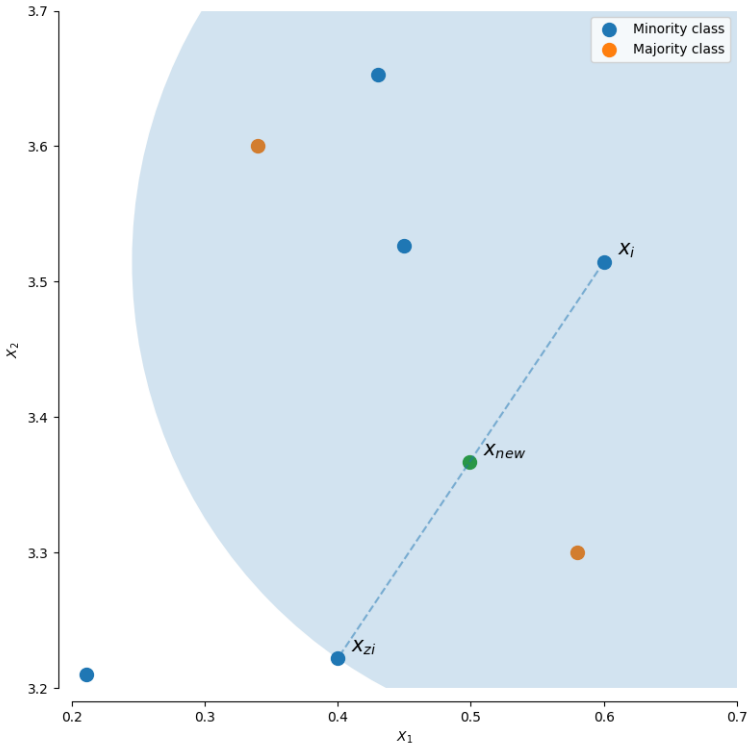

Figure 2.4: Illustration of the SMOTE algorithm (source: [9])

We can see on figure 2.4 that the new sample is generated on the line between x_i and x_{zi}. When the classification tasks have more than 2 classes, a one-vs-rest scheme is used by selecting each class to over-sample separately and apply the algorithm against the rest of the samples from other classes.

2.5.3 Combined over and under sampling strategies

In combined over- and under- sampling strategies, the goal is to over-sample the minority classes, generally with the SMOTE algorithm. Then, noisy data are cleaned with an under-sampling algorithm.

SMOTE edited nearest neighbours

This algorithm uses the regular SMOTE to create data for the minority classes. Then, an edited nearest neighbours algorithm is applied so that to remove noisy samples that can be generated by the SMOTE procedure.

SMOTE Tomek

This algorithm generates new samples for the minority classes from the SMOTE algorithm. After that, noisy data are cleaned by searching and removing the Tomek's links.

2.6 Principal component analysis

Principal component analysis (PCA) is a dimensionality reduction technique. For an input set of p features, PCA's goal is to find q new variable ($q < p$) called principal components, each of these new variables being defined as a weighted sum of the original features. If we note $X_i, i = 1, ..., p$ the original variables and $Y_j, j = 1, ..., q$ the principal components, then:

$$Y_j = w_1 X_1 + ... + w_p X_p$$

w_i is the weight of the i^{th} variable to the principal component Y_j. The principal components are uncorrelated and can be sorted by the amount of variability of the data it captures. By exploiting some correlations between the variable in the original space, PCA allows to capture the variability in the data in a more efficient way than in the original space, allowing to compress the data by reducing the number of dimension kept in the transformed variable without loosing too much information. Actually, it even allows to improve the quality of the data if they are noised because the noise in the data can affect every dimension of the dataset. But thanks to the principal component analysis, we will keep only variables with the highest amount of variability explained by the component, thus dropping the variability introduced by noise for the principal components from $q + 1$ to p. The way of computing the principal components is the following:

1. The data is centred, i.e. the mean for each variable is computed and each data point variables are subtracted by their mean.
2. The covariance matrix is computed on the centred dataset.
3. The different eigenvalues of the covariance matrix are computed.
4. For each eigenvalue, compute an eigenvector with a norm equal to 1. These eigenvectors are the principal components. These eigenvectors point to the major direction of variation in the data.

The eigenvalue associated with a principal component is equal to the amount of variability in the data explained by the principal component. Also, the principal components are uncorrelated between each other and means that no one

have redundant information from another principal component. We can thus sort principal components by their eigenvalues from the highest to the lowest eigenvalue. Then, the variability explained by the q first principal components out of p principal components can be computed:

$$\text{Variability ratio explained by the } q \text{ first principal components} = \frac{\sum_{k=1}^{q} \lambda_k}{\sum_{k=1}^{p} \lambda_k}$$

where λ_i represents the eigenvalue of the i^{th} principal component when sorting eigenvalues in decreasing order.

2.7 Savitzky-Golay filter

Savitzky-Golay filter is a digital filter used for smoothing digital data. It allows to eliminate some noisy fluctuations in the data while keeping the overall tendency of the data samples. A animated gif showing the process of Savitzky-Golay filter is available at `https://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter#/media/File:Lissage_sg3_anim.gif`.

Given a sample of data equally spaced according to the independent variable (often represented by the abscissa axis), let's give an index to each sample data from 0 to the number of data points - 1 starting from the minimum to the maximum value of the independent variable. Then the goal of Savitzky-Golay filter is for each sample data at position index i to fit a polynomial of a chosen order with the points from index $i - k$ to $i + k$, where k is a parameter to choose defining the window size, the windows size being equal to $2 * k + 1$. Then, we replace the data point at position i by the value found at that position by the fitted polynomial. Thus, for each data point in a sample, a new polynomial is fitted and the central data point is replaced by the fitted polynomial value at that position. The polynomial is fitted to the data in the window by a least square fit. It means that, for $f(i)$, the data sample at position i and $p(i)$ the fitted polynomial at position i, the goal is to minimise the total square error noted as:

$$\sum_{x=i-k}^{i+k} (p(x) - f(x))^2$$

Note that if the order of the polynomial chosen is greater or equal to $2k$, then we can find a polynomial that passes exactly to each data point in the window. That is generally not what we want as we want to smooth the signal, which can be obtained by taking a polynomial order strictly less than $2k$. Also, note than

for boundary data points at the $k-1$ first index and $k-1$ last index, the window cannot be completely defined, i.e. it will lack some data at the left of first data point or a the right of last data point. In that case, the Savitzky-Golay filter does not fit a polynomial but assigns a constant value chosen by the user. This choice is generally the value 0.

Chapter 3

Histology Classification

3.1 Goals

When performing a diagnostic, many times this diagnostic is based on histological inspection of some tissues. However, there may exist variations in the diagnostic made by different specialists. This diagnostic is highly based on their past experiences. There might exist samples for which the diagnostic is unclear and the opinions of pathologists may vary. These differences may exist when these specialists must decide between different histology classes in different zones of tissues. However, many subsequent processes and decisions are based on the histology classification these pathologists do. Consequently, it would be very interesting to have a method that automatises the process of classifying tissues in different histology classes. This procedure would not only decrease the pathologists workload, but also help them in making better diagnostics, which can be critical for example when these pathologists must decide if someone has a cancer or not.

In this part of the thesis, the goal will be to classify breast tissue areas in 4 different histological classes: epithelium, stroma, blood and necrosis. This classification will be made thanks to multispectral data used to train machine learning algorithms.

3.2 Multispectral Data Acquisition Technologies

Data used in this part can be freely downloaded from [10] and are linked with the paper [2].

When collecting multispectral data, different microscopy technologies exist. Unfortunately, none of them are perfect. Most of the times, there is a tradeoff

between accuracy of the data acquired and acquisition time. The state-of-the-art technology in infrared multispectral chemical imaging in term of accuracy in the data acquired is Fourier transform infrared microscopy utilising a focal plane array detector. However, this technology is unusable since a typical tissue microarray takes days to be acquired. This duration is too long for any use in clinics diagnostics. Moreover, such a long time makes it difficult to get data from a high number of patients. For machine learning algorithms, it is absolutely required to have variety in the data for developing accurate predictors, and variety implies in this application to have data from many different patients.

Another technology called quantum cascade laser have a higher throughput than Fourier transform infrared technology. According to [11], with quantum cascade laser microscope, it is possible to acquire an infrared chemical image of an entire tissue microarray consisting of 19 millions pixels in only 9 minutes. However, the obtained spectra from quantum cascade laser contain more noise than with the other type of microscope. Nevertheless, there exists techniques for mitigating the noise in the data.

3.3 Data Description

Serial sections of formalin fixed paraffin embedded breast tissue microarray cores were acquired from US Biomax company. In total, the tissue microarray comprises 207 1mm tissue cores, each from a different patient. 192 cores are malignant and the 15 remaining cores are non-malignant. A 5μm section was dewaxed and stained thanks to hematoxylin and eosin so that it can be processed on a standard histological slide. An adjacent section was floated onto a BaF_2 slide and was not dewaxed. Retaining the samples in wax allows to avoid alteration in the tissues.

Multispectral infrared spectra from these waxed tissues were acquired thanks to a Spero quantum cascade laser microscope. The spectral range acquired is from $912cm^{-1}$ to $1800cm^{-1}$ with a step of $4cm^{-1}$. It means there is a total of 223 frequencies acquired. For each core in the tissue microarray, a $313 \times 313 \times 223$ datacube is extracted and stored to a separate file for each core. It means that each multispectral image has a width and a height of 313 and each pixel in the image comprises an array of length 223 storing the absorbances for the different frequencies acquired. The resulting files are in matlab format (.mat extension).

In total, 74 breast tissue cores out of the 207 have been reviewed by specialists to identify several zones corresponding to one of the four histological classes. Spectra from pixels corresponding to one histological class are extracted to create

a database. Then, these data are split in two parts to create a learning and a test set. In order to test the models in real conditions, the separation in learning and test set is made at the level of cores. Thanks to this strategy, spectra found in the test set are from different patients than those present in the learning set. 80% of the cores are given to the learning set, the remaining 20% go the test set. It means that out of the 74 cores, 59 cores are given to the learning set and 15 to the test set. The table 3.1 gives the details of the distribution of the spectra per histological class and for each dataset.

Finally, the data acquired consists of:

- **core files**: These files contains the spectra of each core, with one file per core. The files are in matlab (.mat) format. Even if this format is primarily intended for use in matlab software, it is possible to extract the content of the files in numpy arrays so that these data can be used inside python programs. The module for loading matlab files into numpy arrays is implemented in the scipy library. Each matlab file content is transferred into 2 numpy arrays. One numpy array contains the spectra in a three dimensional matrix where the two first dimensions represent the abscissa and ordinate axes and the third dimension corresponds to the spectra of each pixel located at a specific position represented by the abscissa and ordinate position. Each array is of dimension $313 \times 313 \times 223$ where the abscissa and ordinate are 313 pixels long and the number of spectra is equal to 223. The second numpy array is used to store the wavenumbers of each spectrum element in the same order as the spectra in the first numpy array. As this array of wavenumbers is always the same for each core, the wavenumbers can be loaded only once and used for every operation on all the cores. The size of this array is 223, as the number of spectra wavenumbers acquired for each pixel.

- **overlay files**: These files are microscopy images in png format of the same cores as in the the core files. Moreover, the overlay files were annotated by specialists who created regions corresponding to an identified histology class among epithelium, stroma, blood and necrosis. The class of these regions drawn by pathologists and their corresponding pixels are given by specific colours that are distinct from the colours present in the microscopy images. So, green and red pixels are used to indicate epithelium pixels. The difference between the two is that the green pixel represent non-malignant pixels while red pixels are considered as malignant by experts. Magenta pixels and orange pixels represent stroma pixels. Again, the difference between the two colors is the fact that magenta are non malignant stroma pixels while orange are malignant pixels. Light green pixels represent blood pixels. Finally, blue

	learning set	test set
epithelium	149856	28001
stroma	88412	22899
blood	1766	2665
necrosis	25617	2318
total	**265651**	**55883**

Table 3.1: Number of spectra for each histological class in each dataset before applying the quality test

pixels are used for representing necrosis. The overlay images are of same width and height (i.e. 313×313) than the core files in order to allow making the correspondence between the spectra and their corresponding histological class. One flaw of such region annotations is that the original overlay pixels of the core regions being annotated are hidden because of the necessity to assign a specific colour instead of the original pixel colour. Consequently, the dataset acquired provides no way for retrieving the original overlay (microscopy) files. An example of overlay image is given in figure 3.1.

- **high resolution Hematoxylin and eosin stained tissue micro-array**: The dataset also provides an high-dimensional image of the cores stained with hematoxylin and eosin. A resized version of the original image is available in figure 3.2 (the original image has a shape of 23120×27987). All cores are given an identifier according to the row and the column in which the cores are located in the tissue micro-array. Each row is given a distinct number between 1 and 16, from bottom to top in increasing order. Columns are given each a distinct letter in alphabetical order from A to M and from left to right in increasing order of the alphabetical letters. These unique identifiers used for the cores processed with hematoxylin and eosin are also used for identifying the corresponding overlay files and core files. Indeed, naming conventions for overlay filenames and core filenames are exactly the same.

3.4 Data Processing

Before providing the multispectral data to machine learning algorithms, several preprocessing operations can be done. As suggested by [2], the preprocessing pipeline used is shown in figure 3.3. Subsequent sections describe the utility of each preprocessing operation.

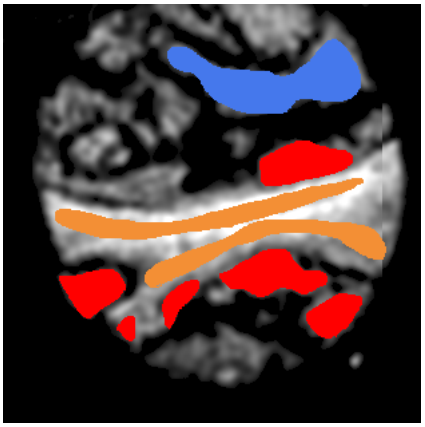

Figure 3.1: Example of overlay image. Red regions represent epithelium (malignant), orange regions represent stroma (malignant) and blue region is used to represent necrosis.

	learning set	test set
epithelium	145386	26224
stroma	88205	22755
blood	1766	2665
necrosis	25382	2318
total	**260739**	**53962**

Table 3.2: Number of spectra for each histological class in each dataset after applying the quality test

3.4.1 Quality Test

A quality test is performed in order to remove spectra corresponding to areas with little or no tissues. These spectra are thus rejected and are used neither for training the machine learning model nor for testing the model because they are considered as irrelevant.

This quality test is performed by comparing the height of the amide I band at wave number 1655 with the wave number 1760. Spectra whose absorbance difference between wave number 1655 and 1760 is between 0.1 and 2 pass the quality test. Otherwise, they fail the quality test. The distribution of the spectra per histological class and per dataset after the quality test is shown in table 3.2.

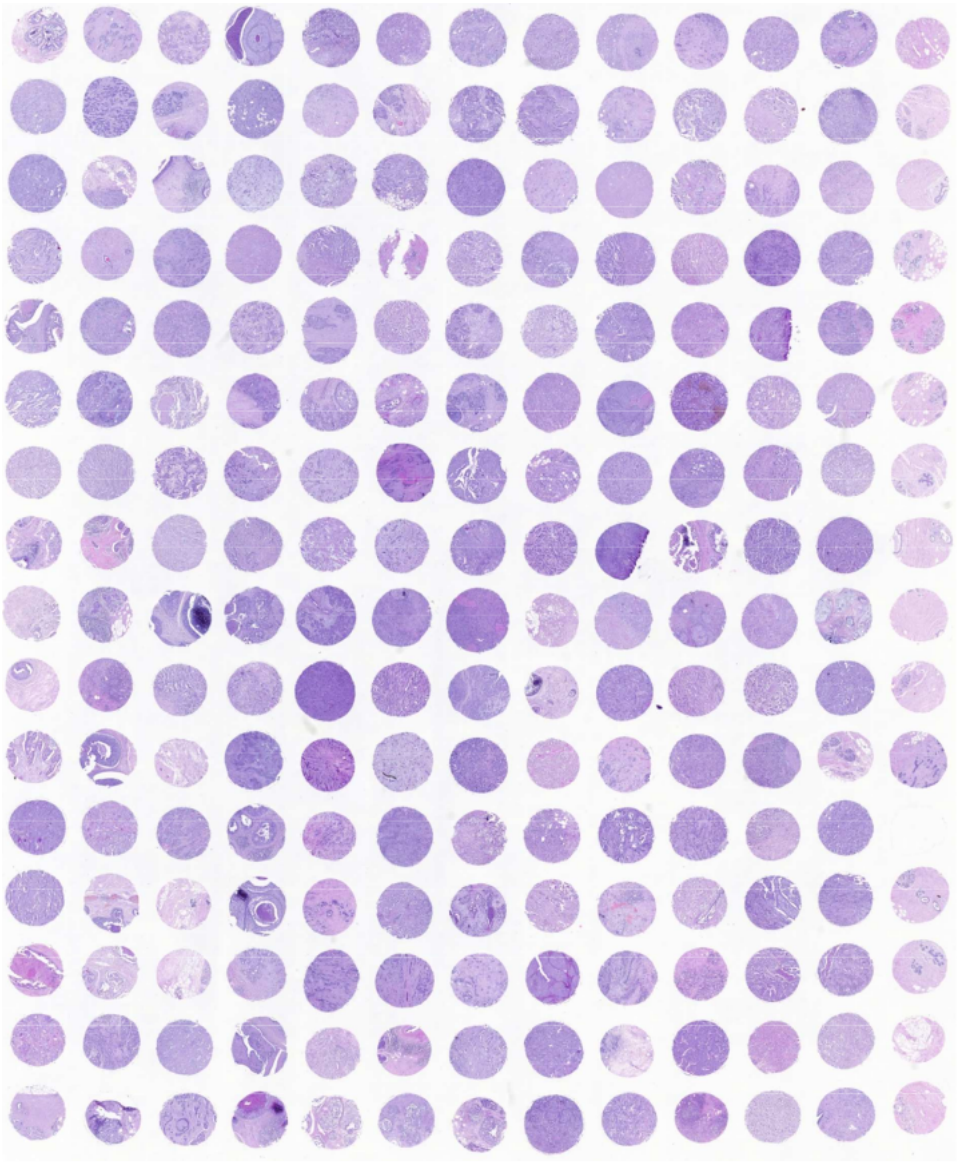

Figure 3.2: Hematoxylin and eosin stained tissue micro-array. The original image has a shape of 23120 × 27987.

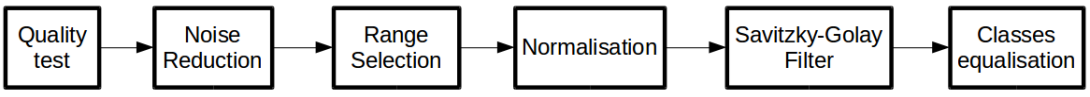

Figure 3.3: Preprocessing pipeline

3.4.2 Noise Reduction

Noise reduction is performed in order to reduce noise in the data. A given amount of first principal component are kept and the rest is rejected. Note that the input dataset is transformed back to its original space and thus, the principal components computations are used only for reducing noise, not to compress the data.

3.4.3 Range Selection

This pre-processing step allows to make a priori feature selection when they are motivations for removing some data. In the histology classification, range selection can be done to remove the absorbance regions of wax between 1350 cm^{-1} and 1490 cm^{-1}. Indeed, wax was used to prepare the core before the spectra were acquired with quantum cascade laser technology. Consequently, spectra in the wave number range of wax are contaminated.

3.4.4 Normalisation

An euclidean norm is computed spectra-wise in order to correct for different thicknesses in the cores. In this way, some human imprecision in the preparation of the cores is reduced.

3.4.5 Savitzky-Golay filter

A Savitzky-Golay filter is applied in order to smooth the signal represented by the spectra. When smoothing the spectra, we can also compute the derivative at each data point of the spectra.

3.4.6 Classes equalisation

Classes equalisation strategies can be used in order to balance the dataset. As you can see in table 3.2, the training set is highly imbalanced. So, applying resampling strategies can be useful to improve the accuracy of the model and/or improve the computation time. Note that the test set is also highly imbalanced but resampling algorithms are never applied to the test set as the test set is only used for evaluating the trained models.

3.4.7 Extension to subwindows

In order to give more information to the training set, it is possible to give not only a pixel spectra with its corresponding class, but also all neighbouring pixels

spectra in order to let the algorithm to find the potential relationships between a pixel class and the neighbourhood of this pixel. For each pixel in the input set, we extract a subwindow of pixel spectra to add to the sample features. Two settings exists for using subwindows: The first consists in keeping a single output classification task where the input are augmented with the neighbouring spectra and the output is the class of the center pixel of the subwindow. If a pixel is near to the border and the corresponding subwindow goes beyond the limits of the image, then out of image pixels are considered as black pixels with a spectra composed of 0 values for each wave number. The second setting is to consider a multi-output task where input are augmented with the neighbouring spectra but their corresponding outputs are also stored in the output set. In order to make a prediction on a given pixel class, all subwindows containing the pixel are extracted and predictions is made by the classifier for all of these subwindows. Then, the predictions of the considered pixel are gathered from each subwindow predictions in order for the classifier to make a final decision on the class of the pixel. There are two ways of gathering the subwindows predictions for the considered pixel. The first consists in predicting the class of the pixel in each subwindow and then take the majority class among the predictions. The second way consists in getting predictions probabilities for the considered pixel in each subwindow containing the pixel and then to sum these probabilities in order to find the most probable class, which is the class assigned to the considered pixel. In the multi-output setting for the training set, if a subwindow associated with a pixel spectra contains unlabelled pixels, then the original pixel is removed from the learning set and its corresponding subwindow is not added to the training set. Indeed, if we keep subwindows with unlabelled pixels, then the training set will be unable to learn from these subwindows.

The pipeline for the subwindows setting is shown in figure 3.4. We can see that subwindows are created late in the pipeline, after the Savitzky-Golay filter. After the quality test, subwindows are determined but their corresponding pixel spectra are only added to the original training set without merging spectra in input samples to augment the number of feature for each sample. Indeed, further pre-processing steps have sense only when applied to input whose samples contain only one pixel spectrum.

3.5 Experiments

Many different tests have been done in order to test various combinations of hyper-parameters. The following subsections explain the different tests and discuss the results.

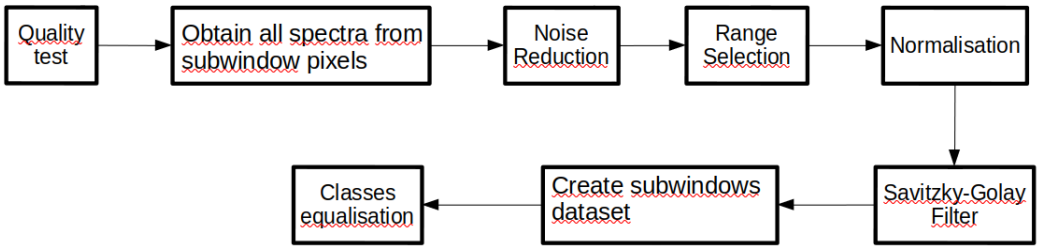

Figure 3.4: Preprocessing pipeline when creating subwindows

3.5.1 Starting Test

The paper [2] suggested some values as a starting point which are tried in this subsection. A quality test is done on each pixel spectrum as explained in section 3.4.1. The noise reduction by principal component analysis uses 40 principal components. The range selection as explained in section 3.4.3 is performed. The normalisation operation as discussed in section 3.4.4 is also done. Then, a Savitzky-Golay filter with a window size of 9 and a polynomial order of 4. The first derivative is computed. Then, a random forest classifer is computed with 500 trees in the forest. The number of features considered at each split is 15. To prevent overfitting of the classifier, the minimum number of samples necessary to create a split is 10. The scikit-learn implementation of random forest is used.

The confusion matrix in table 3.3 shows the performance of the classifier on the test set. Globally, the performance are good. We can however see that the random forest classifier has more difficulty to classify stroma pixels. We can see that misclassified stroma pixels are often confounded with epithelium pixels. The converse (epithelium pixels being classified as stroma pixels) also occurs but the phenomenon is less pronounced. We can see that when a pixel is misclassified, most of the time the pixel is classified as epithelium. This bias towards epithelium might indicate that there are more variations in epithelium spectra, making it difficult to find simple rules to discriminate between the epithelium class and the other classes. Finally, the global accuracy is 89.78% and the weighted f1-score is 89.73%.

3.5.2 Principal Components Analysis Tests

In this part, the purpose is to apply the same protocol than with the starting test, except for the principal component based noise reduction. Many classifiers are fitted by varying the number of principal components hyper-parameter. The number of principal components tested are from 5 to 160 included with a step of

Predicted class / True class	epithelium	stroma	blood	necrosis
epithelium	94.49	4.92	0.48	0.11
stroma	14.00	84.05	1.30	0.65
blood	8.67	0.00	91.33	0.00
necrosis	6.99	2.11	0.00	90.90

Table 3.3: Confusion matrix for the starting test on the test set. The confusion matrix is normalised according to true classes. Results are expressed in percent.

5 principal components.

A plot showing several metrics is available in figure 3.5. We can observe that when the number of principal components is low, the f1-score is not good. But when increasing a little bit the number of principal components, the 3 metrics increase rapidly. The best f1-score is reached at 25 number of principal components. When adding more principal components, the 3 metrics decreases a little bit but the diminution in performance is really not important. From the figure, it seems that the number of principal components parameter tuning is not crucial for the performance of the algorithm. Interestingly, we can notice that the performance of the classifier remains stable even when the number of principal components is high. Indeed, the performance when the number of principal components kept is near 160, we are close to keeping all variance in the data since the number of features in the dataset provided to the random forest is 167 (the original spectra have a dimension of 223 but the range selection pre-processing operation keeps only 167 wavenumbers). Consequently, one might ask if it is useful to keep the principal components noise reduction operation or if the classifier performance remains good when removing this operation. This case will be explored later in this thesis.

3.5.3 Savitzky-Golay Filter Tests

In this section, the purpose is to apply the same protocol as in the starting tests, except for the Savitzky-Golay filter and the number of principal components for noise reduction. The different tests make variations in 3 hyper-parameters: the windows length (also called number of smoothing points), the order of the polynomial, and the order of the derivative. For the window length, the values from 3 to 19 included are tested, with a step of 2. For the polynomial order, the values from 0 to 5 are experimented, with a step of 1. For the derivative order, the values from 0 to 5 included are tested, with a step of 1. All suitable combinations of the 3 hyper-parameters are tested. The number of principal components chosen is 35. The value 35 have been chosen for its good macro average and weighted average

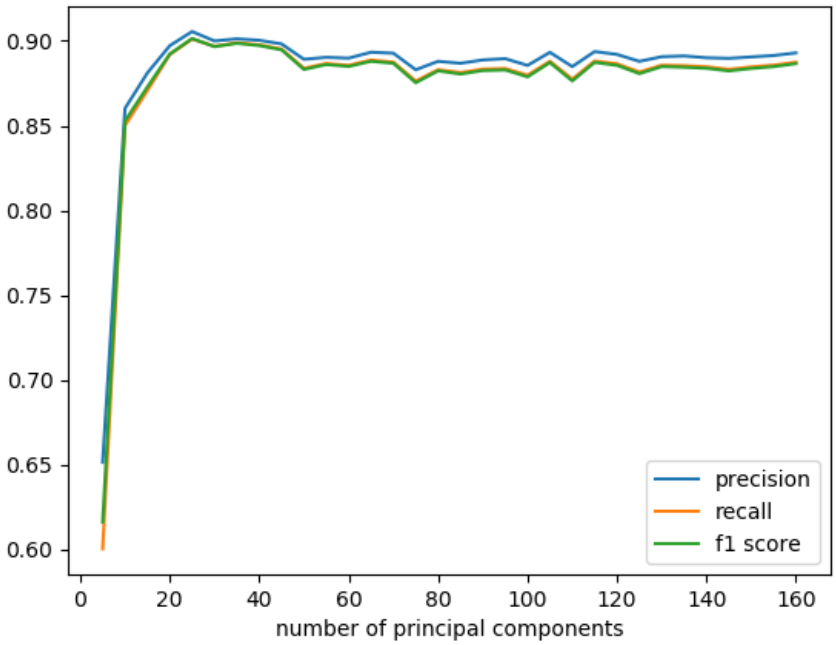

Figure 3.5: weighted average precision, recall and f1-score when varying the number of principal components

f1-score.

Results of the experiments can be found in figure 3.6. We can observe that the derivative order is crucial for creating a good random forest classifier. The first and the second derivative are clearly the best parameters, and the second derivative is also better than the first derivative. Higher order derivatives give from middle to very bad performances. We can also notice that for the derivative order 0, 1, and 2, the window length and the polynomial order has little influence on the performance. The performance difference is sharper for the derivative order 3, 4, and 5.

The positive influence of the derivative order 1 and 2 indicates that the spectra locality is an important concept for creating good classifier. Indeed, for determining derivatives, we must have several points defining the function to derive. Consequently, the derivative in one point contains in fact information about the neighbour points by indicating locally in that point the monotonic properties of the function for the first derivative and the curvature of the function for the second derivative. Computing the derivatives forces the random forest algorithm, when selecting at random a given amount of wavenumbers, to consider also neighbouring wavenumbers to determine which feature decreases the most the node impurity. Considering that, one might ask if exploiting spatial locality of the pixels is useful to improve performance of machine learning algorithms. This question will be discussed later in this thesis.

Another interesting plot showing the performance of the classifier when fixing the derivative hyper-parameter to 2 is available in figure 3.7. We can observe that the ideal hyper-parameter value combination for the number of smoothing points and the polynomial order is 13 smoothing points and a polynomial order of 4. However, the performance difference between each combination is not huge.

3.5.4 Random Forest Tests

In this part, the purpose is to fine-tune 2 random forest hyper-parameters: the maximum number of features to consider at each node split and the minimum number of samples required for a leaf node. For the maximum number of features to check parameter, the values tested go from 5 to 50 with a step of 5. A test with all wavenumbers considered at each node split is also done. For the minimum number of samples in a leaf node, the value 1 (i.e. fully developed trees) is tested. Then, the values from 5 to 50 with a step of 5 is tested. The number of trees in the forest remains fixed and is equal to 500. The number of principal component used for noise reduction is 35. For the Savitzky-Golay filter, the parameters remains

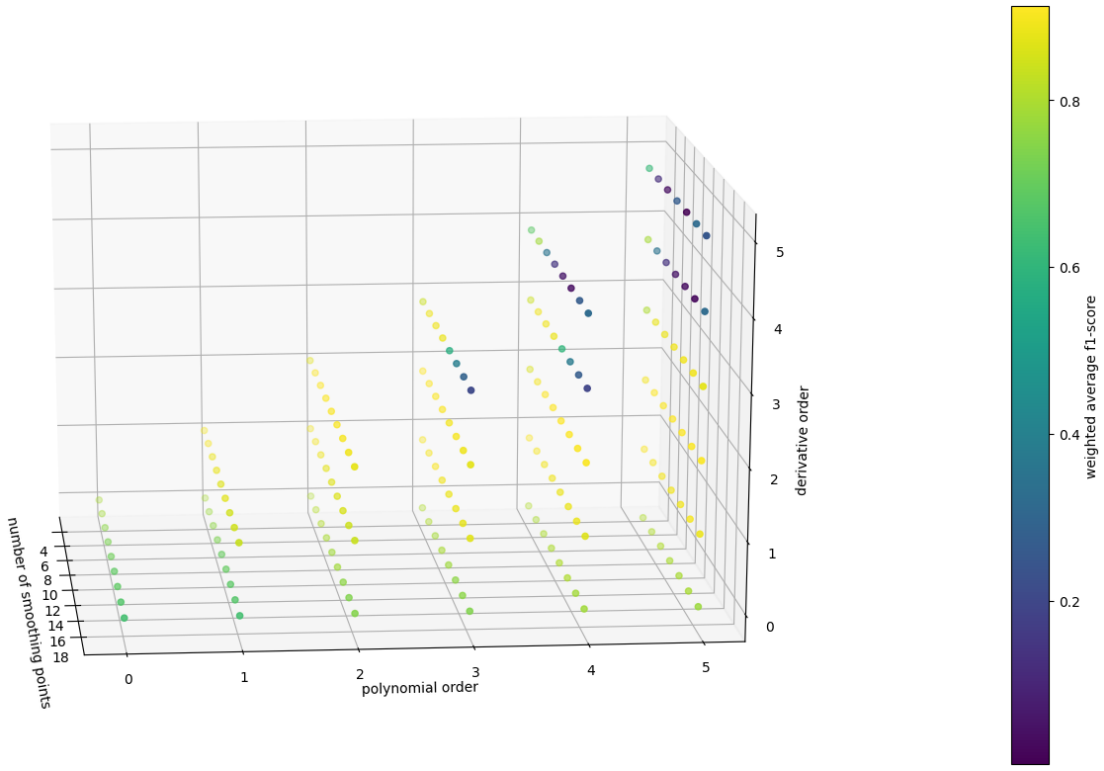

Figure 3.6: Weighted average f1-score when varying the window length, the order of the polynomial and the order of the derivative in the Savitzky-Golay filter

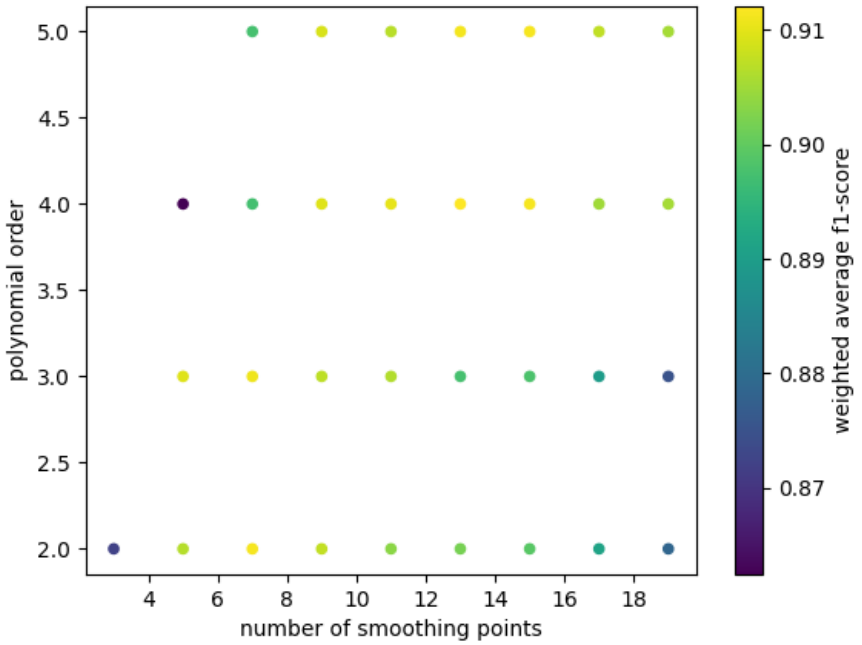

Figure 3.7: Weighted average f1-score when varying the window length and the order of the polynomial in the Savitzky-Golay filter. The derivative order is fixed to 2.

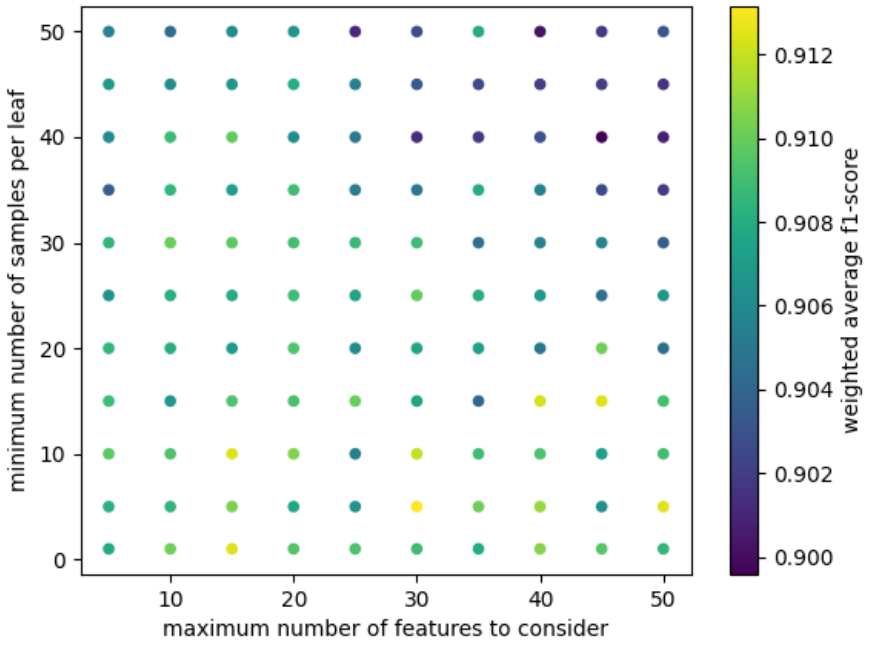

Figure 3.8: Weighted average f1-score when varying the maximum number of features to consider at each node split and the minimum number of samples in a leaf node.

fixed across all tests. The window length is set to 13, the polynomial order is set to 4 and the derivative order is set to 2, as suggested by the Savitzky-Golay filter tests.

Resulting plots can be found in figures 3.8 and 3.9. We can notice that the minimum number of samples for a leaf node seems more important than the maximum number of features to consider at each node split. Interestingly, we can observe that low values for the minimum number of samples for a leaf are the best. It means that nearly fully developed trees gives best results. However, globally, the performance differences between all tests are not huge.

When we consider all features for determining the best split, we can observe the same behaviour for the minimum number of samples for a leaf node than when limiting the number of features to consider at a split. The best value for the minimum number of samples to have in a leaf node is 5.

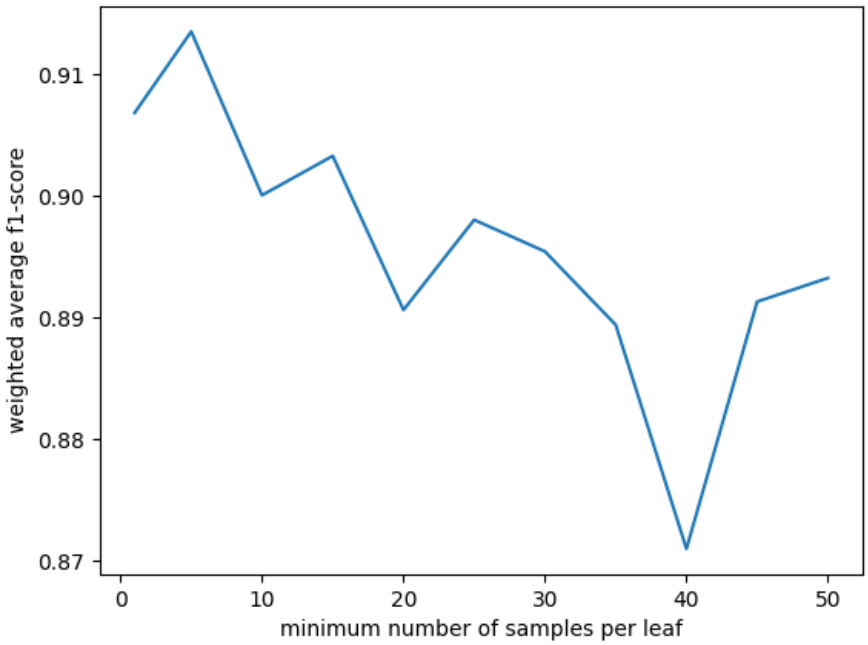

Figure 3.9: Weighted average f1-score when varying the minimum number of samples in a leaf node. All features are considered at each node split.

3.5.5 Pre-processing operations tests

In this section, the purpose is to test various combinations in the pre-processing operations that are made before training the model or predicting with it. Some pre-processing operations are not made in each test except one in order to see the influence of each pre-processing tests on the results. For the principal component-based noise reduction, when it is done, the number of principal component is set to 35. For the interval selection, when it is done, two intervals are selected: between 1000 and 1350 cm^{-1} and between 1490 and 1800 cm^{-1}. For the Savitzky-Golay filter, when it is done, the window length is set to 13, the polynomial order is set to 4 and the second derivative is computed. A random under sampler classes equaliser is applied. The classifier used is a random forest with 500 trees, 15 features inspected when splitting a node and with a minimum of 10 samples per leaf.

Results are shown in table 3.4. We can see that the most important operation is the Savitzky-Golay filter. Surprisingly, other operations seem not really necessary. The best f1-score is obtained when applying only the Savitzky-golay filter with 92.23%. Adding any other pre-processing operation reduces a little bit the performance of the classifier. However, we can also notice that the worst performance is obtained when using no pre-processing operations (72.47%). Applying any other pre-processing operation than Savitzky-Golay filter (alone or not) improves the classifier performance compared to the performance when using no pre-processing operations. When taking only one pre-processing operation per test other than Savitzky-Golay filter, we can see that the normalisation operation has more positive influence than the range selection, but this interval selection has better influence than the principal component-based noise reduction. For the principal component-based noise reduction, we have to notice that the Savitzky-Golay filter also reduces the noise in the data, making the noise reduction by principal component analysis redundant. For the interval selection, in light of the results, one interesting question is to know the importance of the features removed by the range selection because this interval is contaminated by the wax used in the cores preparation. Two hypotheses are possible. The first is that the random forest (almost) never selects the features located in the interval contaminated by wax and consequently, these features are not important. The other hypothesis is that the contamination is low or almost constant for all spectra in the interval but the spectra values in this range do not bring much additional information in order to discriminate between the 4 different histological classes. For the normalisation, it has to be noted that the dataset used consists of only one tissue microarray. The impact of the normalisation might become higher if the dataset consists of several tissue microarrays since the purpose of the normalisation is to correct for different thicknesses of tissues.

Noise Reduction	Range Selection	Normalisation	Savitzky-Golay Filter	Weighted average
✓	✓	✓	✓	90.83
✓	✓	✓	✗	79.88
✓	✓	✗	✓	91.33
✓	✓	✗	✗	75.71
✓	✗	✓	✓	91.41
✓	✗	✓	✗	77.81
✓	✗	✗	✓	90.98
✓	✗	✗	✗	73.88
✗	✓	✓	✓	89.58
✗	✓	✓	✗	80.69
✗	✓	✗	✓	91.34
✗	✓	✗	✗	75.54
✗	✗	✓	✓	91.58
✗	✗	✓	✗	78.69
✗	✗	✗	✓	92.23
✗	✗	✗	✗	72.47

Table 3.4: Weighted average f1-scores of the classifiers for the different combinations of pre-processing operations. Results are expressed in percent.

3.5.6 Random forest tests when only applying Savitzky-Golay filter as pre-processing

In this section, the only pre-processing operation done is the Savitzky-Golay filter. The window length is set to 13 and the polynomial order is set to 4. The second derivative is computed. After that, classes equalisation is done by random under sampling. The purpose is to apply a random forest classifier with 500 trees by varying two hyper-parameters: the number of features to consider at each node split and the minimum number of samples at each leaf of the trees.

Results are available in figures 3.10 and 3.11. We can observe that the minimum number of samples per leaf hyper-parameter has more influence on the classifier performance than the maximum number of features to evaluate at each node split. Interestingly, we can see that, globally, the lower the minimum number of samples per leaf is, the better the weighted f1-score is. The best performance is obtained when setting the minimum number of samples per leaf to 1 and when considering only 15 features maximum at each node split. The best result is consequently obtained when fully developing the trees in the forest.

3.5.7 Extra trees tests when only applying Savitzky-Golay filter as pre-processing

In this section, the goal and setting are the same as in subsection 3.5.6 except that the machine learning algorithm used is the extra trees with 500 trees in the ensemble.

Results are visible in figures 3.12 and 3.13. We can see that both hyper-parameters (maximum number of features to consider at a node split and minimum number of samples in a leaf node) are important for the performance of the model. We have to consider more maximum number of features to consider at a node split in extra trees in order to get similar performance to random forests. This is due to the fact that the splitting value considered for a feature in the extra trees algorithm is chosen randomly. As in the random forest case, it seems better to consider a low value for the minimum number of samples per leaf. However, neither the random forest nor the extra trees seems to be better than the other.

3.5.8 Random forest with subwindow inputs

In this part, the goal is to use make a random forest classification model where the inputs for each pixel contain not only the spectrum of the corresponding pixel, but also the spectra of the surroudning pixels. The goal is to define a subwindow

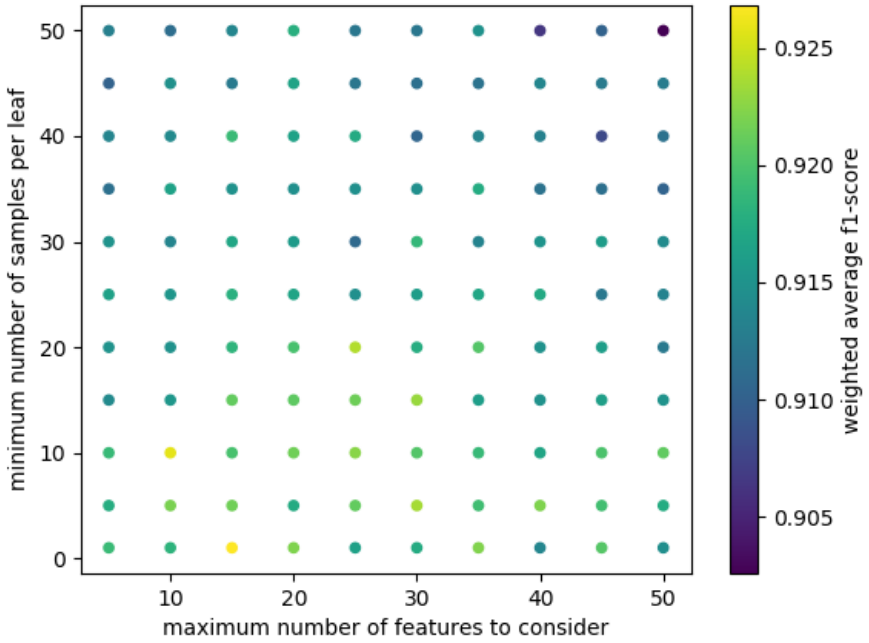

Figure 3.10: Weighted average f1-score for random forest classifiers when varying the maximum number of features to consider at each node split and the minimum number of samples in a leaf node.

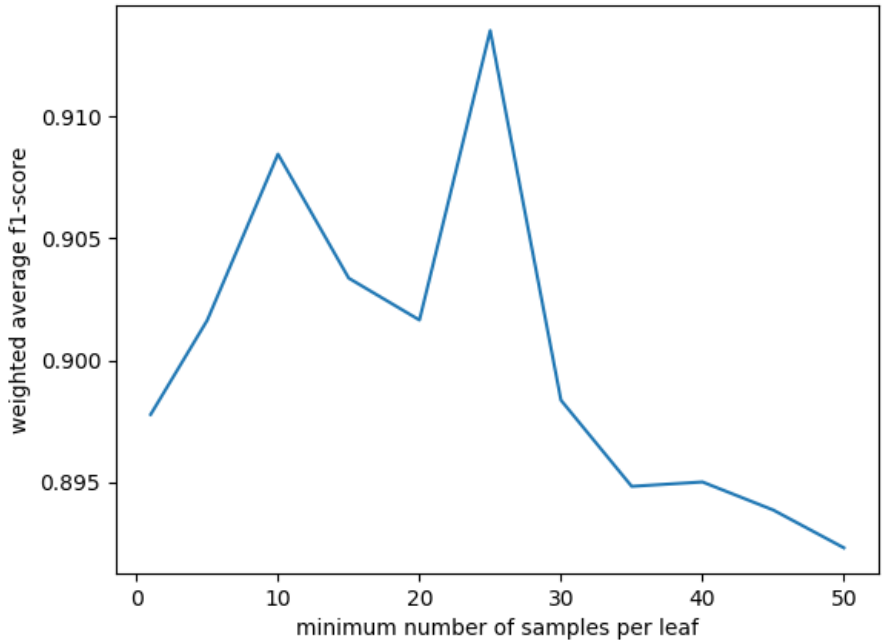

Figure 3.11: Weighted average f1-score for random forest classifiers when varying the minimum number of samples in a leaf node. All features are considered at each node split.

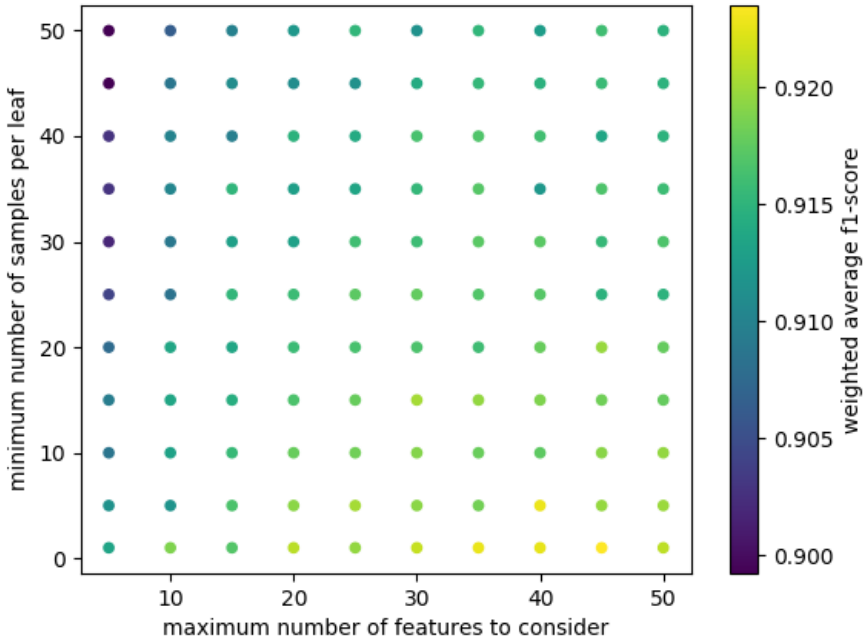

Figure 3.12: Weighted average f1-score for extra trees classifiers when varying the maximum number of features to consider at each node split and the minimum number of samples in a leaf node.

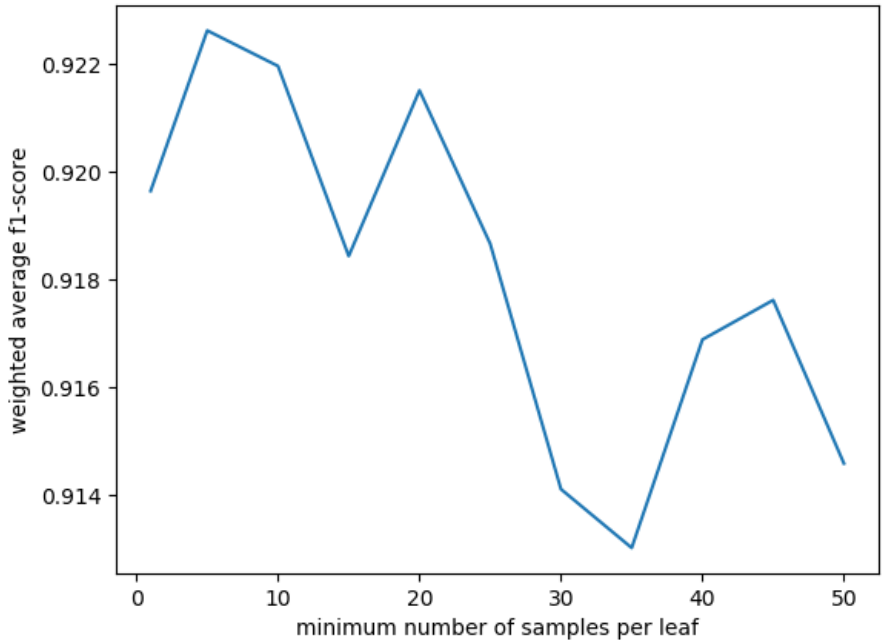

Figure 3.13: Weighted average f1-score for extra trees classifiers when varying the minimum number of samples in a leaf node. All features are considered at each node split.

whose center is the considered pixel. The width and the height for the subwindows are parameters to optimise. Then, the spectra of the entire subwindow are extracted and given as input for the considered central pixel. The spectra are put in order from left to right and from top to bottom of the subwindow. For each spectra in the original training and test set, the corresponding subwindow is computed and the corresponding spectra are extracted and integrated to the learning or test set. The output of the original training set remains the class of the pixel considered, in other words the class associated to each subwindow input is the class of the subwindow center pixel.

The random forest is trained with 500 trees. No principal component noise reduction, no feature selection and no normalisation is done. A Savitzky-Golay filter is used with a window size of 13 and a polynomial order of 4. The second derivative is computed. A random under sampler is applied as classes equalisation method. This is an anticipated random under sampler in the sense that the under sampling is done on the dataset associating a pixel spectra with its associated class and not on the dataset created by taking into account the subwindow. In this way, there is a time and space gain. Indeed, thanks to the anticipated under sampling, subwindow would have been created for nothing since they would be dropped before fitting the classifier. The anticipated random under sampler allows to compute only subwindows that would be kept for training, which is a non negligible time and space gain.

The goal of this test is to see the impact on different metrics when increasing the window width and height from 1 to 29. The window width and height have always the same values in the tests in this subsection. Also note that the case of a window width and height equal to 1 corresponds exactly to the traditional setting where each pixel class is trained or estimated with only the spectrum of its pixel.

Results are available in figures 3.14, 3.15, 3.16, 3.17, 3.18 and 3.19. For the epithelium class, we can see that the f1-score increase slightly when giving small subwindow compared to the elementary subwindow (subwindow whose width and height is equal to 1). But then, for higher subwindows, it decreases slowly, but constantly. We can observe that the precision is the main metric that makes decrease the f1-score. The recall is more constant than the precision. For the stroma class, the inverse can be noticed. The precision is quite stable according to the subwindow size but the recall is quite unstable, which makes the f1-score also unstable. The most strange result is the bad results for recall only when the width and height are equal to 15. But we can also notice that after a width and height of 20, the performances begin to decrease. For the blood class, the curve

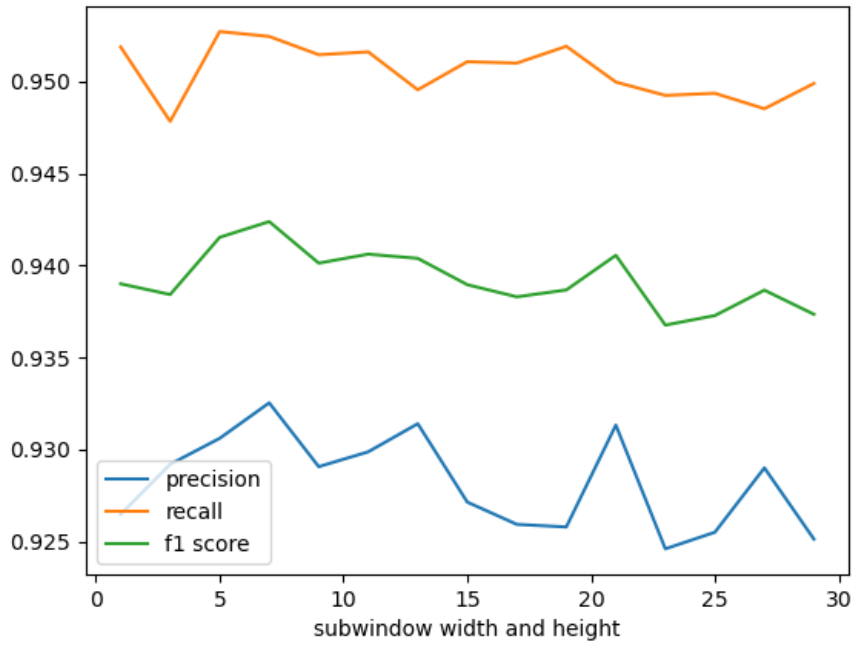

Figure 3.14: Precision, recall and f1-score for the epithelium class according to the width and the height of the subwindows.

for the precision follows approximately the same tendency as the curve for recall for the stroma class, but the instability is still sharper. Conversely, the recall remains extremely stable. The necrosis is the only class where we can clearly see that increasing the subwindows have a positive impact on the precision and f1-score f1-score. However, at the right of the plot, for from a subwindows size of 25 and more, we can see that precision and f1-score plateau. For the recall, for small subwindows, increasing the windows size seems to have a negative implact but this negative impact is reduced after more than a width and height of 19. However, the positive impact of the windows size increasing on the precision is so big that the f1-score always increases when augmenting the window size. Globally, we can see on the macro average that augmenting the window size can increase the f1-score slightly, but there is a limit. After this limit, the f1-score decreases. This behaviour of the f1-score is mainly driven by the precision. The recall remains quite constant. For the weighted average, we can also see that it is possible to increase a little bit the f1-score, but the window size must be chosen carefully. Taking a window size higher than 1 can increase the performance but also decrease them, as we can see for a window size of 15.

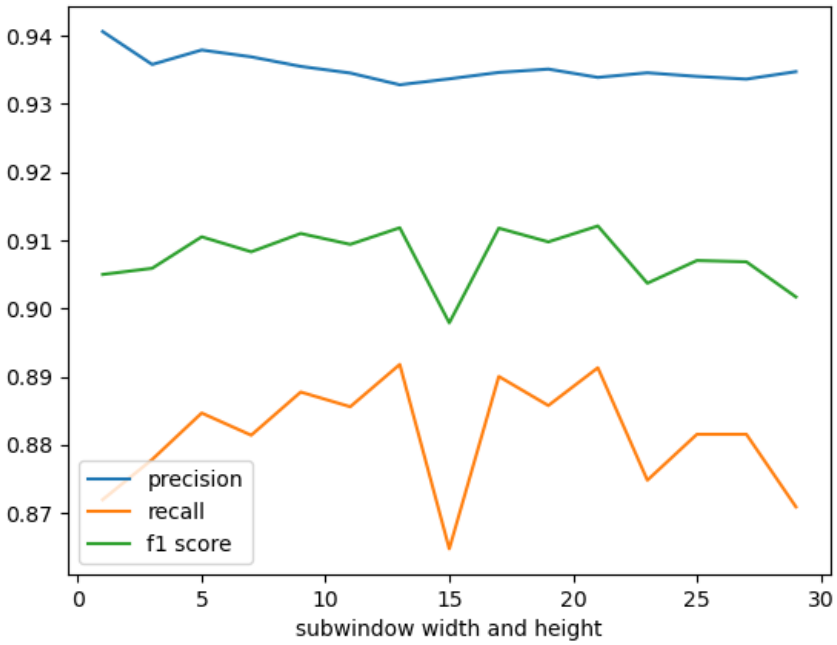

Figure 3.15: Precision, recall and f1-score for the stroma class according to the width and the height of the subwindows.

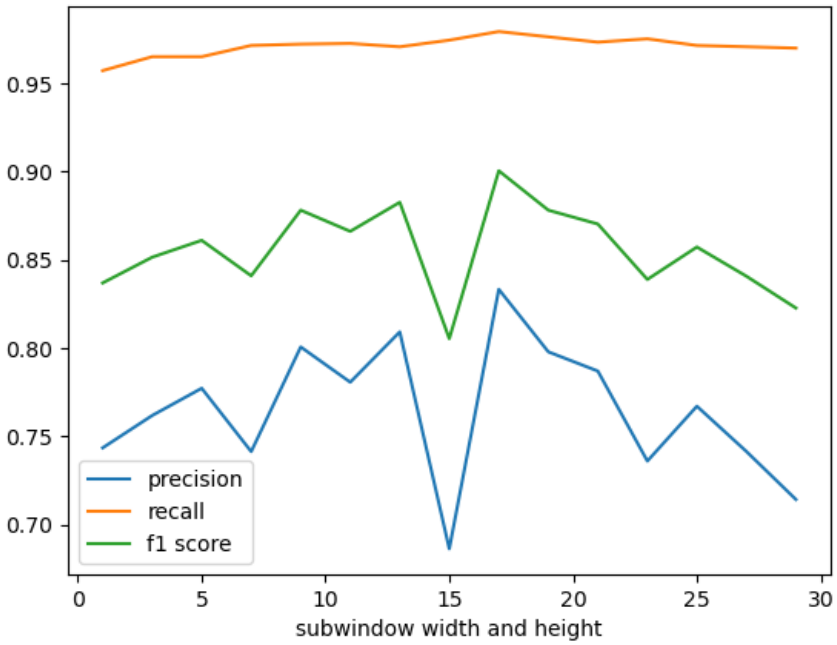

Figure 3.16: Precision, recall and f1-score for the blood class according to the width and the height of the subwindows.

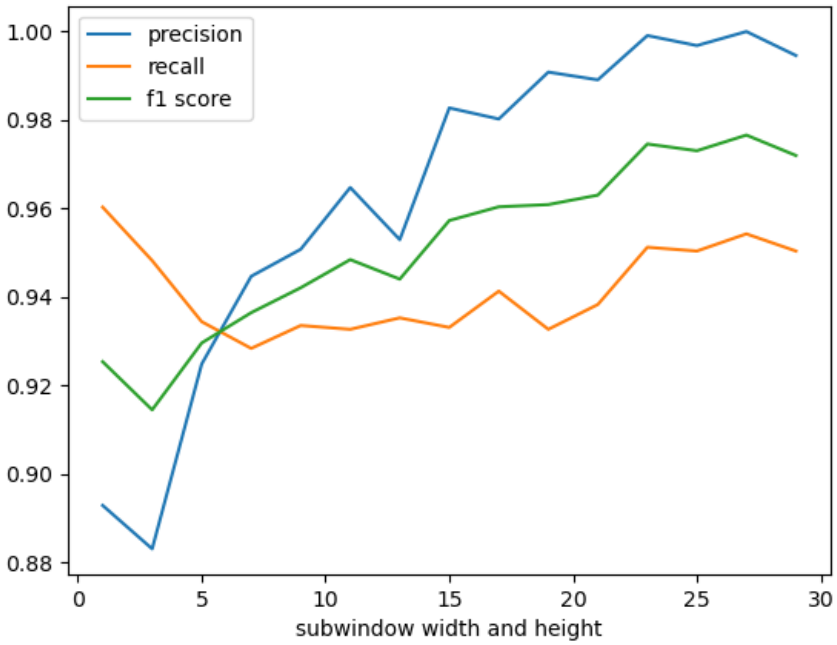

Figure 3.17: Precision, recall and f1-score for the necrosis class according to the width and the height of the subwindows.

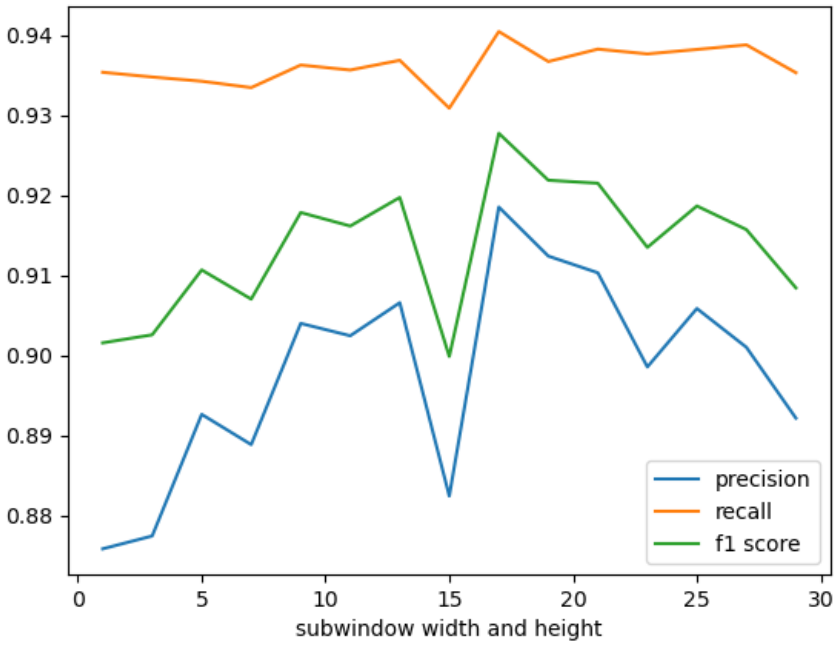

Figure 3.18: Macro average precision, recall and f1-score according to the width and the height of the subwindows.

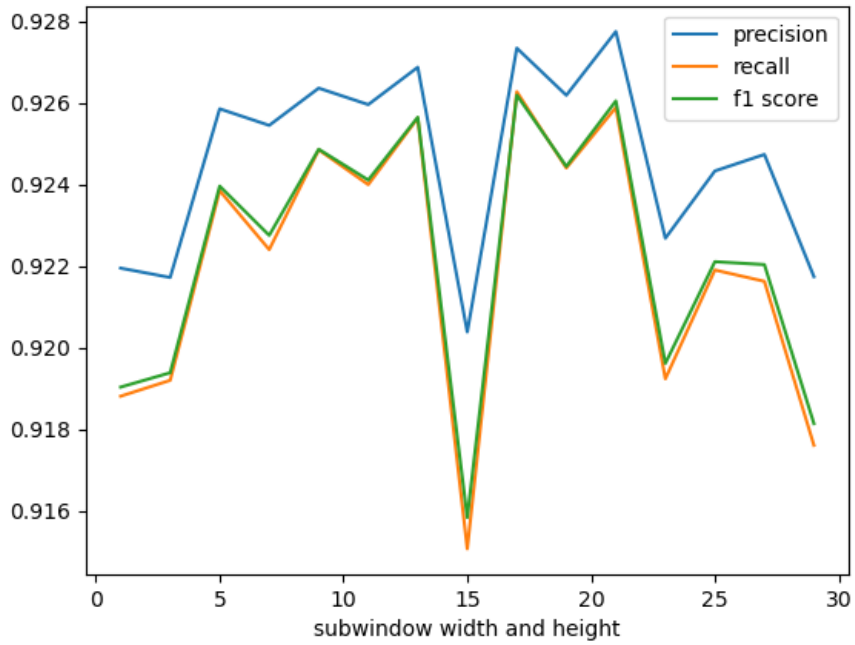

Figure 3.19: Weighted average precision, recall and f1-score according to the width and the height of the subwindows.

3.5.9 Resamplers for classes equalisation

As we saw, the set for histology classification is highly imbalanced. The goal of this section is to apply some resampling strategies in order to balance the dataset. For the tests, no principal component noise reduction is done. No Features selection is done. No normalisation is made. A savitzky-Golay filter is applied with a window size of 13 and the order of the polynomial is set to 4. The second derivative is computed.

The results are available in table 3.5. We can observe that the f1-score for the blood are really affected by the resampling strategy. For the other classes, the differences between equalisation strategies are not very sharp most of the time. An interesting observation is that in average, under sampling techniques work better than over sampling strategies or combination of over and under sampling. For over-sampling strategies, only the borderline-1 algorithm and the random over sampler seems to give similar performances as most of the under sampling strategies. However, the over sampling strategies are more costly in term of space and time because the classifier takes more time to train on the over sampled dataset. But this additional time seems to bring no benefits. For the combined over and under sampling algorithms, it is not very good as well. The main cause of this performance decrease is the bad ability of the resulting classifiers at predicting blood classes. None of the combined over and under sampling techniques give better weighted f1-score than applying no resampling methods. For the under sampling techniques, the near miss version 3 algorithm seems better than all the other resampling techniques. This algorithm has the highest f1-score for the stroma, the blood and the necrosis class. The epithelium class is the only class where the near miss version 3 loses. For the epithelium class, the random under sampler is the best strategy.

Globally, it seems that applying resampling techniques is not the panacea for improving the classifier f1-score performances. However, two of them remain interesting. The first is the near miss version 3 algorithm which shows better results than applying no resampling techniques for all classes. Furthermore, the near miss algorithm wins on 3 classes out of 4 for the f1-score. The other useful resampler for f1-score is the random under sampler. Indeed, this resampler allows to drastically reduce the training computing time of the classifier while keeping good performances. Moreover, among all resampling techniques, this is the one that takes the least time to obtain the resampled dataset. Even if this resampling technique drops many spectra to get balanced dataset, the performance obtained is better than many other tested resampling techniques. It is also better than applying no resampling techniques when looking at the weighted average f1-score. The random

under sampler is even not far from the near miss algorithm in term of weighted average f1-score. The performance difference might be seen as non significant. Consequently, when considering all aspects, the random under sampler is also a good choice when targeting f1-score.

3.5.10 Final model analysis

Now, we we learn a final model from what we learned from all the tests. This model will be analysed more deeply. Only the quality test, the Savitzky-Golay filter and a random under sampler are applied as pre-processing operation. For the Savitzky-Golay filter, the window size is 13 and the polynomial is of degree 4. The second derivative is computed. No subwindow strategies are applied. A random forest classifier is trained. The number of trees in the forest is equal to 100, the number of features selected at each node split is equal to the square root of the number of features in the training set. The minimum number of samples per leaf is set to 10.

The confusion matrix of the predictions on the test set is shown in table 3.6. Now, let's analyse the outputs on the real images of the cores. All the 15 test set cores are shown in figures from 3.6 to . We can see that most of the times, the pixels wrongly classified are from complete regions of interest. It is seldom to see lots of granularity in the results.

Now, we will analyse the feature importance of the model. We can see on figure 3.35 that impurity based and permutation based features importance tie in quite well. We can see that the interval between 1500 cm^{-1} and 1700 cm^{-1} are the most important features. Another interesting plot is shown in figure 3.36 showing the cumulated importances according to the proportion of most important features. We can see that for the 3 feature importances computed, we need few proportion of features to get high cumulated features importances. For example, we can see that 40% of features already contains 95% of the feature importances. Therefore, a new random forest has been trained but only with the most important until 95% of importance cumulated. It means that only 87 features out of 223 are provided to the classifier. We obtain a weighted f1-score of 92%, exactly the same than the model trained with all features. This is a really important result because it allows not only to reduce the training time of the model, but also to potentially adapt the protocol of quantum cascade laser spectra acquisition so as to obtain a higher throughput, with is crucial for future potential applications in clinics.

equalisation strategy	**epithelium class**	**stroma class**	**blood class**	**necrosis class**	**weighted average**
no resamplers	92.51	90.99	89.25	94.14	91.77
edited nearest neighbours resampling = not minority class selection = all neighbours must agree	92.77	91.43	89.04	94.71	92.11
edited nearest neighbours resampling = not minority class selection = a majority vote is done	92.64	91.32	89.1	94.9	92.01
edited nearest neighbours resampling = all classes selection = all neighbours must agree	92.66	91.24	89.08	94.36	91.96
edited nearest neighbours resampling = all classes selection = a majority vote is done	92.52	91.2	89.45	94.19	91.88
near miss version 1	94.31	91.79	75.5	56.23	90.68
near miss version 2	88.48	90.37	58.53	94.84	88.07
near miss version 3	93.69	91.79	90.12	96.97	92.85
random under sampler	94.13	90.6	82.7	93.35	92.04
tomek links resampling = not minority class	92.7	91.29	89.14	94.33	92.0
tomek links resampling = all classes	92.62	91.19	89.48	94.32	91.93
random over sampler	92.7	90.96	87.67	94.65	91.8
SMOTE	91.07	89.93	67.75	93.62	89.55
SMOTE edited nearest neighbours resampling = not minority class selection = all neighbours must agree	90.98	89.71	67.73	91.58	89.32
SMOTE edited nearest neighbours resampling = not minority class selection = a majority vote is done	91.03	89.78	68.8	92.57	89.47
SMOTE edited nearest neighbours resampling = all classes selection = all neighbours must agree	90.97	89.71	67.49	94.26	89.42
SMOTE edited nearest neighbours resampling = all classes selection = a majority vote is done	91.08	89.9	67.54	92.81	89.49
SMOTE tomek resampling = not minority class	91.03	89.78	68.37	93.82	89.5
SMOTE tomek resampling = all classes	91.02	89.71	67.87	92.23	89.37

Table 3.5: F1-scores of the random forest classifiers for different classes equalisation strategies. Results are expressed in percent. When a parameter is not specified, it means that the default value in imbalanced-learn python module is used.

True class \ Predicted class	epithelium	stroma	blood	necrosis
epithelium	95.42	4.34	0.16	0.08
stroma	7.95	87.86	3.23	0.96
blood	4.77	0.00	95.23	0.00
necrosis	0.60	4.06	0.00	95.34

Table 3.6: Confusion matrix for the final model on the test set. The confusion matrix is normalised according to true classes. Results are expressed in percent.

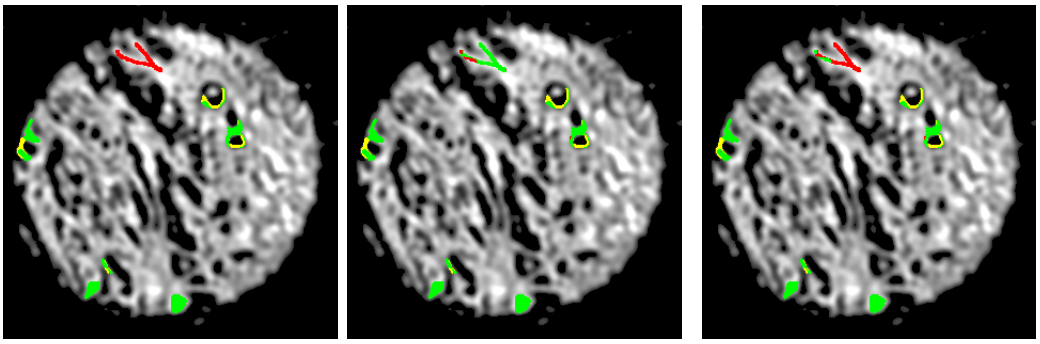

Figure 3.20: label core = A11. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

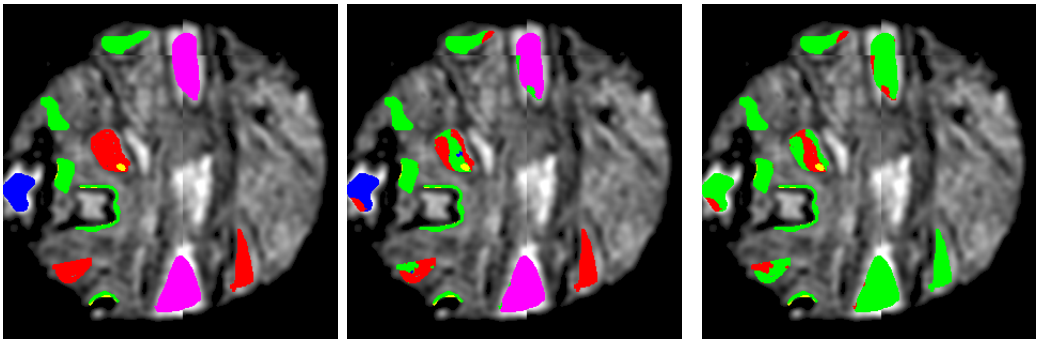

Figure 3.21: label core = B4. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

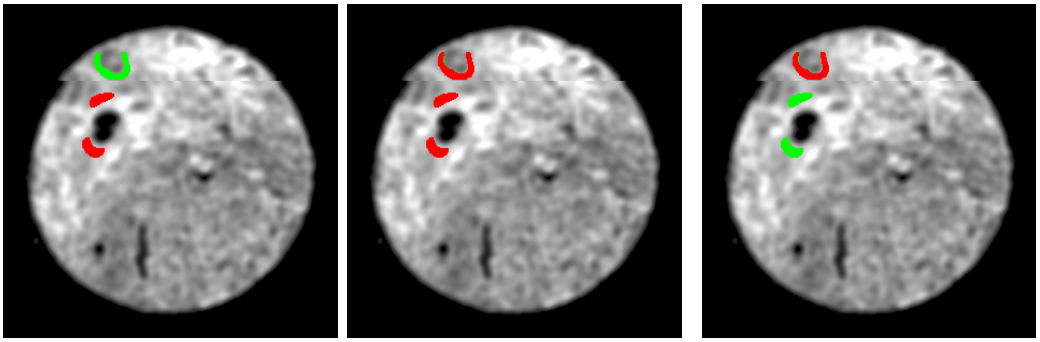

Figure 3.22: label core = B7. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

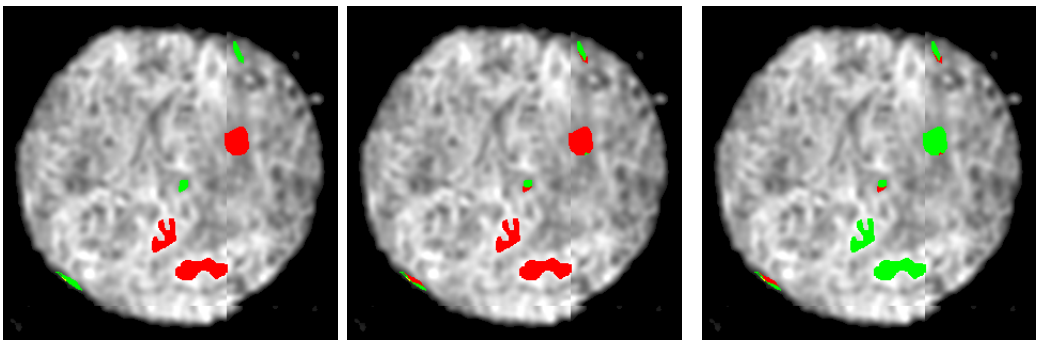

Figure 3.23: label core = D13. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

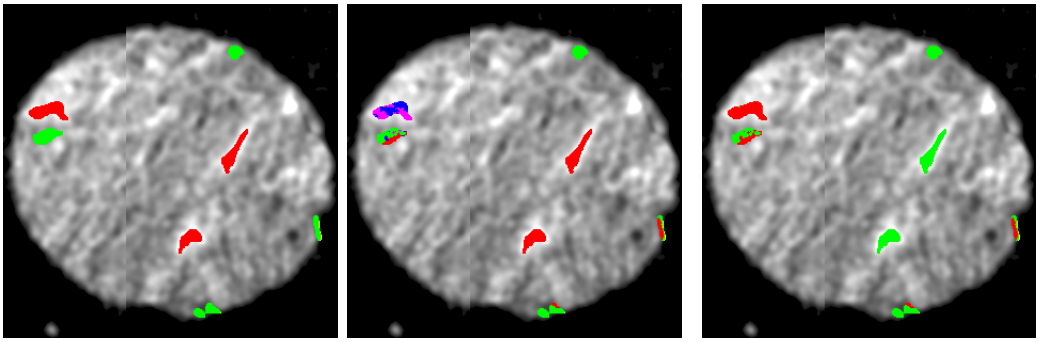

Figure 3.24: label core = E3. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

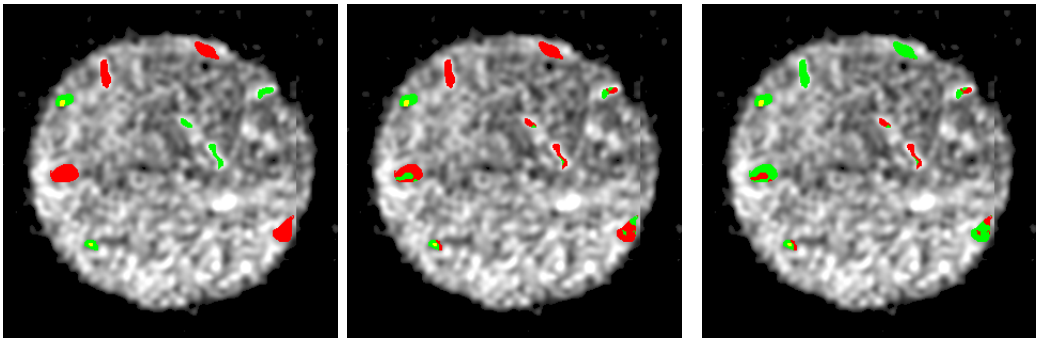

Figure 3.25: label core = F9. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

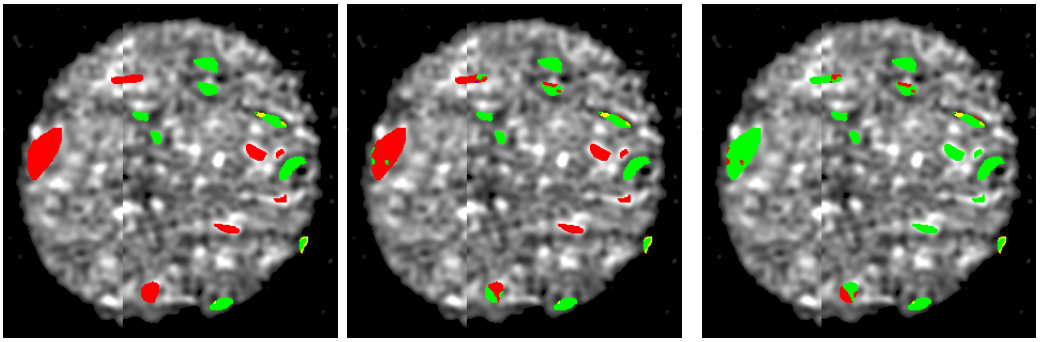

Figure 3.26: label core = F12. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

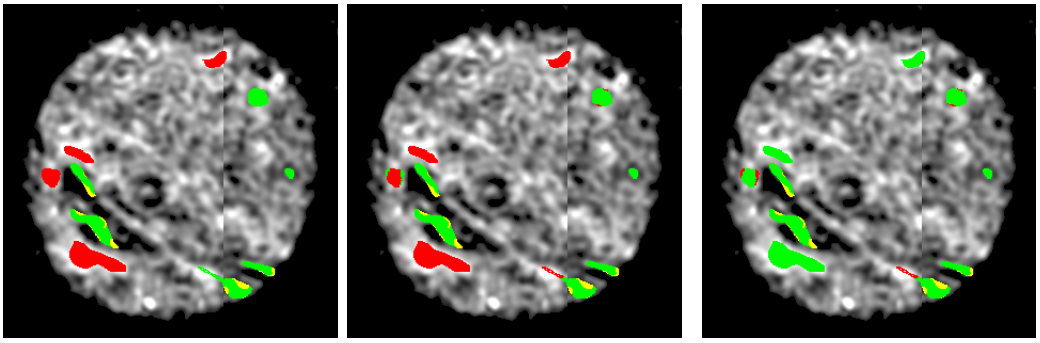

Figure 3.27: label core = J4. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

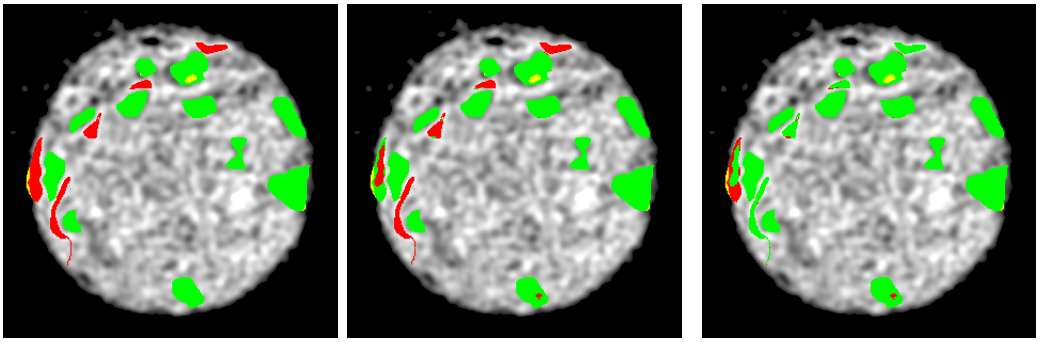

Figure 3.28: label core = J6. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

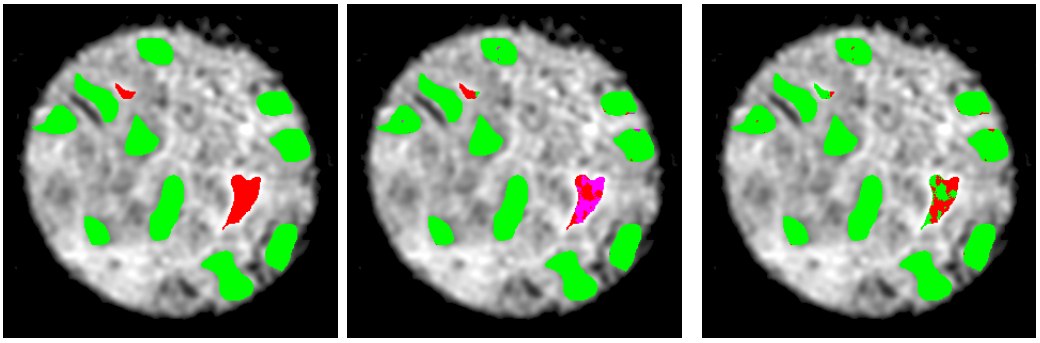

Figure 3.29: label core = J11. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

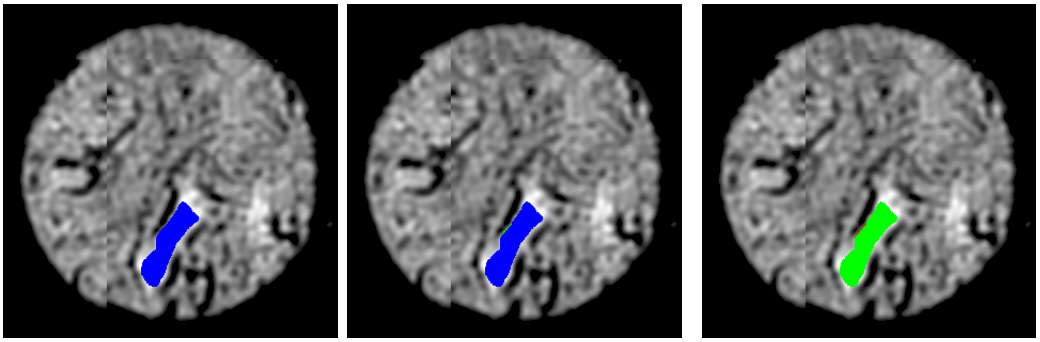

Figure 3.30: label core = K3. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

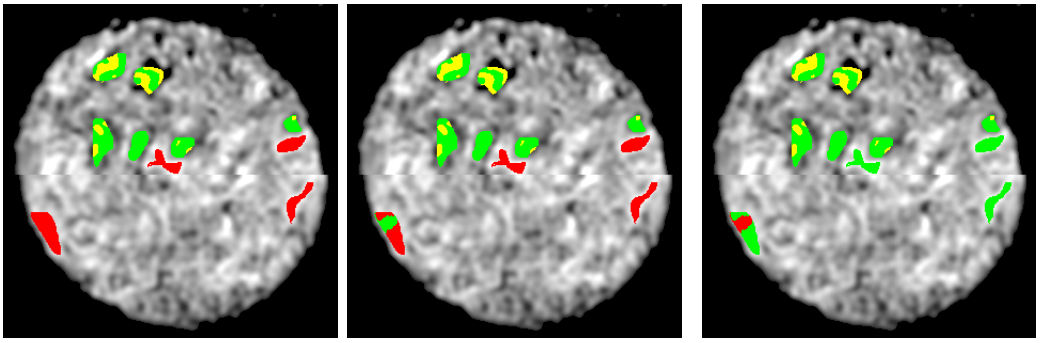

Figure 3.31: label core = L2. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

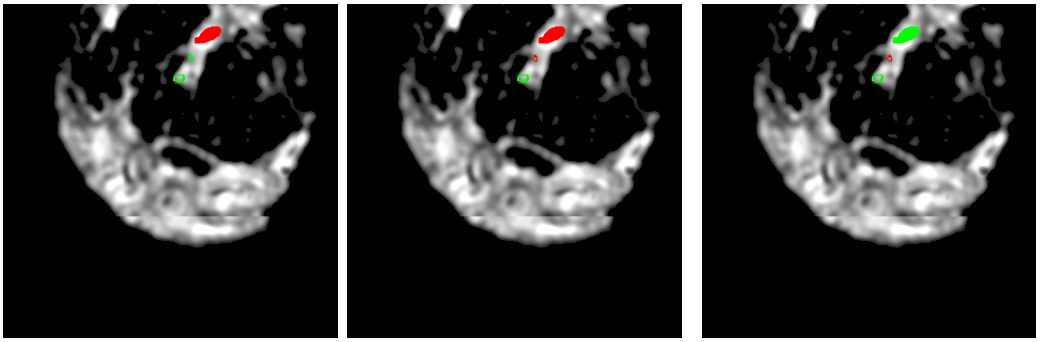

Figure 3.32: label core = M2. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

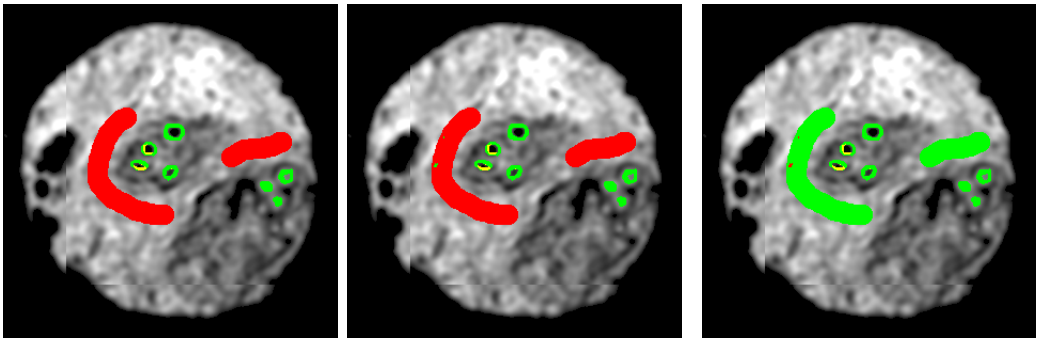

Figure 3.33: label core = M3. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

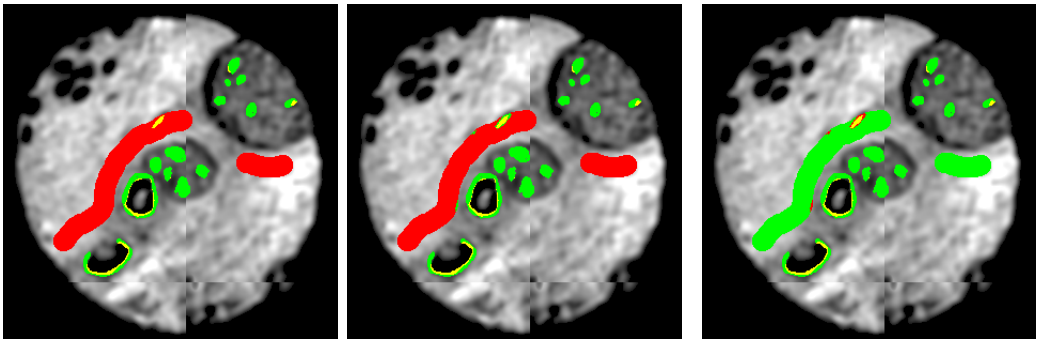

Figure 3.34: label core = M4. Left image = real classes. Centre image = predicted classes. Right image = accuracy classes. For left and centre image: green = epithelium, red = stroma, purple = blood, blue = necrosis, yellow = pixel rejected by the quality test. For the right image: green = pixel correctly classified, red = pixel wrongly classified

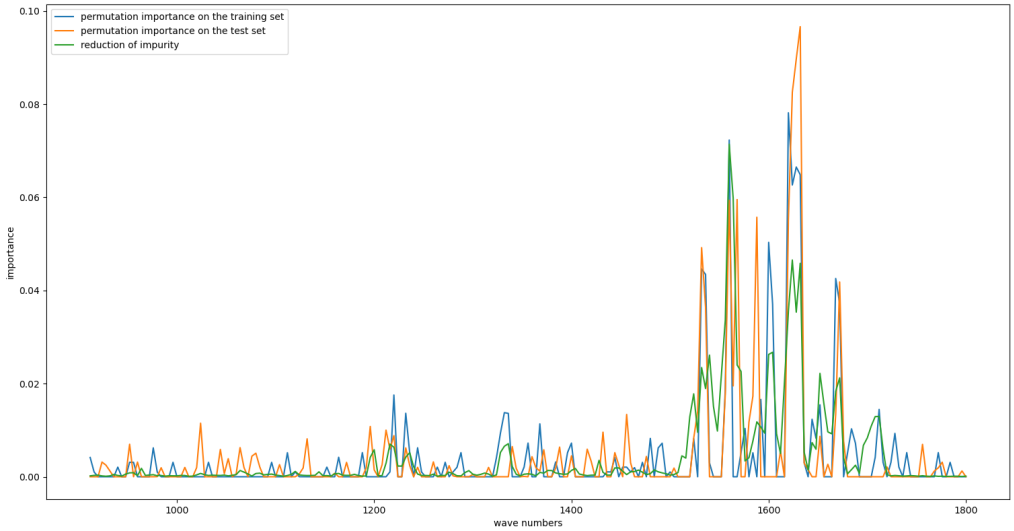

Figure 3.35: features importances normalized to 1

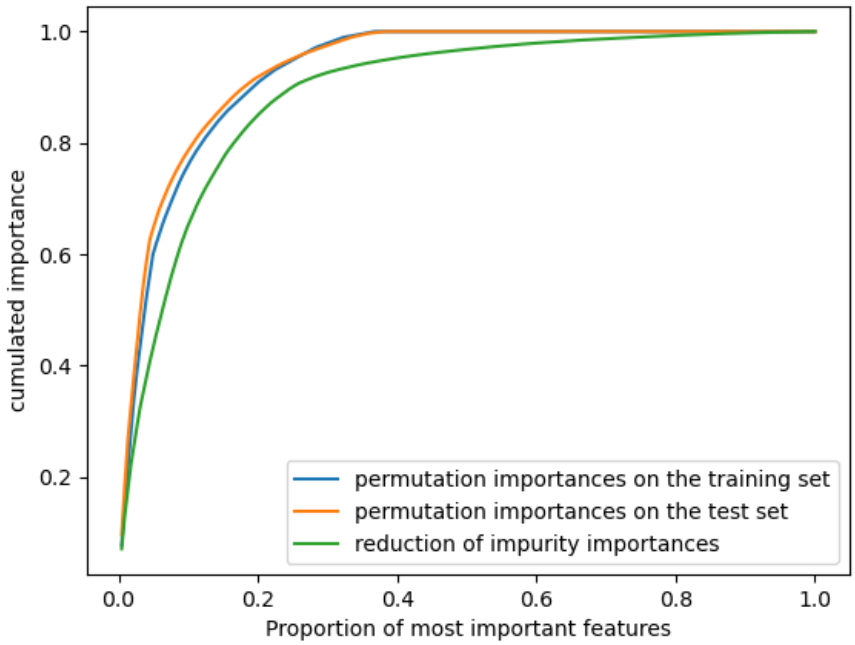

Figure 3.36: importance of features cumulated according to features importance sorted from highest to lowest

Chapter 4

Conclusion

In conclusion, we have developed new modules for multispectral data analysis. These modules were applied to histology classification to obtain very good accuracy and f1-score, which allow potentially to apply the developed models in real medicine application. We showed that some pre-processing operations that some litterature believed to be important for the classifier are actually not so important. These are noise reduction, à priori range selection and normalisation. The Savitzky-Golay filter has shown to be really important for making a good model. We showed that the random forest is really for this kind of application. Subwindows methods showed to be ot very important. They did not improve significantly the classifier performance. Also, we have also seen that none of the tested classes equalisation algorithms are good in creating better classifier. Only the random under-sampler is good because it reduces significantly the training time without impacting on the performance globally. We have shown that in fact, it is possible to take only 40% of the features to achieve this high accuracy, which is really important for future application where high throughput is needed.

Bibliography

[1] Zoph Barret, Vasudevan Vijay, Shlens Jonathon, and Le Quoc V. Learning transferable architectures for scalable image recognition. 2018. Available at https://arxiv.org/pdf/1707.07012.pdf.

[2] Michael J. Pilling, Alex Henderson, and Peter Gardner. Quantum cascade laser spectral histopathology: Breast cancer diagnostics using high throughput chemical imaging. *Analytical Chemistry*, 89(14):7348–7355, 2017. PMID: 28628331.

[3] Full light spectrum and visible spectrum. Available at https://www.pngwing.com/en/free-png-pjjvt.

[4] Additive and subtractive color. Available at https://www.photographytalk.com/217-pt-plus/color-theory/6297-additive-and-subtractive-color.

[5] Ulysse Rubens, Renaud Hoyoux, Laurent Vanosmael, Mehdy Ouras, Maxime Tasset, Christopher Hamilton, Rémi Longuespée, and Raphaël Marée. Cytomine: Toward an open and collaborative software platform for digital pathology bridged to molecular investigations. *PROTEOMICS – Clinical Applications*, 13(1):1800057, 2019.

[6] Decision tree and entropy algorithm. Available at https://zhengtianyu.wordpress.com/2013/12/13/decision-trees-and-entropy-algorithm/.

[7] Pierre Geurts and Louis Wehenkel. Classification and regression trees. Available at https://people.montefiore.uliege.be/lwh/AIA/IML___Trees.pdf.

[8] Permutation feature importance. Available at https://scikit-learn.org/stable/modules/permutation_importance.html.

[9] Illustration of the sample generation in the over-sampling algorithm. Available at https://imbalanced-learn.org/stable/auto_examples/over-sampling/plot_illustration_generation_sample.html.

[10] Michael J. Pilling, Alex Henderson, and Peter Gardner. Quantum cascade laser spectral histopathology: Breast cancer diagnostics using high throughput chemical imaging, 2017. Available at `https://zenodo.org/record/808456`.

[11] Paul Bassan, Miles J. Weida, Jeremy Rowlette, and Peter Gardner. Large scale infrared imaging of tissue micro arrays (tmas) using a tunable quantum cascade laser (qcl) based microscope. *Analyst*, 139:3856–3859, 2014.

[12] Bruce MacEvoy. Additive and subtractive color mixing, 2015. Availalble at `https://www.handprint.com/HP/WCL/color5.html`.

[13] Raphaël Marée, Loïc Rollus, Benjamin Stévens, Renaud Hoyoux, Gilles Louppe, Rémy Vandaele, Jean-Michel Begon, Philipp Kainz, Pierre Geurts, and Louis Wehenkel. Collaborative analysis of multi-gigapixel imaging data using Cytomine. *Bioinformatics*, 32(9):1395–1401, 01 2016.