# 1 Steps to make the codes work

Source codes have been run on linux 64 bits. In order to make them work, use the following steps:

1. Dowwnload the follwing archives:

   - `https://zenodo.org/record/808456/files/br20832_cores.zip?download=1`
   - `https://zenodo.org/record/808456/files/overlay.zip?download=1`

   The first archive contains the datacube files. The second archive contains the overlay files.

2. Unzip the archives where you want on your storage device. The location of each unzipped archive will be given as a parameter to the programs.

3. Copy the python files present in the folder codes_histology_classification where you want on your storage device.

4. You can recreate the anaconda environment used by using the environment_histology_classification.yml file provided in the repository. To recreate the environment, use the following command:

   ```
   conda env create -f environment_histology_classification.yml
   ```

5. Some parts of the code are written in cython language. In order to compile the cython code, use the following command in a terminal:

   ```
   python histology_classification_compilation_file.py build_ext --inplace
   ```

6. Open one of the files following files to edit the parameters:

   - histology_classification_multiple_tests.py;
   - histology_classification_with_feature_selection;
   - histology_classification_analyze_model_computation.py;
   - histology_classification_plots_feature_importance.py;
   - histology_classification_plots_classification.py;
   - histology_classification_plots_mean_spectra_class.py

   the parameters explanations can be found in next section. The parameters are in the end of the file below the line if "__name__" == "__main__". Note that other python files can also be found in the tests folder in order to make plots in specific tests.

7. Run the python file you just edited by typing in a terminal:

   ```
   python [name of the python file]
   ```

# 2 Explanations for specific files

## 2.1 histology_classification_multiple_tests.py

This is probably the first code that you want to run. It allows to load the data, make different preprocessing operations and to train a classifier with the preprocessed training set. Finally, the model is tested on the test set. Various parameters can allow to make tests under different settings.

### 2.1.1 Parameters

- SEED: a seed used for the random number generator. This is useful for reproducibility.
- RANDOMSTATE: random number generator object to use.
- OVERLAYPATH: The path of the folder where you have stored the overlay files.
- DATACUBEPATH: The path to the folder where the datacube files are stored.
- KFOLDINDICESPATH: The path to a file useful for splitting the dataset in a learning and a test set. The separation is made at the core level. The file is provided in the repository. It is named "splitting_file.mat".
- COLOURS: A numpy array storing the colours of the classes stored in the overlay files. To make the code work, you must not change this parameter. This parameter is here if codes are reused for other datasets in the future.
- PEAKWAVEQUALITYTEST: The wave number at which the peak of absorbance must be found for the quality test to keep the pixel.
- THROUGHWAVEQUALITYTEST: The wave number at which the trough of absorbance must be found for the quality test to keep the pixel.
- MINIMUMQUALITYTEST: The minimum absorbance to have when computing the absorbance difference between the peak and the trough.
- MAXIMUMQUALITYTEST: The maximum absorbance to have when computing the absorbance difference between the peak and the trough.
- MAKEQUALITYTESTTRAINIGSET: A boolean telling if the quality test must be done on the training set.
- MAKEQUALITYTESTTESTSET: A boolean telling if the quality test must be done on the test set.
- NOISEREDUCTIONFIRSTLIST: A list of boolean telling if we want to make the principal component based noise reduction before a priori interval selection. The array allows to put all possibilities you want to test.
- NOISEREDUCTIONLIST: A list of tuples with each tuple containing 3 values. Each tuple corresponds to a different test to make. The first tuple element is the name of the class which will apply the principal component analysis. The class must respect the scikit-learn conventions. The second element in the tuple is a boolean telling if we want to get back the

original space after applying principal component analysis. The third element in the tuple is a dictionary storing additional parameters to provide the the principal component analysis class, except the number of components to keep (see next parameter). The keys are a string the parameter name and the value of the key is the value associated with the parameter. Note that the first element can be set to None for making no principal components analysis. The two other parameters in the tuple can then be set to anything.

- NUMBERPRINCIPALCOMPONENTSLIST: A list of number of principal components to test.

- INTERVALSTOSELECTLIST: A list of numpy arrays. Each numpy array correspond to a different test. The numpy array must have a number of rows equal to the number of intervals to select. The number of columns must be equal to 2. In a given row, the first element is the inferior boundary of the interval to select. The second element is the superior boundary of the interval to select. Instead of a numpy array, None can be provided in order to bypass a priori interval selection.

- NORMALIZE: A list of boolean telling if the spectra must be normalized. The list allows to potentially specify both settings in order to test both.

- METHODSAVITZKYGOLAYFILTERLIST: A list which can contain two strings: savitzky_golay_filter and first_derivative. The first string allows to use the scipy.signal.savgol_filter function from scipy module. The second string uses a reimplementation in python of a matlab function (which can be found on the matlab folder you downloaded) smoothing spectra with a Savitzky-Golay filter and computing the first derivative. The list allows to put both settings to test both.

- SMOOTHINGPOINTSLIST: A list containing each of the number of smoothing points to use for the Savitzky-Golay filter. The number of smoothing points is also called the window size.

- ORDERPOLYNOMIALFITLIST: A list of the orders of the polynomial used to fit the smoothing points. Each order in the list will be tested.

- ADDITIONALPARAMETERSSAVITZKYGOLAYFILTERLIST: This is a list of dictionaries of additional parameters to provide to the scipy.signal.savgol_filter function. In the dictionary, the keys are the string of the parameters name and the values associated to the keys are the corresponding parameters values. This parameter is only used if the Savitzky-Golay function string set in METHODSAVITZKYGOLAYFILTERLIST is equal to savitzky_Golay_filter. Otherwise, the parameter is ignored and can be set to anything.

- DICTIONARYCLASSESMAPPING: a dictionary whose keys are a tuple of numbers corresponding to classes and the values are the new class value to set for the corresponding classes. This parameter is used to merge malignant and non-malignant epithelium pixels and malignant and non-malignant stroma pixels. In order to make the code works for histology classification, you must not change this parameter. This parameter is available if the code is reused for other datasets in the future.

- CLASSESEQUALIZATIONLIST: A list of tuples containing two elements. The first is a class that implements a resampling methods. It must implement the imbalanced-learn python

module conventions. The second element in the tuple is a dictionary giving additional parameters to the resampling class. The keys contains strings of the parameters name and the associated values are the values of the corresponding parameters. The string "anticipated_random_under_sampler" is also a valid value for the first element of the tuples. When subwindows containing more than one pixel spectrum are created, anticipated_random_under_sampler allows to resample before creating the subwindows, allowing to speed up the process and gaining space as subwindows for pixel removed by resampling will not be computed. If a value different to "anticipated_random_under_sample"r is given, then classes equalisation occurs after creating the subwindows. For subwindow of size equal to one single pixel, making an anticipated random under sampler is the same as making a classical random under sampler. When anticipated random under sampler is used, the potential additional parameters are the same as for the classic random under sampler. Note that classes equalisation can be bypassed by setting the first tuple element to None and the second to an empty dictionary.

- FEATURESELECTIONLIST: A list of tuple of two elements where the first element of the tuple is a feature selection class and the second is a dictionary of additional parameters to provide to the feature importance class. The class must respect scikit-learn conventions. This feature selection operation can be bypassed by setting the first element of the tuple to None and the second to an empty dictionary. Note that the feature selection operation available here is not the same as the feature selection in histology_classification_with_feature_selection.py. For the histology_classification_multiple_tests.py code, feature selection is an operation made after classes equalisation.

- MODELALGORITHMLIST: A list of tuples of two elements where the first element is the class of the learning algorithm to use. The second element of the tuple is a dictionary of additional parameters to provide to the training algorithm. The learning algorithm must respect the scikit-learn conventions. The list allows to create several tuples. Each of these tuples will be tested.

- TESTPARAMETERSLIST: A list of tuples with two elements. The meaning of all two elements differ in function of the subwindow dimension. There exists two cases. The first is when the width and the height of the subwindows are set to 1. The other case is any other configuration of width and height of the subwindows. For the first configuration, if the learning algorithm is not a bagging method, then the second element of the tuple must be set to None and the first to anything. If the learning algorithm is a bagging method, then the value "hard" or "soft" can be set as first tuple element and the second tuple element can be any value between 0 and 1. This is used for telling to the code that if the probability of the most probable class for a predicted pixel is below the value in the second tuple element, then the pixel prediction is rejected and the pixel remains unpredicted and not used for computing the classifier performances. The "hard" set to first tuple element tells to create pixel probabilities by making a prediction on each model in the ensemble algorithm. The probability of each class is the proportion of models in the ensemble having predicted the given class. When "soft" is used as first tuple element, then the prediction probability of each class is computed on each model in the ensemble and then these probabilities vector are averaged to get the final vector of probabilities. When subwindows width and/or height sizes are greater than 1, then if the class predicted is only the centre of the subwindows, then the parameters in the

tuple have the same behaviour as in the classification with a width and a height of 1. When creating a multioutput classification tasks on subwindows of width and/or height greater than one, then the second tuple element must always be set to None. The first tuple element can be set to "hard" or "soft". In the "hard" case, it means that for predicting a pixel class, all subwindows where the given pixel is present are predicted, then the predicted classes for the given pixels are obtained from each of the subwindows in which it appears. Then, a majority vote is done to determine the pixel class. In the "soft" case, it means that all subwindows in which a given pixel is present are predicted with the corresponding probability vectors. Then, the probability vectors for each the subwindows predictions for the given pixel are gathered and averaged. The most probable class is then selected as the predicted class. No pixel rejection due to low probability of the most probable class is implemented in the multioutput case.

- FILENAMETEXT: A string for a path to a file created by the code giving information on all tests done and the performances obtained for the classifier. Each test is given an unique identifier value provided in the file written. The path to the file does not need to exist before the code execution. The code will create the path in that case. Note that if a file already exists at the given location, this file will be erased and replaced by the new one.

- FOLDERDATAFRAME: This is a string containing a path to a folder where some dataframes in csv format will be stored. These dataframes store metrics showing the classifier performance for a given test. Each test performance dataframe is stored in a separated file whose name corresponds to the identifier given to the test and this identifier is provided in the file stored in FILENAMETEXT. Note that the path to the dataframes folder does not need to exist before executing the code. However, if the folder already exists at the given path, the dataframes it contains can be deleted.

- SUBWINDOWPARAMETERSLIST: This is a list of tuples containing each 5 elements. The first indicates if we want to make classification by giving to the classifier only the class of the center pixel of the subwindows or if we want to make a multioutput problem. In the first case, the first tuple element must be set to "subwindow_center" and in the second case to "subwindow_multioutput". Note that if the width and the size of the subwindow is set to 1, setting None to the first tuple element is also accepted. The second tuple element is the width of the subwindows and the third tuple element is the height of the subwindows. The fourth tuple element is a boolean indicating if data augmentation by making mirrors of the subwindows must be done. The fifth tuple element is also a boolean indicating if data augmentation by making rotations of the subwindows must be done.

- IDMODELBEGIN: An value giving the starting identifier number for the test. After each test, the identifier value is incremented by 1.

- FOLDERMODELS: a string containing a path to a folder where to store the trained models and its parameters and the corresponding parameters used for each pre- and post- processing operations. The file are created using pickle python module. Note that the path to the folder does not need to exist before executing the code. However, if the path to the folder already exists, some files inside it can be deleted and replaced.

## 2.2 histology_classification_with_feature_selection.py

This code is a revised version of histology_classification_multiple_tests.py Only the parameters FEATURESELECTIONLIST is changed. For this file, it stores a list of tuples with 3 elements. The first is a string that can have two values: "impurity_based_importances" or "permutation_importances". The second tuple element is the proportion of most important features to keep for training. The third tuple element depends on the value given to the first tuple element. If "impurity_based_importances" is chosen, then a path to a pickle file created by histology_classification_multiple_tests.py must be specified. If "permutation_importances" is given as first tuple element, then a path to a pickle file created by histology_classification_analyze_models_computations.py must be specified. Note that that the code of histology_classification_with_feature_selection.py is not adapted for subwindows width width or height greater than 1. Only a width and a height of 1 can be used.

## 2.3 histology_classification_analyze_models_computations.py

This code is used to make an analysis on a model trained with the "histology_classification_multiple_tests.py" file. Analysis mainly consists in computing permutation importances of the features.

### 2.3.1 Parameters

- SEED: a seed used for the random number generator. This is useful for reproducibility.
- RANDOMSTATE: random number generator object to use.
- OVERLAYPATH: The path of the folder where you have stored the overlay files.
- DATACUBEPATH: The path to the folder where the datacube files are stored.
- KFOLDINDICESPATH: The path to a file useful for splitting the dataset in a learning and a test set. The separation is made at the core level. The file is provided in the repository. It is named "splitting_file.mat".
- IDMODELBEGIN: A value giving the starting identifier number for the analyses. After each analysis, the identifier value is incremented by 1.
- MODELPATH: The path to the pickle file storing the trained model to analyze generated by the "histology_classification_multiple_tests.py" code
- MAKEPERMUTATIONIMPORTANCES: A boolean telling if permutations importances must be computed.
- MAKETREEINTERPRETER: A boolean telling if each spectra must be analyzed by the treeinterpreter python module.
- FOLDERPERMUTATIONIMPORTANCES: The path to the folder where permutation importances results must be stored. A pickle file is generated for each analysis and usable in histology_classification_with_feature_selection.py and histology_classification_plots_feature_importances.py. Note that the path to the folder does

not need to exist before executing the code. However, if the folder already exists and contains files, these files can be removed and replaced.

- TREEINTERPRETERPATH: The path where to store the results returned by the treeinterpreter python module. Results are stored in files with the pickle python module. Note that the path to the folder does not need to exist before executing the code. However, if the folder already exists and contains files, these files can be removed and replaced.

- PERMUTATIONIMPORTANCESLIST: A list of tuples where each tuple contains 3 elements. The first element is the additional parameters to provide the the scikit-learn implementation for computing permutation importances. These parameters are available at `https://scikit-learn.org/0.23/modules/generated/sklearn.inspection.permutation_importance.html`. The second is a class implementing a resampling algorithm. This class must respect the imbalanced-learn conventions. Note that setting this second tuple element to None will bypass the resampling. The third tuple element is a dictionary giving additional parameters to provide to the resampling class. This third tuple element is ignored if the second is set to None. In this case, the third tuple element must be set to an empty dictionary.

## 2.4 histology_classification_plots_feature_importances.py

This code make various plots about the feature importances computed before. Both reduction of impurity importances and permutation importances are processed. The meaning of each plot is written on the terminal when executing the code.

### 2.4.1 Parameters

- MODELPATH: The path to the pickle file containing the trained model generated by the histology_classification_multiple_tests.py code.

- PERMUTATIONIMPORTANCESPATHTRAININGSET: The path to the pickle file storing the results of permutation importances computed on the training set generated by the histology_classification_analyze_models_computations.py code.

- PERMUTATIONIMPORTANCESPATHTESTSET: The path to the pickle file storing the results of permutation importances computed on the test set generated by the histology_classification_analyze_models_computations.py code.

- USEMODELFEATUREIMPORTANCES: Boolean telling if the feature importances possibly stored directly in the model must be used. This is the case for reduction of impurity importances in tree-based model in scikit-learn.

- ALPHAVALUE: A transparency measure used when creating standard deviations bands in some plots generated by the code.

- NUMBERFEATURESTOSHOWBARPLOT: The number of features to show in bar plots generated by the code.

## 2.5 histology_classification_plots_classification.py

This code allows to generate some images of the classification showing the real classes ox pixels, the predicted classes of these pixels, the pixels correctly and wrongly predicted, the pixels rejected by the quality test and the pixels for which no classification is made because the model is not confident enough about the predictions. Also, some performances statistics are computed for each core independently.

- OVERLAYPATH: The path of the folder where you have stored the overlay files.
- DATACUBEPATH: The path to the folder where the datacube files are stored.
- KFOLDINDICESPATH: The path to a file useful for splitting the dataset in a learning and a test set. The separation is made at the core level. The file is provided in the repository. It is named "splitting_file.mat".
- MODELPATH: The path to the pickle file containing the trained model generated by the histology_classification_multiple_tests.py code.
- METHODPREDICTION and ACCEPTANCETHRESHOLD: Both parameters values depends on the values of the other. The meaning of all two variables differs in function of the subwindow dimension. There exists two cases. The first is when the width and the height of the subwindows are set to 1. The other case is any other configuration of width and height of the subwindows. For the first configuration, if the learning algorithm is not a bagging method, then the ACCEPTANCETHRESHOLD variable must be set to None and the METHODPREDICTION to anything. If the learning algorithm is a bagging method, then the value "hard" or "soft" can be set for METHODPREDICTION variable and the ACCEPTANCETHRESHOLD can be any value between 0 and 1. This is used for telling to the code that if the probability of the most probable class for a predicted pixel is below the value in the ACCEPTANCETHRESHOLD variable, then the pixel prediction is rejected and the pixel remains unpredicted and not used for computing the classifier performances. The "hard" set to the METHODPREDICTION variable tells to create pixel probabilities by making a prediction on each model in the ensemble algorithm. The probability of each class is the proportion of models in the ensemble having predicted the given class. When "soft" is used for METHODPREDICTION variable, then the prediction probability of each class is computed on each model in the ensemble and then these probabilities vectors are averaged to get the final vector of probabilities. When subwindows width and/or height sizes are greater than 1, then if the class predicted is only the centre of the subwindows, then the parameters METHODPREDICTION and ACCEPTANCETHRESHOLD have the same behaviour as in the classification with a width and a height of 1. When creating a multioutput classification tasks on subwindows of width and/or height greater than one, then the ACCEPTANCETHRESHOLD variable must always be set to None. The METHODPREDICTION variable can be set to "hard" or "soft". In the "hard" case, it means that for predicting a pixel class, all subwindows where the given pixel is present are predicted, then the predicted classes for the given pixel are obtained from each of the subwindows in which it appears. Then, a majority vote is done to determine the pixel class. In the "soft" case, it means that all subwindows in which a given pixel is present are predicted with the corresponding probability vectors. Then, the probability vectors for each subwindows predictions for the given

pixel are gathered and averaged. The most probable class is then selected as the predicted class. No pixel rejection due to low probability of the most probable class is implemented in the multioutput case.

- COLOURCLASSES: A 2D numpy array where each row contains a elements of size 3 corresponding to a colour. These are the colour for the classes of pixels (real or predicted). The row indexes of the class colours must be the same as the indexes used for the corresponding classes when training the model.
- COLOURCORRECTCLASSIFICATION: A numpy vector of length 3 storing the colour to use to show a correct classification of the pixels.
- COLOURWRONGCLASSIFICATION: A numpy vector of length 3 storing the colour to use to show a wrong classification of the pixels.
- COLOURREJECTED: A numpy vector of length 3 storing the colour to use to show that the model is not confident enough about the classification of the pixel according to the given threshold. Consequently, the pixel remains unpredicted.
- COLOURQUALITYTESTNOTPASSED: A numpy vector of length 3 storing the colour to use to show that a pixel does not pass the quality test.
- FOLDERRESULTINGIMAGESTESTSET: The path to the folders where to store the core classification images. Note that the path to the folder does not need to exist before executing the code. However, if the path to the folder already exists, some files inside it can be deleted and replaced.
- STATISTICSPATH: The path to a file storing some statistics about each core whose classification images are created. The identifier of each core is provided. The path to the file does not need to exist before the executing the code. However, if the path to the file exists and if a file with the same name as the provided name is present, then this file is removed and replaced by the new file generated by the code.
- FOLDERDATAFRAME: This is a string containing a path to a folder where some dataframes in csv format will be stored. These dataframes store metrics showing the classifier performance for a specific cores. The name of the files are the identifier of each core. Note that the path to the dataframes folder does not need to exist before executing the code. However, if the folder already exists at the given path, the dataframes it contains can be deleted.
- SHOWCOREMICROSCOPE: A boolean telling if the microscope images of the cores must be provided. Otherwise, real classes, predicted classes, the rejected pixels, the pixels which do not passed the quality test and the images showing if the pixels are correctly classified are drawn on black images.
- INVERTTRAININGTESTSET: A boolean telling if the code must process the training set cores. Otherwise, the test set cores are processed.
- MAKEQUALITYTESTTESTSET: A boolean telling if the quality test must be done on the training or test set, depending on the value chosen for the variable INVERTTRAININGTESTSET.

## 2.6 histology_classification_plots_mean_spectra_class.py

This code creates plots showing general properties about the training and the test set. The meaning of each plot is written on the terminal when executing the code.

- OVERLAYPATH: The path of the folder where you have stored the overlay files.
- DATACUBEPATH: The path to the folder where the datacube files are stored.
- KFOLDINDICESPATH: The path to a file useful for splitting the dataset in a learning and a test set. The separation is made at the core level. The file is provided in the repository. It is named "splitting_file.mat".
- COLOURS: A numpy array storing the colours of the classes stored in the overlay files. To make the code work, you must not change this parameter. This parameter is here if codes are reused for other datasets in the future.
- PEAKWAVEQUALITYTEST: The wave number at which the peak of absorbance must be found for the quality test to keep the pixel.
- THROUGHWAVEQUALITYTEST: The wave number at which the trough of absorbance must be found for the quality test to keep the pixel.
- MINIMUMQUALITYTEST: The minimum absorbance to have when computing the absorbance difference between the peak and the trough.
- MAXIMUMQUALITYTEST: The maximum absorbance to have when computing the absorbance difference between the peak and the trough.
- MAKEQUALITYTESTTRAINIGSET: A boolean telling if the quality test must be done on the training set.
- MAKEQUALITYTESTTESTSET: A boolean telling if the quality test must be done on the test set.
- DICTIONARYCLASSESMAPPING: a dictionary whose keys are a tuple of numbers corresponding to classes and the values are the new class value to set for the corresponding classes. This parameter is used to merge malignant and non-malignant epithelium pixels and malignant and non-malignant stroma pixels. In order to make the code work for histology classification, you must not change this parameter. This parameter is available if the code is reused for other datasets in the future.
- ALPHAVALUE: A transparency measure used when creating standard deviations bands in some plots generated by the code.