Paralabe 2020

Projet de fin d’études de Lucien Le Guellec, étudiant en 5e année au département informatique de Polytech Tours, faisant suite au stage de l’été 2019 de Yann Galan, qui a travaillé sur le projet Paralabe

Paralabe2020 : l’objectif

On souhaite créer un programme qui, à partir d’une base d’images extraites de documents anciens et associées à des vecteurs, puisse les diviser en *clusters* dont tous les éléments représenteraient la même chose, permettre à l’utilisateur (**un historien**) de corriger les éventuels défauts de ces *clusters* (en **fusionner** deux s’ils représentent manifestement la même chose, en **séparer** un s’il contient manifestement des éléments différents, **déplacer** une image si elle est dans le mauvais *cluster*) et enfin de les **annoter**, c’est à dire d’indiquer ce qu’ils représentent (un caractère ou une chaîne de caractères, le plus souvent).

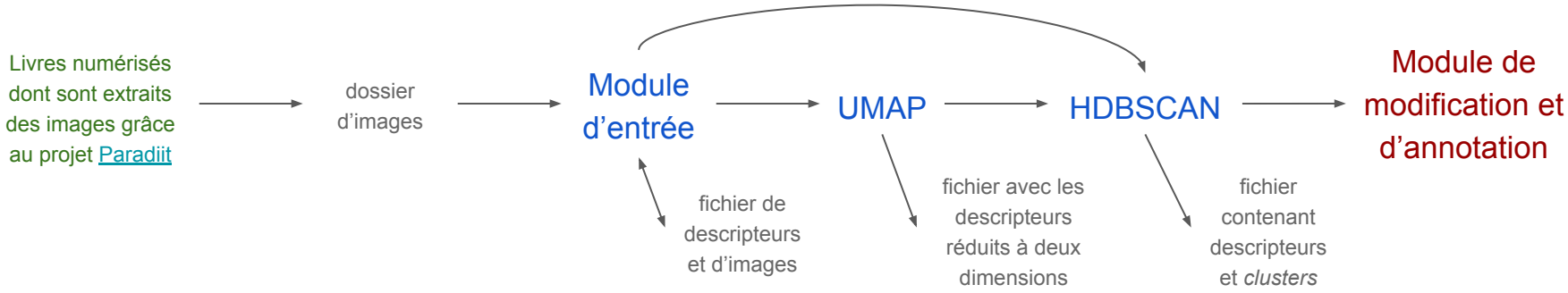
Les premières étapes peuvent déjà être réalisées, mais la suite pose problème, et l’on souhaite qu’elle s’inspire de Cartolabe, voire se fasse directement avec son aide.

Paralabe2020 : les modules et les fichiers

Le projet peut être vu comme une succession de modules, correspondant chacun à une IHM.

- Module 1 : Choix des entrées et production de vecteurs le cas échéant
- Module 2 : UMAP, qui permet la réduction du nombre de dimensions
- Module 3 : HDBSCAN, qui permet le *clustering*
- Module 4 : Modification et annotation des *clusters*

Sur le schéma ci-dessous, est en vert ce qui existe déjà, en bleu ce qui sera produit à la fin du PRD et en rouge ce qui reste à étudier avec l'Inria.



La structure des fichiers JSON

Les fichier traités par Paralabe2020, en dehors des dossiers d'images d'entrée, seront tous au format JSON et respecteront cette structure :

```
[{       "nom": "retro_pattern1.bmp",
       "image": "[...]AA==",
"descripteurs":
            { "nd": [255, 255, [...], 255],
              "2d":  [7.525875091552734, 2.7696139812469482]},
    "clusters":
     { "hdbscan_1": 1, "hdbscan_2": 1},
        "tags": [ "lettre", "a"]},
[...]]
```

→ le nom de l'image (facultatif)

→ les images en base64

→ les descripteurs associés à l'image (il peut y en avoir plusieurs, de dimensions différentes par exemple

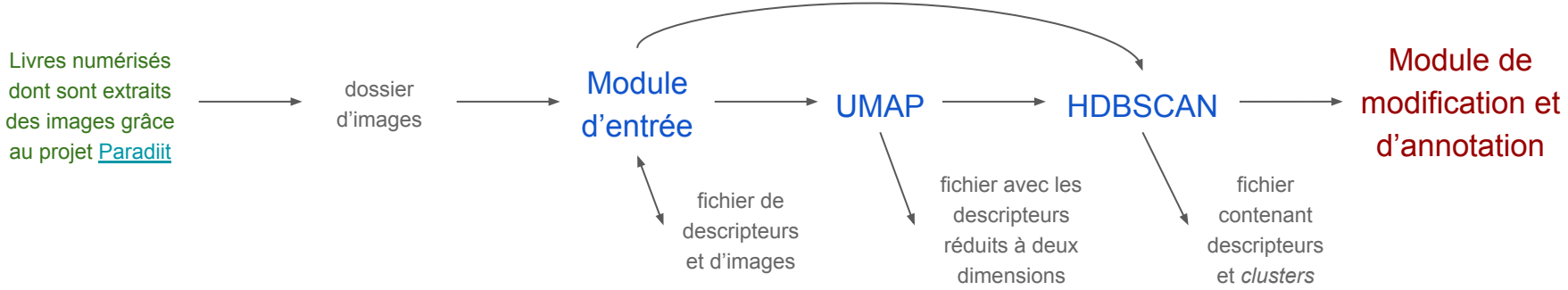
→ les clusters contenant l'image dans chaque instance de *clustering*

→ les tags associés à l'image, qui pourront servir pour l'annotation

→ et ainsi de suite pour chaque image

Tous ces champs peuvent être vides au début du traitement à part "image" et "descripteurs" qui doit au moins contenir une liste.

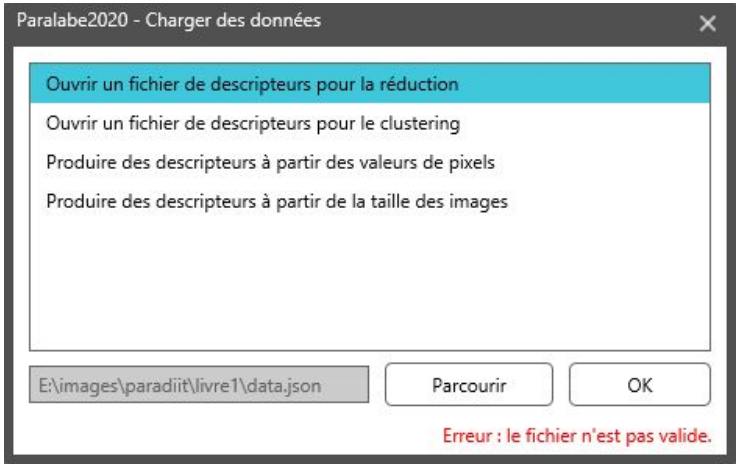
Paralabe2020 : le module d'entrée



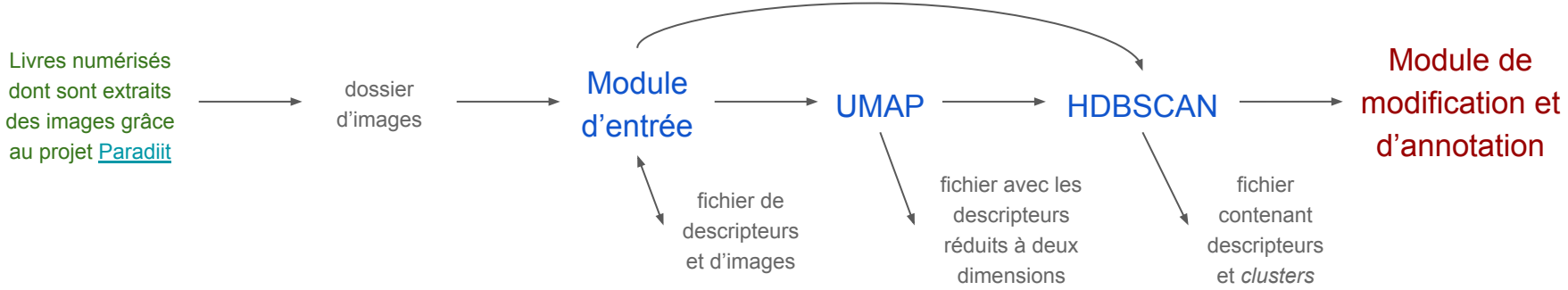
Le module d'entrée proposera quatre options, correspondant chacun à une fonction acceptant un fichier ou dossier d'entrée et produisant un résultat différent.

- **F1** : Ouvrir un fichier de descripteurs pour la réduction
- **F2** : Ouvrir un fichier de descripteurs pour le clustering
- **F3** : Produire des descripteurs à partir des valeurs de pixels
- **F4** : Produire des descripteurs à partir de la taille des images

Ces deux dernières fonctions seront chacune comprises dans une classe fille de la classe Extractor et le programme acceptera de nouveaux algorithmes d'extractions de descripteurs par l'ajout d'une nouvelle classe fille dans le dossier approprié.

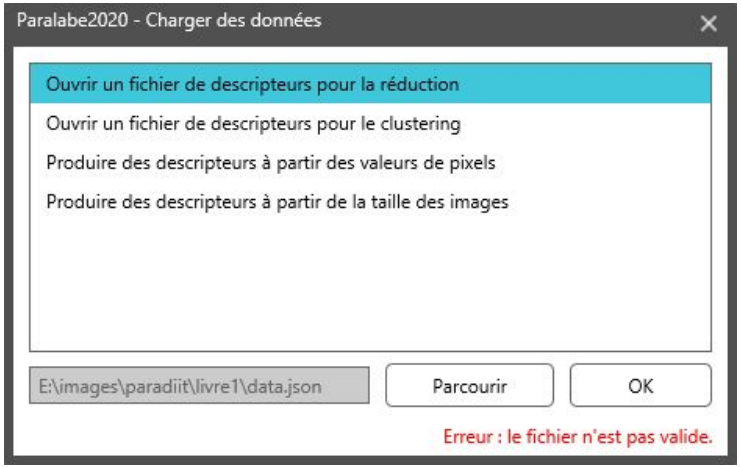


Paralabe2020 : le module d'entrée — F1

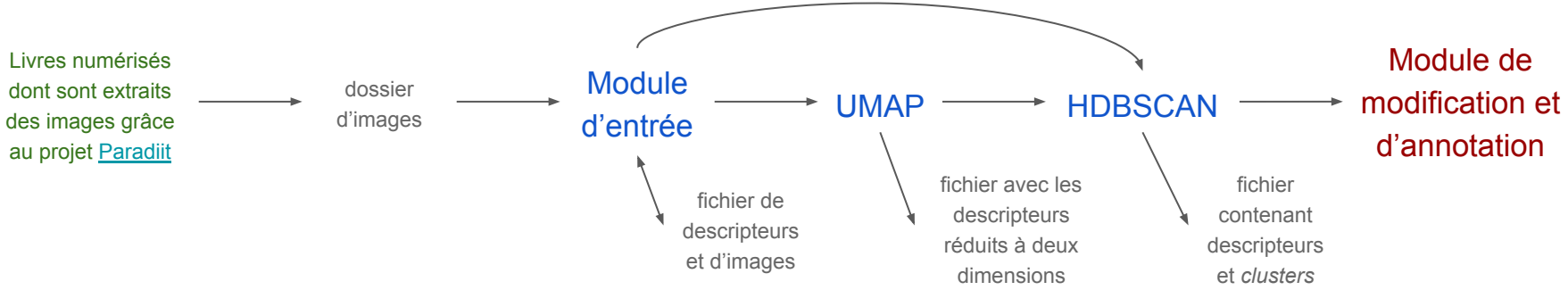


F1 ouvre un fichier JSON contenant déjà des descripteurs, charge les données en mémoire dans une variable dictionnaire et ouvre le second module de réduction des dimensions avec UMAP.

- **Entrée** : fichier JSON conforme
- **Autres paramètres** : aucun
- **Sortie** : aucune
- **Erreur** : si le fichier n'est pas conforme et que les descripteurs ne sont pas trouvés, on ne passe pas au module suivant et l'erreur s'affiche en bas.

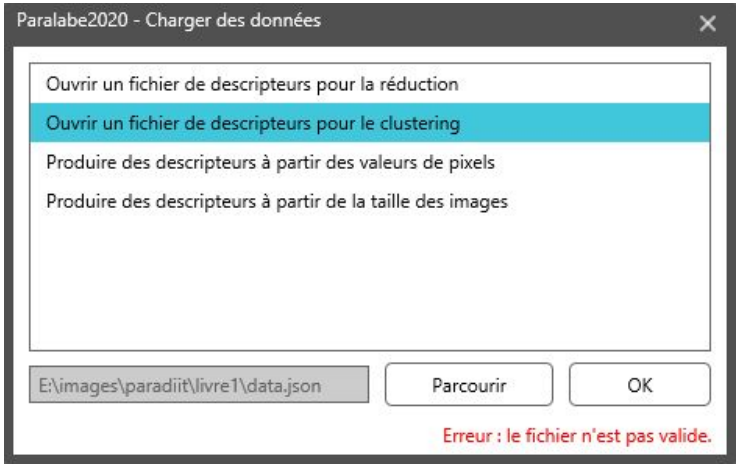


Paralabe2020 : le module d'entrée — F2

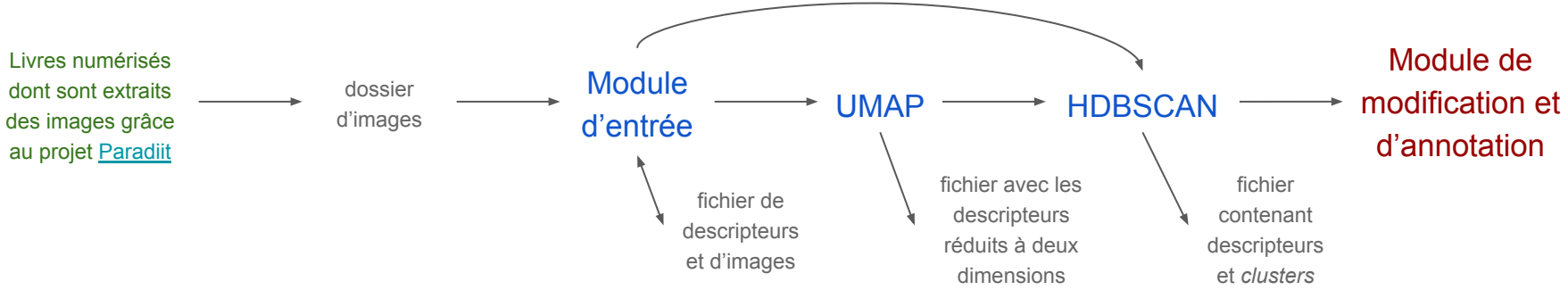


F2 ouvre un fichier JSON contenant déjà des descripteurs, charge les données en mémoire dans une variable dictionnaire et ouvre directement le troisième module de *clustering* avec HDBSCAN, soit parce que le fichier contient déjà des vecteur réduits, soit parce que l'utilisateur souhaite utiliser HDBSCAN sans réduction préalable.

- **Entrée** : fichier JSON conforme
- **Autres paramètres** : aucun
- **Sortie** : aucune
- **Erreur** : si le fichier n'est pas conforme et que les descripteurs ne sont pas trouvés, on ne passe pas au module suivant et l'erreur s'affiche en bas.



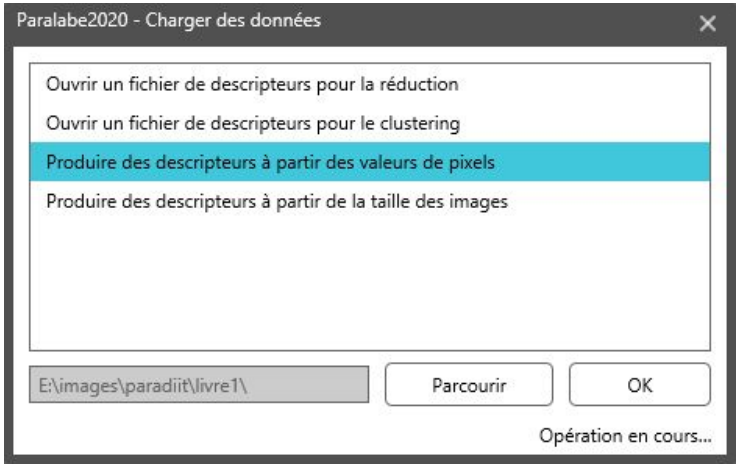
Paralabe2020 : le module d'entrée — F3



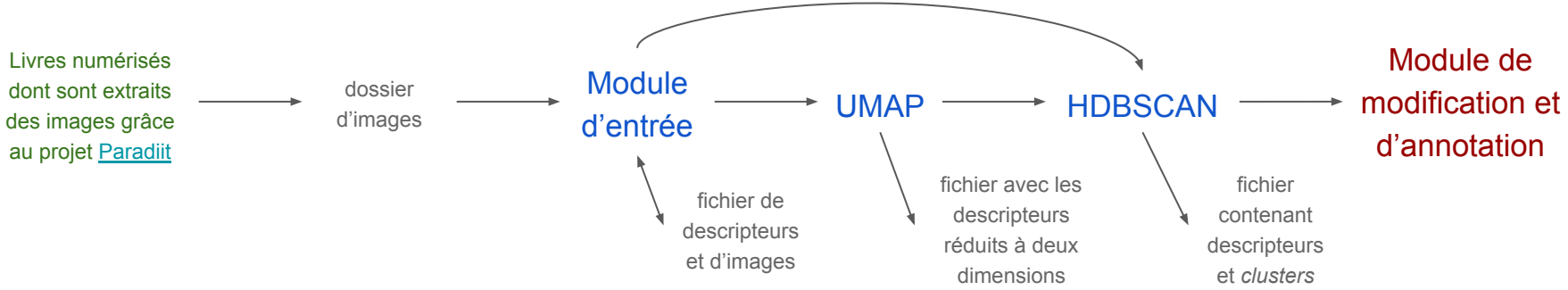
F3 produit des descripteurs à partir de la valeur des pixels de chaque image d'un dossier puis ouvre le second module.

- **Entrée** : dossier contenant des images au format BMP en niveaux de gris
- **Autres paramètres** : aucun
- **Sortie** : fichier data.json contenant les descripteurs
- **Erreur** : si le dossier ne contient pas d'images conforme au format, on ne passe pas au module suivant et l'erreur s'affiche en bas.

F3 s'inspire du script paradiit_to_json.py de Yann Galan. Elle devra être testée avec le dossier d'images venant de Paradiit déjà disponible dans le projet Cartolabe.



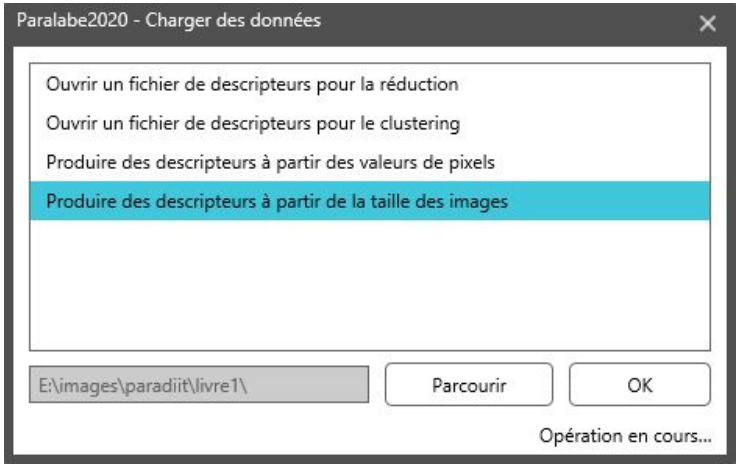
Paralabe2020 : le module d'entrée — F4



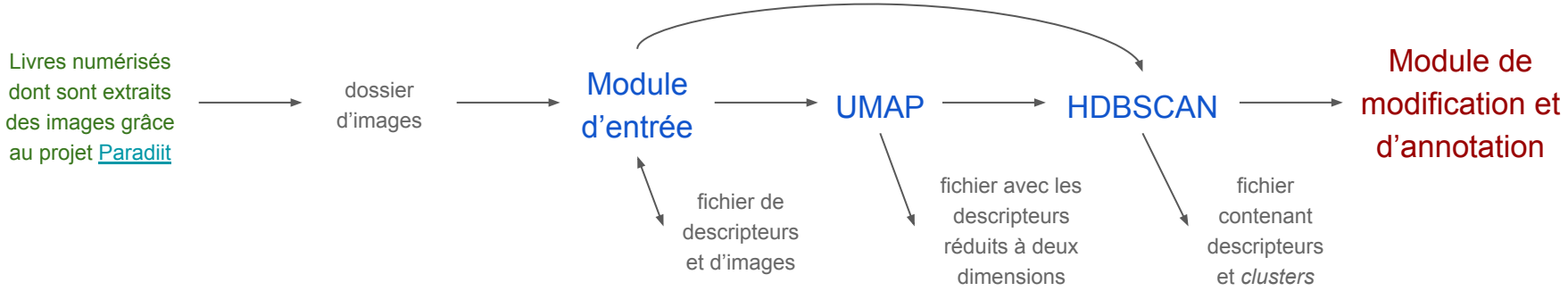
F4 produit des descripteurs à partir des dimensions de chaque image d'un dossier puis ouvre le second module.

- **Entrée** : dossier contenant des images
- **Autres paramètres** : aucun
- **Sortie** : fichier data.json contenant les descripteurs
- **Erreur** : si le dossier ne contient pas d'images, on ne passe pas au module suivant et l'erreur s'affiche en bas.

F4 sert principalement à montrer comment d'autres classes filles d'Extractor peuvent augmenter les possibilités d'entrées mais a peu d'utilité pratique puisque les descripteurs choisis ont peu de sens. Elle devra être testée avec un jeu de centaines d'images de taille différentes au contenu indifférent.



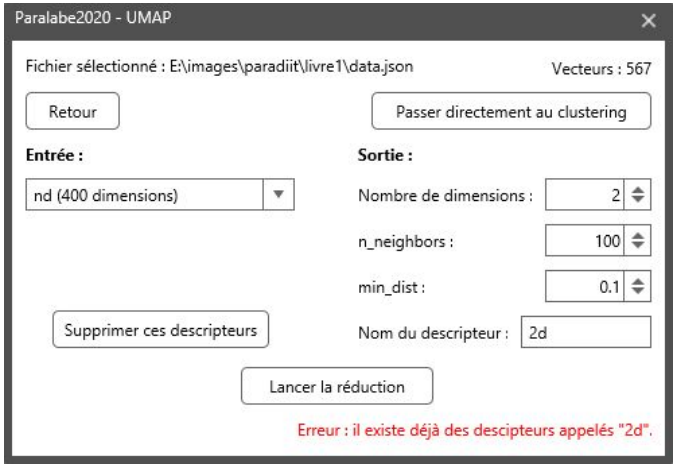
Paralabe2020 : UMAP



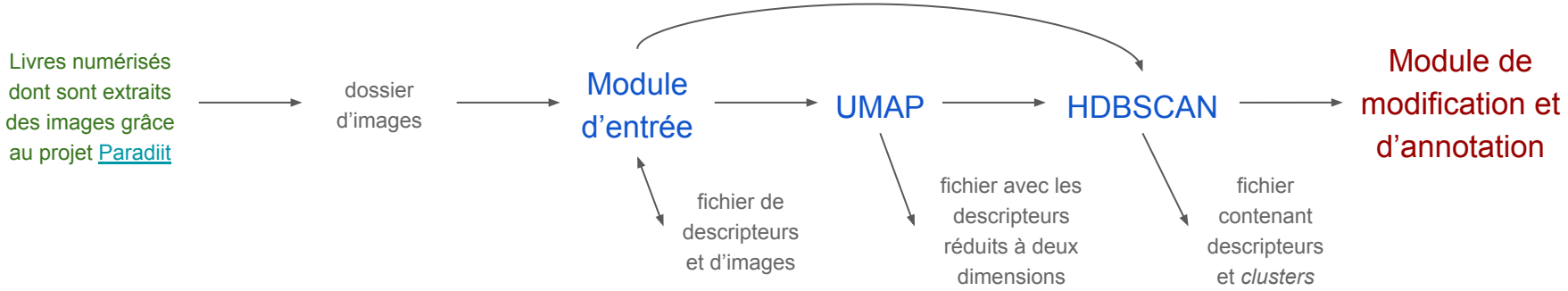
Le module UMAP permet d'ajouter des descripteurs aux vecteurs à partir d'autres descripteurs dont la dimension est plus grande par l'algorithme de réduction UMAP (fonction **F5**), puis de passer au *clustering*.

- **Entrée** : un fichier de descripteurs sélectionné au module précédent
- **Autres paramètres** : le choix des descripteurs, le nombre de dimensions, n_neighbors et min_dist qui servent à UMAP, et le nom du nouveau descripteur.
- **Sortie** : le fichier d'entrée gagne un descripteur. Aucun nouveau fichier n'est produit mais aucune donnée n'est écrasée.
- **Erreur** : si l'un des paramètres n'est pas valide (valeurs négatives, par exemple) ou que le nom qu'on souhaite donner au nouveau descripteur est déjà pris, on ne passe pas au *clustering* et l'erreur s'affiche en bas.

F5 s'inspire du script paralabe_clustering.py de Yann Galan.

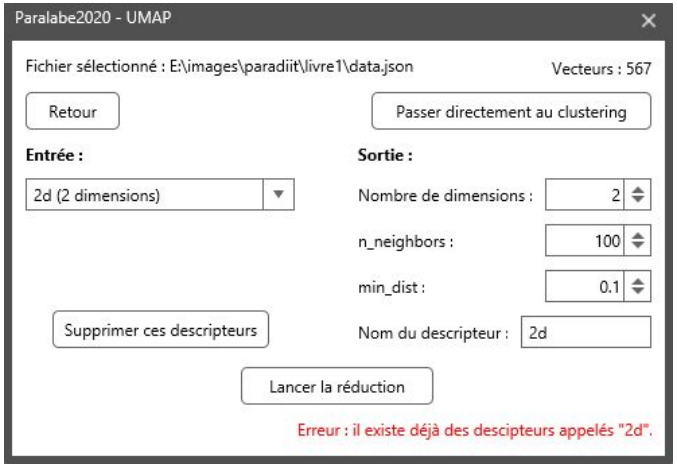


Paralabe2020 : UMAP

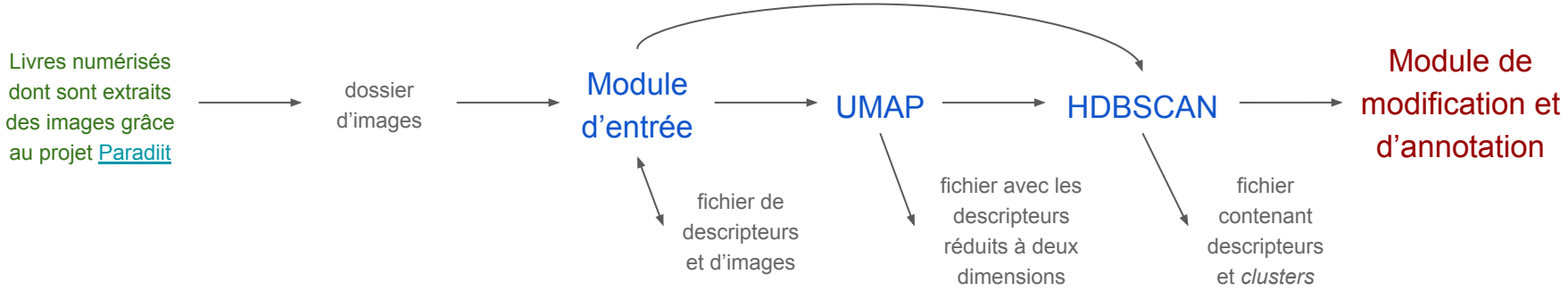


L'IHM du module UMAP permet par ailleurs de retourner au module précédent, de passer au module suivant sans réduction, et de supprimer des descripteurs (fonction **F6**), avec pop-up de confirmation, notamment dans le cas où une utilisation précédente d'UMAP aurait été insatisfaisante.

- **Entrée** : un fichier de descripteurs sélectionné au module précédent
- **Autres paramètres** : le nom du descripteur à supprimer.
- **Sortie** : le fichier perd un descripteur.



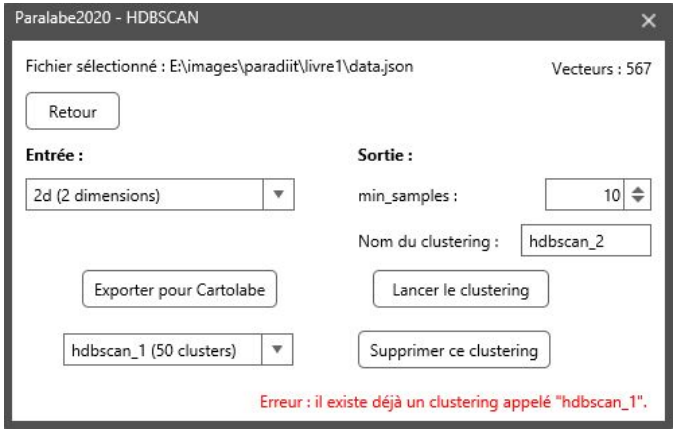
Paralabe2020 : HDBSCAN



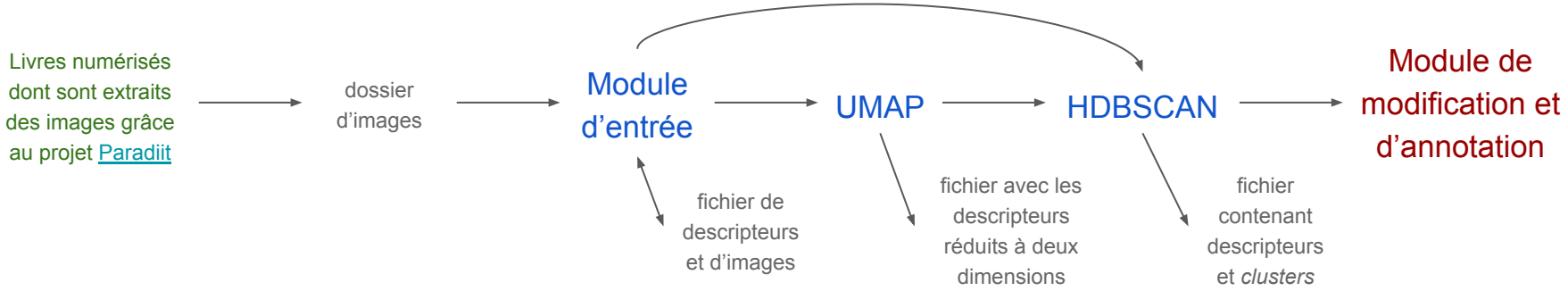
Le module HDBSCAN permet de former des *clusters* à partir de descripteurs d'un fichier de vecteurs (fonctions **F7**).

- **Entrée** : toujours le même fichier de descripteurs
- **Autres paramètres** : le choix des descripteurs, min_sample utile à HDBSCAN et le nom du *clustering*.
- **Sortie** : le fichier d'entrée gagne un *clustering*. Aucun nouveau fichier n'est produit mais aucune donnée n'est écrasée.
- **Erreur** : si l'un des paramètres n'est pas valide (valeurs négatives, par exemple) ou que le nom qu'on souhaite donner au *clustering* est déjà pris, l'erreur s'affiche en bas.

F7 s'inspire du script paralabe_clustering.py de Yann Galan.

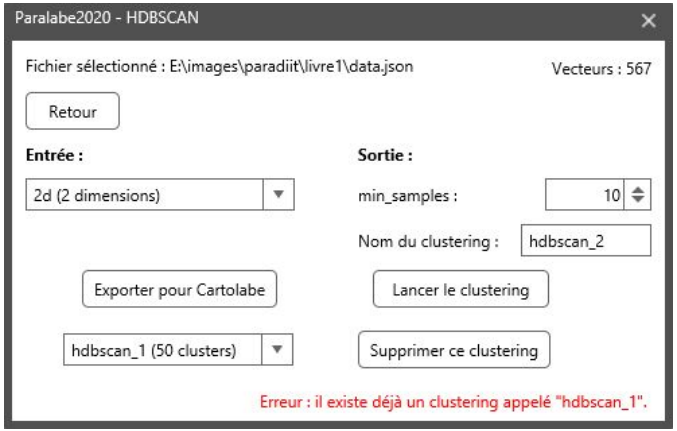


Paralabe2020 : HDBSCAN

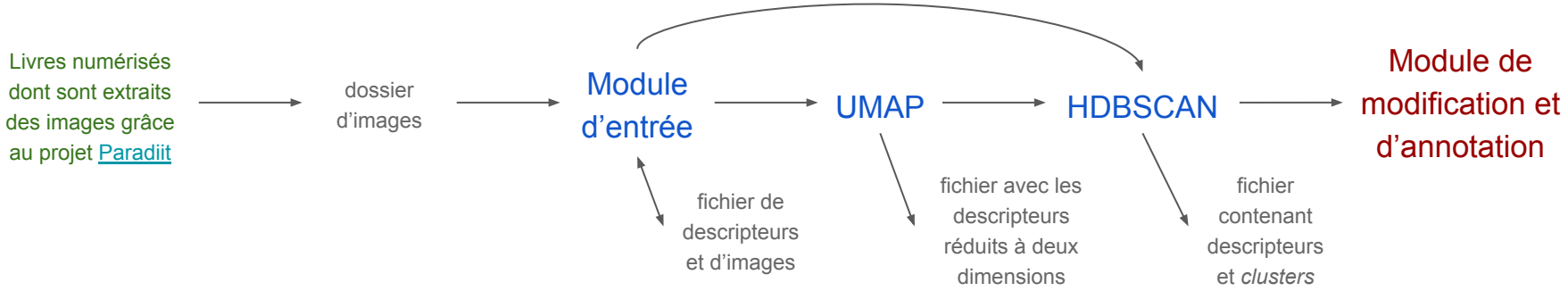


L'IHM du module HDBSCAN permet par ailleurs de retourner au module précédent, et de supprimer un *clustering* (fonction **F8**), avec pop-up de confirmation, notamment dans le cas où une utilisation précédente de HDBSCAN aurait été insatisfaisante.

- **Entrée** : toujours le même fichier de descripteurs
- **Autres paramètres** : le nom du descripteur à supprimer
- **Sortie** : le fichier perd un descripteur.

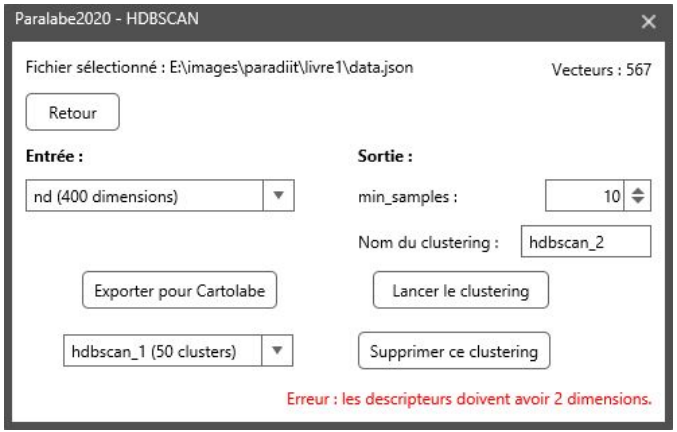


Paralabe2020 : HDBSCAN

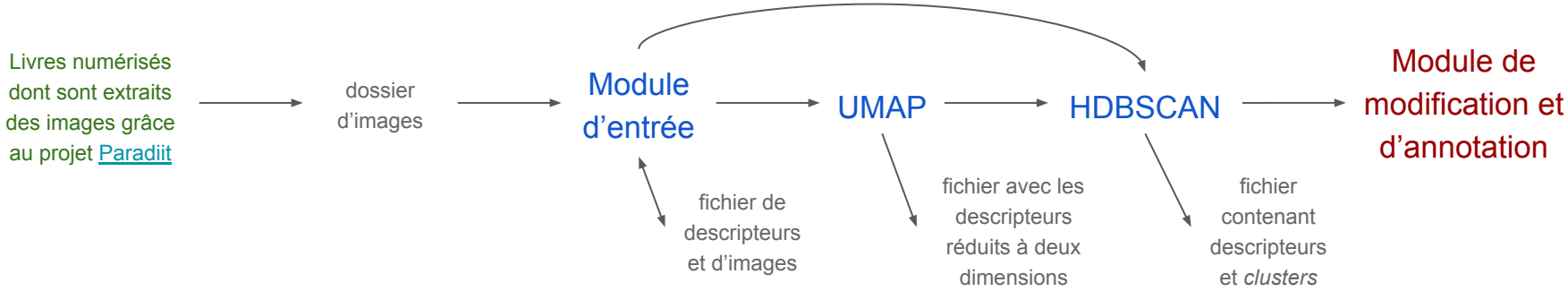


Enfin, le module HDBSCAN permet au moins d'exporter des descripteurs à 2 dimensions au format compatible avec Cartolabe (fonction **F9**), qui est un peu différent de celui utilisé par Parabale2020 (voir diapositives suivantes) et qui ne prend pas le *clustering* en compte.

- **Entrée** : toujours le même fichier de descripteurs
- **Autres paramètres** : le nom des descripteurs à prendre en compte
- **Sortie** : un fichier export.json
- **Erreur** : si les descripteurs n'ont pas 2 dimensions, l'erreur s'affiche en bas et aucun fichier n'est produit.



Cartolabe et le module de modification et d'annotation



Le module de modification et d'annotation pourrait être réalisé de deux manières :

- Paralabe2020 pourrait produire comme sortie finale les vecteurs et leurs clusters, dans un format compatible avec Cartolabe, qui serait à définir. La possibilité de fusionner, diviser, modifier et annoter les clusters seraient alors ajoutées à Cartolabe.
- Paralabe2020 pourrait utiliser et enrichir une partie du code de Cartolabe pour permettre l'affichage de la cartes des données, avec les clusters, la possibilité de les modifier, de les annoter, etc. dans une dernière IHM.

Pour l'instant, l'obstacle principal à ces deux options est que Cartolabe, étant avant tout un outil d'affichage, ne traite pas les clusters comme des ensembles de points mais comme des points eux-mêmes qui s'affichent à certains niveaux de zoom au-dessus d'une région. Plus qu'un ajout de fonctionnalité, cette évolution représenterait un changement de paradigme.

Cartolabe et le module de modification et d’annotation

Cartolabe permet d’afficher des tuiles qui sont produites à partir d’ElascticSearch, qui accepte lui-même des fichiers JSON où chaque point est au format suivant, d’après le Wiki de Cartolabe :


```
{
    "label": "Theoretical and computational studies of the diversity
management problem",
    "nature": "articles",
    "rank": 8,
    "score": 1.0,
    "position": [4.14397668838501, 2.117488145828247],
    "year": "2000",
    "url": "tel-00004710",
    "neighbors":
        {"articles": [8, 40736, 52728, 26428, 50417, 45257, 48433],
        "authors": [67264, 79350, 65196, 66678, 66332, 70104, 76775],
        "words": [98419, 90097, 93525, 88289, 90067, 89153, 90100],
        "teams": [101499, 101375, 101430, 101507, 101313, 101310, 101371],
        "labs": [102697, 104736, 102963, 102729, 103817, 104484]
        }
}
```


“The fields on a Point are

- position - the 2D coordinates of the point on the map
- score - a float representing the point's interest value
- rank - the point's index in the list
- nature - a string that correspond to a `Nature`'s key on the Dataset.
- label - the point's title
- url - a string that represents a relative or absolute url for this point
- neighbors - a list of `Neighbors`”

Structure du projet :

- Package extraction
  - extractor.py qui contient la classe abstraite Extractor
  - extractor_pixels.py (**F3**)
  - extractor_size.py (**F4**)
- Package IHM
  - module1.py
  - module2.py
  - module3.py
- Package fonctions
  - umap.py (**F5**)
  - hdbscan.py (**F7**)
  - autres.py (**F1**, **F2**, **F6**, **F8**, **F9**)
- main.py qui ouvre la premère fenêtre

Planning :

2 mars : livraison des IHM et de F1 et F2 qui permettent de charger des fichiers

6 mars : livraison de F5 (UMAP) et F7 (HDBSCAN)

13 mars : livraison de la classe Extractor et de ses deux filles F3 et F4

20 mars : livraison de F6, F8 et F9

27 mars : livraison finale avec rapport