Application of DeepCellMap to a new Dataset

🌟 Goal : use DeepCellMap on “NewDataset”

▼ Overview

DeepCellMap is able to deal with immunohistochemistry data as well as fluorescence data.

Since these two modalities are somehow different in the way they are encoded, the configuration files could differ a bit but DeepCellMap work the same for both kind of data.

This guide has been produce to help you analyse your data with DeepCellMap. You just have to follow the following steps.

▼ 1. Creation of the config file

All the details about the dataset should be written in this file (and only in this file). This will drive and adjust the behavior of DeepCellMap.

- If the dataset is Fluorescence data :

  → Copy/past file `DeepCellMap/code/config/dataset_new_fluorescence.py` and rename it;

- If the dataset is Immunohistochemistry data

  → Copy/past file `DeepCellMap/code/config/dataset_new_ihc.py` and rename it;

- Change Classname `FluorescenceNewConfig` to `FluorescenceDatasetNameConfig`

Check the items one by one and use the page 🔗 ConfigFile explanation (1 dataset = 1 Config file) to help you choose the good values.

▼ 2. Ensure location and image names are good

- All the images should be saved in `DeepCellMap/data/NewDataset`
- Each instance should be renamed so that all images are called `00n.tif` or `00n.svs` with n the numero of the sample. The best way to conserve the previous names is have a svs file or to rename images one by one and copy/past the real name to the list `NewDatasetConfig.mapping_img_name`

▼ 3. Images preprocessing - Notebook 1 Image processing

Set `image_list = [1,2,]` with the image numbers you want to process

▼ 1. Apply get_info_data

Running `slide.get_info_data(dataset_config, image_list=image_list)` will help to set the attributes of the config file. (particularly considering the values of tile width and height and downscaling factor. If the image size is > 20.000 pixels, use at least scale_factor = 16)

▼ 2. Image downscaling

Run `slide.training_slide_range_to_images(dataset_config, image_list=image_list)` to downscale images by factor `scale_factor` (config file)

▼ 3. Tissue extraction

If images have not been processed already, the default filter may not be the good one. The `Notebook 2 - Laboratory tissue segmentation` will help to find the good steps either in the collection of tissue-filters or by constructing new filter.

Run `segmentation.multiprocess_apply_filters_to_images( save =True, display =False, html =False, image_num_list =image_list, dataset_config =dataset_config)` to create downscaled images with tissue extracted. This will be helpfull to save computation time (because next computations will be done only on tiles containing enough tissue)

▼ 4. Slide tilling

Original images are far too large to be computed by even simple algorithms. To keep maximum informations during computation, images are tiled in little squares of size `tile_height` x `tile_width` .

Running `segmentation.multiprocess_apply_filters_to_images( save =True, display =False, html =False, image_num_list =image_list, dataset_config =dataset_config)` will tile the image and create system of coordinates to be able to refer to each little square. A HTML file is generated containing an image per line and columns contain downscaled original image, downscaled original image with tissue extracted and downscaled original tiled images with the possibility to visualise some randomly chosen tiles in the image.

🌟 **Once** images are tiles, you can select tiles from them and set the values of `tile_test_segmentation` so that to set the processing steps for cell segmentation these tiles will be used !

▼ 4. Tissue extraction laboratory - Notebook 2 - laboratory tissue extraction

Different steps can be performed to extract tissue in an image. With IHC data, steps are applied to the entire RGB downscaled image. With fluorescence, an channel can be chosen (with argument `channel_used_to_segment_tissue`) to create the mask of the tissue. Usually, the channel containing the nuclei is used.

▼ 1. Definition of a segmentor

Idea : A sequence of the following `possible image processing steps` is given to the variable `sequence_processing` This describe the steps that are going to be performed on the downscaled original image.

▼ Library of the `possible image processing steps`

```
"cellpose" : param_best_cellpose,

#Binarization steps
"multi_otsu_thresholding" :  ["dont_consider_background"
"otsu_thresholding" : None,
"manual_threshold" : 10,
"rgb_threshold_based_binarization_microglia_IHC": None,

#Morphological operations
"erosion": 2,
"dilation" : 4,
"dilation2":3,
"opening": 5,

#Other operations
"fill_holes" : None,
"filter_center_cells" : dataset_config.tile_width,
"remove_small_objects" : 150,
```

```
 "remove_large_objects" : 10000,
 "rgb_to_eosin" : None,
 "take_largest_component" : 1,
```

```
# Exemple of sequence of steps to extract tissue
sequence_processing = dict({
    "manual_threshold" : 10,
    "dilation2" : 30,
    "fill_holes" : None,
    "remove_small_objects" : 20000,
    "take_largest_component" : 1})
```

A tissue segmentor is defined as follows :

```
#Define segmentor
segmentor = segmentation.ModelSegmentation(dataset_config=d
```

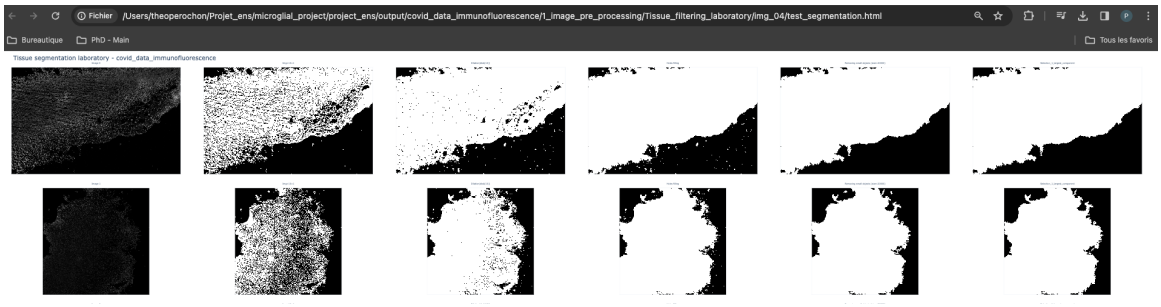
▼ 2. Segmentation of the tissue and html visualisation of the performed steps

Images used to test the segmentor are selected

```
#Get images to segment
list_images, list_images_names  = slide.get_images_to_segme

#Apply segmentation based on param and save results in html
list_images_processed, list_process_names = segmentor.test_

# segmentor.laboratory_segmentation_save_images(list_images
segmentation.ModelSegmentation.laboratory_segmentation_save
```



▼ 5. Cell segmentation laboratory - Notebook 3 - laboratory cell segmentation

As well as with tissue, different processing steps can be performed on images to segment cells. The input of the cell segmentor is a RGB image with IHC data and a grayscale image when it is fluorescence data since each fluorophore is associated with 1 IxL image.

Note : segmentor = succession of image processing steps whose goal is to segment cells body.

▼ 1. Definition of several tiles to try a segmentor

Tiles should have been specified using the parameter `tile_test_segmentation` as in the following exemple :

```
#ConfigfFile.py class ConfigNewDataset
self.tile_test_segmentation = dict({
            "001": [(18,18), (19,26), (19,10),(15,38), (18,
            "002": [(18,18), (19,26), (19,10),(15,38), (18,
            "003": [(18,18), (19,26), (19,10),(15,38), (18,
        })
```

▼ `NewFeature` - Automatic selection of random tiles

▼ Prompts to create the functions

Can you create a portion of code that will be integrated in my project. The goal of this function is to randomly select some tiles in Whole Slide Image but ensuring that the final corpus covers the heterogeneity of the entire image. I want for example the 10 most different tiles but also représentative (I don't want these tiles to be where there is artefacts).

```
import openslide
import numpy as np
from sklearn.feature_extraction import image
from sklearn.metrics.pairwise import cosine_similarit

def load_whole_slide_image(file_path):
    slide = openslide.OpenSlide(file_path)
    return slide

def calculate_tile_heterogeneity(tile):
    # You can implement your own method to calculate
    # This example uses the standard deviation of pix
    return np.std(tile)

def select_representative_tiles(slide, tile_size=(256
    slide_width, slide_height = slide.dimensions
    tiles = []

    # Extract random tiles from the slide
    while len(tiles) < num_tiles:
        x = np.random.randint(0, slide_width - tile_s
        y = np.random.randint(0, slide_height - tile_
        tile = np.array(slide.read_region((x, y), 0,
        tiles.append(tile)

    # Calculate heterogeneity scores for each tile
    heterogeneity_scores = [calculate_tile_heterogene

    # Find the most representative tiles based on het
```

```
    most_representative_indices = np.argsort(heteroge
    most_representative_tiles = [tiles[i] for i in mo

    return most_representative_tiles

# Example usage:
file_path = "path/to/your/whole_slide_image.svs"
slide = load_whole_slide_image(file_path)
representative_tiles = select_representative_tiles(sl
```

The code you provided is good. But the notion of heterogene tiles is not understood. What I want is to have 10 tiles that represent the heterogeneity of the entire image. The final goal is generate a function that will segment cells on these tiles but to be robust, this function should be found based on its skills on 10 different and representative tiles. Is it clear ?

```
import openslide
import numpy as np
from sklearn.feature_extraction import image
from sklearn.cluster import KMeans

def load_whole_slide_image(file_path):
    slide = openslide.OpenSlide(file_path)
    return slide

def select_representative_tiles(slide, tile_size=(256
    slide_width, slide_height = slide.dimensions
    num_tiles_x = slide_width // tile_size[0]
    num_tiles_y = slide_height // tile_size[1]

    # Extract patches from the slide
    patches = []
    for y in range(num_tiles_y):
        for x in range(num_tiles_x):
            patch = np.array(slide.read_region((x * t
```

```
            patches.append(patch)

    # Reshape patches for clustering
    patches = np.array(patches)
    reshaped_patches = patches.reshape(-1, tile_size[

    # Apply KMeans clustering
    kmeans = KMeans(n_clusters=num_clusters, random_s
    kmeans.fit(reshaped_patches)

    # Get centroids of clusters as representative til
    representative_indices = kmeans.cluster_centers_.
    representative_tiles = [patches[idx] for idx in r

    return representative_tiles

# Example usage:
file_path = "path/to/your/whole_slide_image.svs"
slide = load_whole_slide_image(file_path)
representative_tiles = select_representative_tiles(sl
```

With visualisation

```
import openslide
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction import image
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

def load_whole_slide_image(file_path):
    slide = openslide.OpenSlide(file_path)
    return slide

def visualize_clusters(slide, tile_size=(256, 256), n
    slide_width, slide_height = slide.dimensions
```

```
    num_tiles_x = slide_width // tile_size[0]
    num_tiles_y = slide_height // tile_size[1]

    # Extract patches from the slide
    patches = []
    for y in range(num_tiles_y):
        for x in range(num_tiles_x):
            patch = np.array(slide.read_region((x * t
            patches.append(patch)

    # Reshape patches for clustering
    patches = np.array(patches)
    reshaped_patches = patches.reshape(-1, tile_size[

    # Apply KMeans clustering
    kmeans = KMeans(n_clusters=num_clusters, random_s
    kmeans.fit(reshaped_patches)

    # Use PCA to reduce dimensions for visualization
    pca = PCA(n_components=2)
    reduced_data = pca.fit_transform(reshaped_patches
    reduced_centroids = pca.transform(kmeans.cluster_

    # Plot clusters and centroids in 2D
    plt.figure(figsize=(8, 6))
    plt.scatter(reduced_data[:, 0], reduced_data[:, 1
    plt.scatter(reduced_centroids[:, 0], reduced_cent
    plt.title('2D Visualization of Clusters with Cent
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.colorbar()
    plt.show()

# Example usage:
file_path = "path/to/your/whole_slide_image.svs"
```

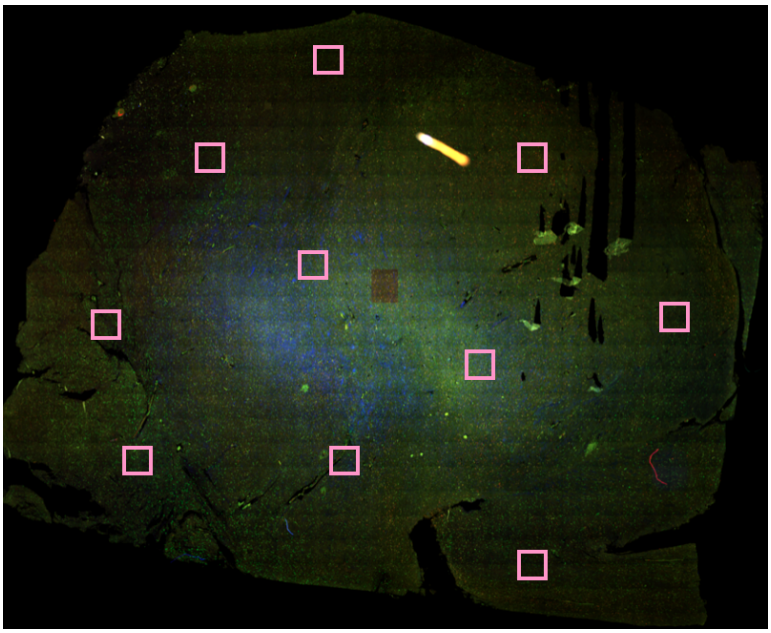
```
slide = load_whole_slide_image(file_path)
visualize_clusters(slide)
```

▼ Fluorescence images

```
# Which image to use
image_list = [1]
# Number of tiles to test
n_tiles_per_channel = 10
# Channel to test
channels_to_segment = [0]

tiles_list, tiles_names  = tiles.get_tiles_to_segment(da
```

▼ IHC images



▼ 2. Definition of a cell segmentor

▼ Sequencing steps with fluorescence images

The dictionary containing the different processing steps is described as follows

```
cell_segmentation_param = dict({
    0: {
        "multi_otsu_thresholding" :  ["dont_consider_bac
        "fill_holes" : None,
        "remove_small_objects" : 200,
        "remove_large_objects" : 20000
    },
    1: {
        "otsu_thresholding" :  [None,-1],
        },
    2: {
        "test_cas1__cas_2" :  {"dilation_test":2, "thres
        "fill_holes" : None,
        "remove_small_objects" : 200,
        "remove_large_objects" : 10000
        },
    3: {
        "otsu_thresholding" :  [None,70],
        "fill_holes" : None,
        "remove_small_objects" : 200,
        "remove_large_objects" : 20000

        }
    })
```

`0` , `1` , `2` and `3` are the different channels.

▼ Sequencing steps with IHC images

```
cell_segmentation_param = dict(
    {
        "rgb_to_eosin" : "mean_channels",
        "otsu_thresholding" : [None,-1],
        "opening" : 1,
        "dilation" : 2,
        "fill_holes" : None,
        "remove_small_objects": 400,
    }
)
```

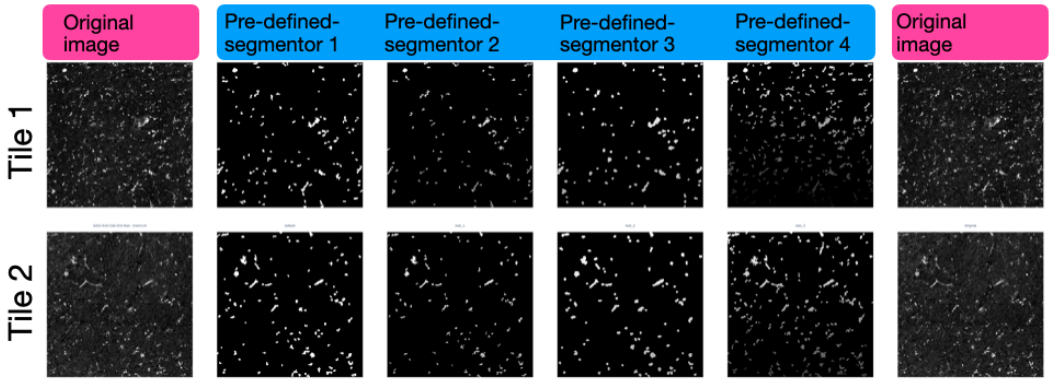
When segmentor have been find for several images, the function `TODO` can be used to test all the segmentor on a new image. It is very probable that a new image has the same properties than a previously processed image.

▼ 3. Test predefined segmentor to select a starting segmentor

Some pre-defined segmentor exists and can be tried on the new images. They are automatically defined by `cell_segmentation_param_by_cannnel` (fluorescence) or `cell_segmentation_param` (ihc) in the `ConfigFile` . This dictionary is also enriched with the segmentors found for the previously processed images.

- Running `segmentation.test_existing_segmentation_config(dataset_config,tiles_list, tiles_names,channels_to_segment, image_list =image_list)` will test all the predefined segmentors on the defined tiles. All the results are presented in an html file like in example image.

Once a pre-defined segmentor is considered good, the processing steps can be copy-paste from the `ConfigFile` to the notebook.



▼ 4. Test cell-segmentor on different tiles

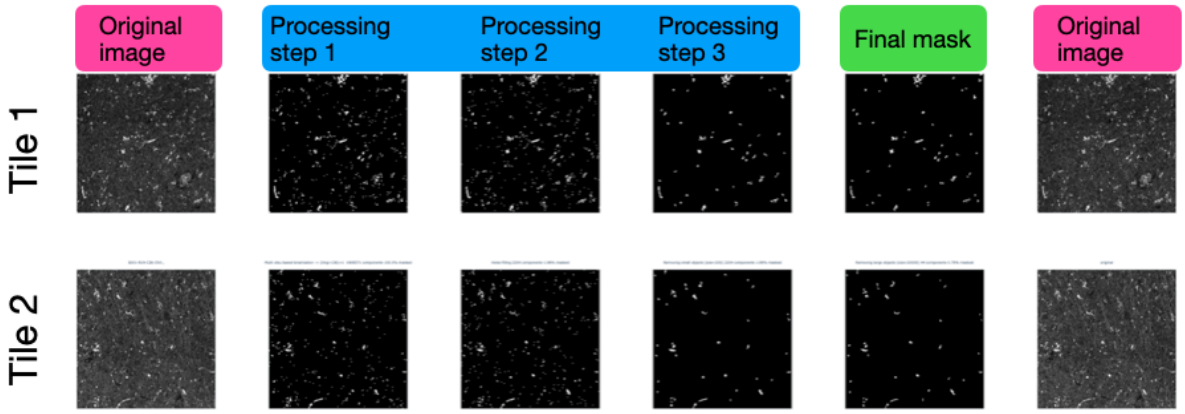
The following portion of code enable to

```
#Find tiles to test cell segmentation
tiles_list, tiles_names  = tiles.get_tiles_to_segment(datas
#Perform segmentation
list_images_processed, list_process_names = cell_segmentor.
#Save different steps of cell segmentation for displaying
segmentation.ModelSegmentation.laboratory_segmentation_save
```

Visualisation of the results and segmentor adjustments

Performed processing steps can be visualised in the generated html file that looks like the following image.

- **If the penultimate column is correct :** the processing steps can be copy/past to the ConfigFile in the `cell_segmentation_param_by_cannnel` with a new key
- **If it is incorrect :** `cell_segmentation_param` can be modified.



Once a good segmentor have been find. The corresponding dictionnary is added to `cell_segmentation_param_by_cannnel[image_n]` (Fluorescence) or `cell_segmentation_param` (Brightfield).

▼ 6. Cell classification on the entire image - Notebook 4 - Cell classification

Select a `dataframe_name` and `image_list` .

Process the entire image to segment and classify cells

Run `table_cells = classification.segment_classify_cells(1, dataset_config, particular_roi =particular_roi, channels_cells_to_segment = channels_cells_to_segment)`

Outputs :

- Masks of all the cells are saved so that the mask of any ROI can be reconstructed
- Dataframe `tiles_and_tissue_percentage.csv` contains the tiles and their percentage of tissue
- Dataframe `slide_001_cells.csv` containing information on all the cells.

▼ If Fluorescence Data

With fluorescence Data, some cell types result from the combination of several channels. These cells should be defined in `cells_from_multiple_channels` like in the following example where iba1_tnem119 is the combination of positive cells on channel 1 and channel 2.

```
self.cells_from_multiple_channels = dict({
            "iba1_tnem119" : [1,2]
        })
```

To create these celltypes, `table_cells = classification.segment_cells_coming_from_several_channels(1, dataset_config, particular_roi=None, cells_from_multiple_channels = None)` is needed.

Save results in a dataframe

`df = slide.get_statistics_images( dataset_config =dataset_config, image_list = image_list)` create a dataframe (or add a line) to the dataframe containing main information on each image like cell_numbers etc.

Display first results on cell classification

- **Histogram cell proportion:** `classification.histogramm_cell_number( dataset_config , slide_num )`
- **Heatmap creation :** `classification.generate_heatmap_cells( dataset_config , slide_num )`

▼ 7. Region of Interest creation and DeepCellMap application on a Region of

Interest - Notebook 6 - DCM application on a ROI

The goal of this notebook is to run DeepCellMap on any Region of Interest in the image. A Region of interest is defined with the coordinates of the top-left and the bottom-right tiles.

The applicaiton of DeepCellMap consist on

1. Colocalisation analysis
2. DBSCAN analysis
3. Neighbors analysis

▼ 1. ROI reconstruction

Definition of the slide and ROI coordinates

```
slide_num= 1
entire_image = False
if entire_image :
    origin_row, origin_col, end_row, end_col = None, None,N
else :
    origin_row, origin_col, end_row, end_col = 12,18,13,19
```

Creation of the ROI

```
# Creation of the ROI
roi = Roi(dataset_config,slide_num, origin_row, origin_col,

#Get tissue mask
roi.get_mask_tissue()

#Visualisation RGB et channels
roi.get_img_original(channels_of_interest = "all_as_rgb")
# roi.display_save_img_original(save = True)
```

```
# # #Get table cells
roi.get_table_cells(filter_cells_from_several_channels=Fals

# # #Get mask regions
roi.get_cell_masks()

# # #get predictions and masks
roi.display_segmented_cells(save_fig = True)

# # #get roi statistics
roi.get_statistics_roi(save = True)
```

▼ TODO fluo

display_mask(roi.image_w_borders[:,:,0], *title* = " TNEM119", *figsize* = (30,30))

TODO : iamge ROI cells

▼ 2.DeepCellMap application

Colocalisation

Run `coloc_analysis = ColocAnalysis(roi)`

▼ to display levelsets

Run

```
display_B_in_A_levelsets = False
if display_B_in_A_levelsets :
    coloc_analysis.display_B_in_A_levelsets(save_fig = T

coloc_analysis.visualise_z_score()
```

```
coloc_analysis.heatmap_colocalisation(feature_name= "ass
coloc_analysis.heatmap_colocalisation(feature_name= "dis
```

DBSCAN analysis

Run `dbscan_analysis = DbscanAnalysis(roi, min_sample =4)`

▼ To display clusters

```
if display_figure_clusters_convex_hulls :
    dbscan_analysis.display_convex_hull(robust_clusters_
```

Neighbors analysis

Run `neighbors_analysis = NeighborsAnalysis(roi)`

Merge all results

Run `results_entire_image = roi.get_df_entire_image()`

TODO : image