

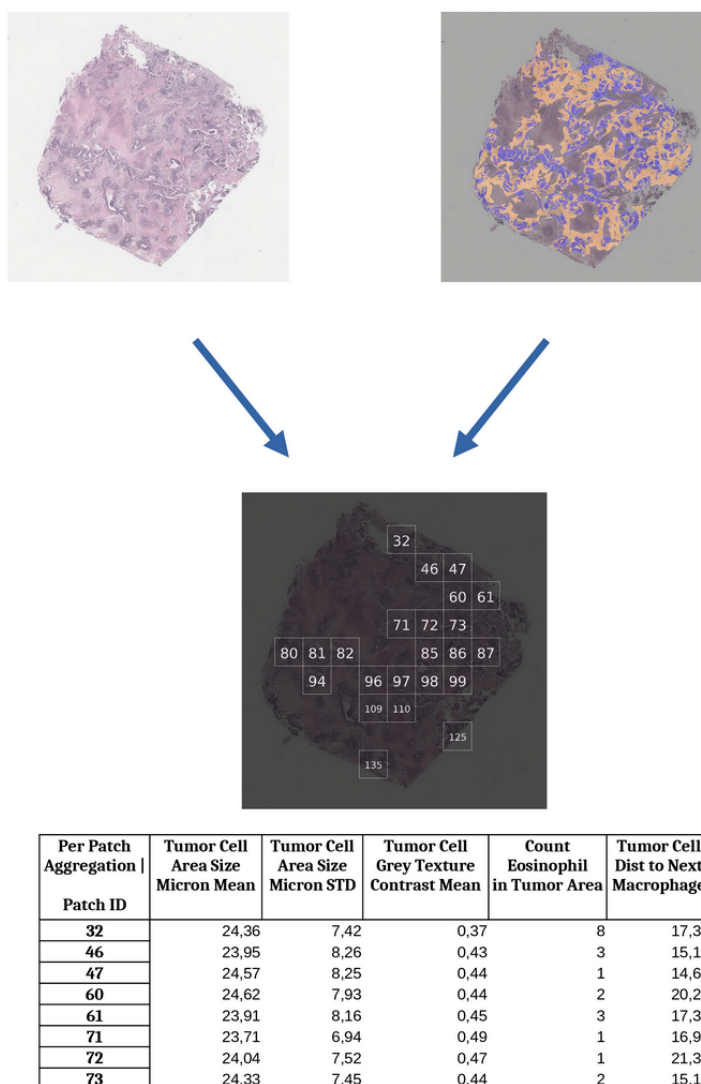
Code Description (whole slide image analysis using ready SPARK parameters)

This is a modular pipeline for evaluation of digitized whole slide images (WSI) at different resolutions. Modern segmentation algorithms allow for a detailed and automated annotation of tissue-areas and single cells. However, for a downstream analysis, this information typically must be encoded into a set of numerical features. This pipeline computes and organizes various features at cell-resolution (e.g. cell size, texture heterogeneity and MANY more), and then aggregates/averages them at a chosen zoom-level (e.g. mean size of tumor cells in local patches/tiles of side-length 1mm, number of lymphocytes per patch, and many more).

INPUTS:

1. whole-slide image that can be opened via openslide (WSI)
2. geojson of single cell nuclei polygons - ideally with celltype label (e.g. from CellVit, HoVer-NeXt, StarDist or similar)
3. tissue-segmentation mask (low-resolution greyscale png image that encodes tissues/regions of interest, quality control etc.)

OUTPUT: Evaluation of single-cell-features and statistics at 1) chosen zoom and 2) for relevant patches.



Example evaluation for publicly available TCGA LUAD 64-5774. Given a WSI and a tissue-mask (here: tumor in blue, and surrounding tumor-stroma in orange for visualization), we choose to

evaluate patches of side-length 1mm, and only patches with at least 30% tumor area are taken into account. After computing the single-cell features in an efficient way, we store per-patch summarizing statistics for downstream analysis. Note that if necessary (for example if the cell neighborhood is a needed information), all computations are done GLOBALLY, and no information is lost due patch-sampling. The patches are only used to organize the final output.

****Inputs****

Per default, the mask image should be a greyscale png, and a 1:1 downsampled version of the WSI. Assure that no padding or cropping prevents a 1:1 downsampling. Typically, 0 encodes the "Background" (areas that should be ignored in any analysis), all other integers encode a certain tissue-labeling. In our example, we encode the different tissues via `tissue_map = {"Background":0, "Tumor":1, "Stroma":2}`. The geojson is supposed to be a json, in format `{"type": "FeatureCollection", "features": [{"type": "Feature", "geometry": {"type": "Polygon", "coordinates": [[[72076.0, 55764.0], [72068.0, 55772.0], [72064.0, 55780.0], [72064.0, 55788.0], [72076.0, 55800.0], [72088.0, 55804.0], [72096.0, 55800.0], [72100.0, 55792.0], [72100.0, 55776.0], [72096.0, 55768.0], [72088.0, 55764.0], [72076.0, 55764.0]]]}, "properties": {"classification": "Plasma"}}]}`, where "features" is a list of cell objects as given in the example. As long as celltype and polygon-boundary are available, the input-function `WSI_Evaluation.process_geojson_single_cells` can easily be modified accordingly. As with the mask, the (optional) thumbnail must be a 1:1 downsampled version of the WSI, but can have a different resolution.

If the single cell labels were generated with [HoVer-NeXt](https://github.com/digitalpathologybern/hover_next_inference), we provide a ready-to-use cleaning and preprocessing pipeline in the folder `Hovernext_Cleaning_Preprocessing`. This is a lightweight version of the pipeline that is supposed to run fast and with few requirements, it needs a different conda environment (`CONDA_ENV_cleaning_minimal.yaml`). The provided script `"MAIN_PostProcessing_Geojson.py"` optimizes the single cell labels under the assumption that the tissue segmentation mask is in general more reliable than the single cell classification, and removes cells that are in the background/unlabeled area of the image mask. The single cell geojson format and the mask-json (optional output for debugging) can both be visualized with [QuPath](<https://qupath.github.io/>).

****AI-written features (SPARK)****

In the folder `Sparks_AI_Written_Features/WSI_Eval_Pipeline`, you find a ready-to-use pipeline that computes a large selection of AI-written features, as part of the publication of our SPARKS system. Again, this version works in a different conda environment. The patches are hard-coded to have a side-length of 1mm, some rare features also use a patch side-length of 5mm. Originally, the pipeline was designed to be run only on WSIs with `mpp=0.25` (these functions are stored in `AI_FEATURE_FUNCTION_DIR/function_dir_old_mpp_025_fixed`). However, we also provide a modified version, with features adjusted to scale with the resolution of the WSI (`AI_FEATURE_FUNCTION_DIR/function_dir_new_mpp_scale`), thank you Pita for going through this with me :) ****WARNING****: Mixing results from WSIs from different cohorts can still cause problems, due to staining variation and since some more complex functions don't scale well with the mpp (for example image smoothing, box counting over different resolutions, area padding etc.). However, we found that many features that yield a biological signal (mutation/survival) are also those that have robust code structure. Note that the AI-written features are extremely CPU-demanding and not optimized, and a large instance can take up to 100GB RAM (for ~600.000 cells) and many hours of computation time.

Please check the

`"Sparks_AI_Written_Features/WSI_Eval_Pipeline/MAIN_FUNCTION_CALLER.py"` for an example evaluation workflow. By providing a reduced `function_library.json` (default: `function_library_475.json` containing ALL features), you can sub-select which feature-functions to run. There is an alternative

set of features for selection, coded by the LLM agent based on human ideas, check
AI_FEATURE_FUNCTION_DIR/ALTERNATIVE_library_of_human_features.