
Core API

SlideData

The central class in `PathML` for representing a whole-slide image.

class pathml.core.SlideData(*filepath*, *name=None*, *masks=None*, *tiles=None*, *labels=None*, *backend=None*, *slide_type=None*, *stain=None*, *platform=None*, *tma=None*, *rgb=None*, *volumetric=None*, *time_series=None*, *counts=None*, *dtype=None*)

Main class representing a slide and its annotations.

Parameters:

- **filepath** (*str*) – Path to file on disk.
- **name** (*str, optional*) – name of slide. If `None`, and a `filepath` is provided, name defaults to filepath.
- **masks** (*pathml.core.Masks, optional*) – object containing {key, mask} pairs
- **tiles** (*pathml.core.Tiles, optional*) – object containing {coordinates, tile} pairs
- **labels** (*collections.OrderedDict, optional*) – dictionary containing {key, label} pairs
- **backend** (*str, optional*) – backend to use for interfacing with slide on disk. Must be one of {“OpenSlide”, “BioFormats”, “DICOM”, “h5path”} (case-insensitive). Note that for supported image formats, OpenSlide performance can be significantly better than BioFormats. Consider specifying `backend = "openslide"` when possible. If `None`, and a `filepath` is provided, tries to infer the correct backend from the file extension. Defaults to `None`.
- **slide_type** (*pathml.core.SlideType, optional*) – slide type specification. Must be a `SlideType` object. Alternatively, slide type can be specified by using the parameters `stain`, `tma`, `rgb`, `volumetric`, and `time_series`.
- **stain** (*str, optional*) – Flag indicating type of slide stain. Must be one of [‘HE’, ‘IHC’, ‘Fluor’]. Defaults to `None`. Ignored if `slide_type` is specified.
- **platform** (*str, optional*) – Flag indicating the imaging platform (e.g. CODEX, Vectra, etc.). Defaults to `None`. Ignored if `slide_type` is specified.
- **tma** (*bool, optional*) – Flag indicating whether the image is a tissue microarray (TMA). Defaults to `False`. Ignored if `slide_type` is specified.
- **rgb** (*bool, optional*) – Flag indicating whether the image is in RGB color. Defaults to `None`. Ignored if

`slide_type` is specified.

- **volumetric** (*bool, optional*) – Flag indicating whether the image is volumetric. Defaults to `None`. Ignored if `slide_type` is specified.
- **time_series** (*bool, optional*) – Flag indicating whether the image is a time series. Defaults to `None`. Ignored if `slide_type` is specified.
- **counts** (*anndata.AnnData*) – object containing counts matrix associated with image quantification

property counts🔗

extract_region(*location*, *size*, **args*, ***kwargs*)🔗

Extract a region of the image. This is a convenience method which passes arguments through to the `extract_region()` method of whichever backend is in use. Refer to documentation for each backend.

Parameters:

- **location** (*Tuple[int, int]*) – Location of top-left corner of tile (i, j)
- **size** (*Union[int, Tuple[int, int]]*) – Size of each tile. May be a tuple of (height, width) or a single integer, in which case square tiles of that size are generated.
- ***args** – positional arguments passed through to `extract_region()` method of the backend.
- ****kwargs** – keyword arguments passed through to `extract_region()` method of the backend.

Returns:

image at the specified region

Return type:

np.ndarray

generate_tiles(*shape=3000*, *stride=None*, *pad=False*, ***kwargs*)🔗

Generator over Tile objects containing regions of the image. Calls `generate_tiles()` method of the backend. Tries to add the corresponding slide-level masks to each tile, if possible. Adds slide-level labels to each tile, if possible.

Parameters:

- **shape** (*int or tuple(int)*) – Size of each tile. May be a tuple of (height, width) or a single integer, in which case square tiles of that size are generated. Defaults to 256px.
- **stride** (*int*) – stride between chunks. If `None`, uses `stride = size` for non-overlapping chunks. Defaults to `None`.
- **pad** (*bool*) – How to handle tiles on the edges. If `True`, these edge tiles will be zero-padded and yielded with the other chunks. If `False`, incomplete edge chunks will be ignored. Defaults to `False`.
- ****kwargs** – Other arguments passed through to `generate_tiles()` method of the backend.

Yields:

pathml.core.tile.Tile – Extracted Tile object

plot(*ax=None*)🔗

View a thumbnail of the image, using matplotlib. Not supported by all backends.

Parameters:

ax – matplotlib axis object on which to plot the thumbnail. Optional.

run(*pipeline*, *distributed=True*, *client=None*, *tile_size=256*, *tile_stride=None*, *level=0*, *tile_pad=False*, *overwrite_existing_tiles=False*, *write_dir=None*, ***kwargs*)🔗

Run a preprocessing pipeline on SlideData. Tiles are generated by calling self.generate_tiles() and pipeline is applied to each tile.

Parameters:

- **pipeline** (*pathml.preprocessing.pipeline.Pipeline*) – Preprocessing pipeline.
- **distributed** (*bool*) – Whether to distribute model using client. Defaults to True.

- **client** – dask.distributed client
- **tile_size** (*int, optional*) – Size of each tile. Defaults to 256px
- **tile_stride** (*int, optional*) – Stride between tiles. If `None`, uses `tile_stride = tile_size` for non-overlapping tiles. Defaults to `None`.
- **level** (*int, optional*) – Level to extract tiles from. Defaults to `None`.
- **tile_pad** (*bool*) – How to handle chunks on the edges. If `True`, these edge chunks will be zero-padded symmetrically and yielded with the other chunks. If `False`, incomplete edge chunks will be ignored. Defaults to `False`.
- **overwrite_existing_tiles** (*bool*) – Whether to overwrite existing tiles. If `False`, running a pipeline will fail if `tiles is not None`. Defaults to `False`.
- **write_dir** (*str*) – Path to directory to write the processed slide to. The processed SlideData object will be written to the directory immediately after the pipeline has completed running. The filepath will default to "<write_dir>/<slide.name>.h5path. Defaults to `None`.
- ****kwargs** – Other arguments passed through to `generate_tiles()` method of the backend.

property shape

Convenience method for getting the image shape. Calling `wsi.shape` is equivalent to calling `wsi.slide.get_image_shape()` with default arguments.

Returns:

Shape of image (H, W)

Return type:

Tuple[int, int]

write(*path*)

Write contents to disk in h5path format.

Parameters:

path (*Union[str, bytes, os.PathLike]*) – path to file to be written

Convenience SlideData Classes

class pathml.core.HESlide(**args*, ***kwargs*)

Convenience class to load a SlideData object for H&E slides. Passes through all arguments to `SlideData()`, along with `slide_type = types.HE` flag. Refer to `SlideData` for full documentation.

class pathml.core.VectraSlide(**args*, ***kwargs*)

Convenience class to load a SlideData object for Vectra (Polaris) slides. Passes through all arguments to `SlideData()`, along with `slide_type = types.Vectra` flag and default `backend = "bioformats"`. Refer to `SlideData` for full documentation.

class pathml.core.MultiparametricSlide(**args*, ***kwargs*)

Convenience class to load a SlideData object for multiparametric immunofluorescence slides. Passes through all arguments to `SlideData()`, along with `slide_type = types.IF` flag and default `backend = "bioformats"`. Refer to `SlideData` for full documentation.

class pathml.core.CODEXSlide(**args*, ***kwargs*)

Convenience class to load a SlideData object from Akoya Biosciences CODEX format. Passes through all arguments to `SlideData()`, along with `slide_type = types.CODEX` flag and default `backend = "bioformats"`. Refer to `SlideData` for full documentation.

TODO:

hierarchical biaxial gating (flow-style analysis)

Slide Types

class pathml.core.SlideType(*stain=None*, *platform=None*, *tma=None*, *rgb=None*, *volumetric=None*, *time_series=None*)

SlideType objects define types based on a set of image parameters.

Parameters:

- **stain** (*str, optional*) – One of [‘HE’, ‘IHC’, ‘Fluor’]. Flag indicating type of slide stain. Defaults to None.
- **platform** (*str, optional*) – Flag indicating the imaging platform (e.g. CODEX, Vectra, etc.).
- **tma** (*bool, optional*) – Flag indicating whether the slide is a tissue microarray (TMA). Defaults to False.
- **rgb** (*bool, optional*) – Flag indicating whether image is in RGB color. Defaults to False.
- **volumetric** (*bool, optional*) – Flag indicating whether image is volumetric. Defaults to False.
- **time_series** (*bool, optional*) – Flag indicating whether image is time-series. Defaults to False.

Examples

```
>>> from pathml import SlideType, types
>>> he_type = SlideType(stain = "HE", rgb = True)    # define slide type manually
>>> types.HE == he_type    # can also use pre-made types for convenience
True
```

asdict()🔗

Convert to a dictionary. None values are represented as zeros and empty strings for compatibility with h5py attributes.

If `a` is a SlideType object, then `a == SlideType(**a.asdict())` will be `True`.

We also provide instantiations of common slide types for convenience:

Type	stain	platform	rgb	tma	volumetric	time_series
`pathml.core.types.HE`	‘HE’	None	True	False	False	False
`pathml.core.types.IHC`	‘IHC’	None	True	False	False	False
`pathml.core.types.IF`	‘Fluor’	None	False	False	False	False
`pathml.core.types.CODEX`	‘Fluor’	‘CODEX’	False	False	False	False
`pathml.core.types.Vectra`	‘Fluor’	‘Vectra’	False	False	False	False

Tile🔗

class pathml.core.Tile(*image*, *coords*, *name=None*, *masks=None*, *labels=None*, *counts=None*, *slide_type=None*, *stain=None*, *tma=None*, *rgb=None*, *volumetric=None*, *time_series=None*)🔗

Object representing a tile extracted from an image. Holds the array for the tile, as well as the (i,j) coordinates of the top-left corner of the tile in the original image. The (i,j) coordinate system is based on labelling the top-leftmost pixel as (0, 0)

Parameters:

- **image** (*np.ndarray*) – Image array of tile
- **coords** (*tuple*) – Coordinates of tile relative to the whole-slide image. The (i,j) coordinate system is based on labelling the top-leftmost pixel of the WSI as (0, 0).
- **name** (*str, optional*) – Name of tile
- **masks** (*dict*) – masks belonging to tile. If masks are supplied, all masks must be the same shape as the tile.
- **labels** – labels belonging to tile
- **counts** (*AnnData*) – counts matrix for the tile.
- **slide_type** (*pathml.core.SlideType, optional*) – slide type specification. Must be a `SlideType` object. Alternatively, slide type can be specified by using the parameters `stain`, `tma`, `rgb`, `volumetric`, and `time_series`.

- **stain** (*str, optional*) – Flag indicating type of slide stain. Must be one of ['HE', 'IHC', 'Fluor']. Defaults to `None`. Ignored if `slide_type` is specified.
- **tma** (*bool, optional*) – Flag indicating whether the image is a tissue microarray (TMA). Defaults to `False`. Ignored if `slide_type` is specified.
- **rgb** (*bool, optional*) – Flag indicating whether the image is in RGB color. Defaults to `None`. Ignored if `slide_type` is specified.
- **volumetric** (*bool, optional*) – Flag indicating whether the image is volumetric. Defaults to `None`. Ignored if `slide_type` is specified.
- **time_series** (*bool, optional*) – Flag indicating whether the image is a time series. Defaults to `None`. Ignored if `slide_type` is specified.

plot(*ax=None*)🔗

View the tile image, using matplotlib. Only supports RGB images currently

Parameters:

ax – matplotlib axis object on which to plot the thumbnail. Optional.

property shape🔗

convenience method. Calling `tile.shape` is equivalent to calling `tile.image.shape`

SlideDataset🔗

class pathml.core.SlideDataset(*slides*)🔗

Container for a dataset of WSIs

Parameters:

slides – list of SlideData objects

run(*pipeline*, *client=None*, *distributed=True*, ***kwargs*)🔗

Runs a preprocessing pipeline on all slides in the dataset

Parameters:

- **pipeline** (*pathml.preprocessing.pipeline.Pipeline*) – Preprocessing pipeline.
- **client** – dask.distributed client
- **distributed** (*bool*) – Whether to distribute model using client. Defaults to True.
- **kwargs** (*dict*) – keyword arguments passed to `run()` for each slide

write(*dir*, *filenames=None*)🔗

Write all SlideData objects to the specified directory. Calls .write() method for each slide in the dataset. Optionally pass a list of filenames to use, otherwise filenames will be created from `.name` attributes of each slide.

Parameters:

- **dir** (*Union[str, bytes, os.PathLike]*) – Path to directory where slides are to be saved
- **filenames** (*List[str], optional*) – list of filenames to be used.

Tiles and Masks helper classes🔗

class pathml.core.Tiles(*h5manager*, *tiles=None*)🔗

Object wrapping a dict of tiles.

Parameters:

tiles (Union[dict[tuple[int], *~pathml.core.tiles.Tile*], list[*~pathml.core.tiles.Tile*]]) – tile objects

add(*tile*)🔗

Add tile indexed by tile.coords to tiles.

Parameters:

tile (*Tile*) – tile object

property keys

remove(*key*)

Remove tile from tiles.

Parameters:

key (*str*) – key (coords) indicating tile to be removed

property tile_shape

update(*tile*)

Update a tile.

Parameters:

tile (*pathml.core.tile.Tiles*) – key of tile to be updated

class pathml.core.Masks(*h5manager*, *masks=None*)

Object wrapping a dict of masks.

Parameters:

- **h5manager** (*pathml.core.h5pathManager*) –
- **masks** (*dict*) – dictionary of np.ndarray objects representing ex. labels, segmentations.

add(*key*, *mask*)

Add mask indexed by key to self.h5manager.

Parameters:

- **key** (*str*) – key
- **mask** (*np.ndarray*) – array of mask. Must contain elements of type int8

property keys

remove(*key*)

Remove mask.

Parameters:

key (*str*) – key indicating mask to be removed

slice(*slicer*)

Slice all masks in self.h5manager extending of numpy array slicing.

Parameters:

slices – list where each element is an object of type slice indicating how the dimension should be sliced

Slide Backends

OpenslideBackend

class pathml.core.OpenSlideBackend(*filename*)

Use OpenSlide to interface with image files.

Depends on openslide-python which wraps the openslide C library.

Parameters:

filename (*str*) – path to image file on disk

extract_region(*location*, *size*, *level=None*)

Extract a region of the image

Parameters:

- **location** (*Tuple[int, int]*) – Location of top-left corner of tile (i, j)
- **size** (*Union[int, Tuple[int, int]]*) – Size of each tile. May be a tuple of (height, width) or a single integer, in which case square tiles of that size are generated.
- **level** (*int*) – level from which to extract chunks. Level 0 is highest resolution.

Returns:

image at the specified region

Return type:

np.ndarray

generate_tiles(*shape=3000*, *stride=None*, *pad=False*, *level=0*)🔗

Generator over tiles.

Padding works as follows: If `pad is False`, then the first tile will start flush with the edge of the image, and the tile locations will increment according to specified stride, stopping with the last tile that is fully contained in the image. If `pad is True`, then the first tile will start flush with the edge of the image, and the tile locations will increment according to specified stride, stopping with the last tile which starts in the image. Regions outside the image will be padded with 0. For example, for a 5x5 image with a tile size of 3 and a stride of 2, tile generation with `pad=False` will create 4 tiles total, compared to 6 tiles if `pad=True`.

Parameters:

- **shape** (*int or tuple(int)*) – Size of each tile. May be a tuple of (height, width) or a single integer, in which case square tiles of that size are generated.
- **stride** (*int*) – stride between chunks. If `None`, uses `stride = size` for non-overlapping chunks. Defaults to `None`.
- **pad** (*bool*) – How to handle tiles on the edges. If `True`, these edge tiles will be zero-padded and yielded with the other chunks. If `False`, incomplete edge chunks will be ignored. Defaults to `False`.
- **level** (*int, optional*) – For slides with multiple levels, which level to extract tiles from. Defaults to 0 (highest resolution).

Yields:

pathml.core.tile.Tile – Extracted Tile object

get_image_shape(*level=0*)🔗

Get the shape of the image at specified level.

Parameters:

level (*int*) – Which level to get shape from. Level 0 is highest resolution. Defaults to 0.

Returns:

Shape of image at target level, in (i, j) coordinates.

Return type:

Tuple[int, int]

get_thumbnail(*size*)🔗

Get a thumbnail of the slide.

Parameters:

size (*Tuple[int, int]*) – the maximum size of the thumbnail

Returns:

RGB thumbnail image

Return type:

np.ndarray

BioFormatsBackend🔗

class pathml.core.BioFormatsBackend(*filename*, *dtype=None*)🔗

Use BioFormats to interface with image files.

Now support multi-level images. Depends on python-bioformats which wraps ome bioformats java library, parses pixel and metadata of proprietary formats, and converts all formats to OME-TIFF. Please cite: https://pubmed.ncbi.nlm.nih.gov/20513764/

Parameters:

- **filename** (*str*) – path to image file on disk
- **dtype** (*numpy.dtype*) – data type of image. If `None`, will use BioFormats to infer the data type from the image's OME metadata. Defaults to `None`.

Note

While the Bio-Formats convention uses XYZCT channel order, we use YXZCT for compatibility with the rest of PathML which is based on (i, j) coordinate system.

extract_region(*location*, *size*, *level=0*, *series_as_channels=False*, *normalize=True*)🔗

Extract a region of the image. All bioformats images have 5 dimensions representing (i, j, z, channel, time). Even if an image does not have multiple z-series or time-series, those dimensions will still be kept. For example, a standard RGB image will be of shape (i, j, 1, 3, 1). If a tuple with len < 5 is passed, missing dimensions will be retrieved in full.

Parameters:

- **location** (*Tuple[int, int]*) – (i, j) location of corner of extracted region closest to the origin.
- **size** (*Tuple[int, int, ...]*) – (i, j) size of each region. If an integer is passed, will convert to a
- **of** (*tuple*) – dimensions will be retrieved in full.
- **level** (*int*) – level from which to extract chunks. Level 0 is highest resolution. Defaults to 0.
- **series_as_channels** (*bool*) – Whether to treat image series as channels. If `True`, multi-level images are not supported. Defaults to `False`.
- **normalize** (*bool, optional*) – Whether to normalize the image to int8 before returning. Defaults to True. If False, image will be returned as-is immediately after reading, typically in float64.

Returns:

image at the specified region. 5-D array of (i, j, z, c, t)

Return type:

np.ndarray

generate_tiles(*shape=3000*, *stride=None*, *pad=False*, *level=0*, ***kwargs*)🔗

Generator over tiles.

Padding works as follows: If `pad is False`, then the first tile will start flush with the edge of the image, and the tile locations will increment according to specified stride, stopping with the last tile that is fully contained in the image. If `pad is True`, then the first tile will start flush with the edge of the image, and the tile locations will increment according to specified stride, stopping with the last tile which starts in the image. Regions outside the image will be padded with 0. For example, for a 5x5 image with a tile size of 3 and a stride of 2, tile generation with `pad=False` will create 4 tiles total, compared to 6 tiles if `pad=True`.

Parameters:

- **shape** (*int or tuple(int)*) – Size of each tile. May be a tuple of (height, width) or a single integer, in which case square tiles of that size are generated.
- **stride** (*int*) – stride between chunks. If `None`, uses `stride = size` for non-overlapping chunks. Defaults to `None`.
- **pad** (*bool*) – How to handle tiles on the edges. If `True`, these edge tiles will be zero-padded and yielded with the other chunks. If `False`, incomplete edge chunks will be ignored. Defaults to `False`.
- ****kwargs** – Other arguments passed through to `extract_region()` method.

Yields:

pathml.core.tile.Tile – Extracted Tile object

get_image_shape(*level=None*)🔗

Get the shape of the image on specific level.

Parameters:

level (*int*) – Which level to get shape from. If `level is None`, returns the shape of the biggest level. Defaults to `None`.

Returns:

Shape of image (i, j) at target level

Return type:

Tuple[int, int]

get_thumbnail(*size=None*)🔗

Get a thumbnail of the image. Since there is no default thumbnail for multiparametric, volumetric images, this function supports downsampling of all image dimensions.

Parameters:

size (*Tuple[int, int]*) – thumbnail size

Returns:

RGB thumbnail image

Return type:

np.ndarray

Example

Get 1000x1000 thumbnail of 7 channel fluorescent image. shape = data.slide.get_image_shape() thumb = data.slide.get_thumbnail(size=(1000,1000, shape[2], shape[3], shape[4]))

DICOMBackend🔗

class pathml.core.DICOMBackend(*filename*)🔗

Interface with DICOM files on disk. Provides efficient access to individual Frame items contained in the Pixel Data element without loading the entire element into memory. Assumes that frames are non-overlapping. DICOM does not support multi-level images.

Parameters:

filename (*str*) – Path to the DICOM Part10 file on disk

extract_region(*location*, *size=None*, *level=None*)🔗

Extract a single frame from the DICOM image.

Parameters:

- **location** (*Union[int, Tuple[int, int]]*) – coordinate location of top-left corner of frame, or integer index of frame.
- **size** (*Union[int, Tuple[int, int]]*) – Size of each tile. May be a tuple of (height, width) or a single integer, in which case square tiles of that size are generated. Must be the same as the frame size.

Returns:

image at the specified region

Return type:

np.ndarray

generate_tiles(*shape*, *stride*, *pad*, *level=0*, ***kwargs*)🔗

Generator over tiles. For DICOMBackend, each tile corresponds to a frame.

Parameters:

- **shape** (*int or tuple(int)*) – Size of each tile. May be a tuple of (height, width) or a single integer, in which case square tiles of that size are generated. Must match frame size.

- **stride** (*int*) – Ignored for DICOMBackend. Frames are yielded individually.
- **pad** (*bool*) – How to handle tiles on the edges. If `True`, these edge tiles will be zero-padded and yielded with the other chunks. If `False`, incomplete edge chunks will be ignored. Defaults to `False`.

Yields:

pathml.core.tile.Tile – Extracted Tile object

static get_bot(*fp*)

Reads the value of the Basic Offset Table. This table is used to access individual frames without loading the entire file into memory

Parameters:

fp (*pydicom.filebase.DicomFile*) – pydicom DicomFile object

Returns:

Offset of each Frame of the Pixel Data element following the Basic Offset Table

Return type:

list

get_image_shape()

Get the shape of the image.

Returns:

Shape of image (H, W)

Return type:

Tuple[int, int]

abstract get_thumbnail(*size*, ***kwargs*)

h5pathManager

class pathml.core.h5managers.h5pathManager(*h5path=None*, *slidedata=None*)

Interface between slidedata object and data management on disk by h5py.

add_mask(*key*, *mask*)

Add mask to h5. This manages **slide-level masks**.

Parameters:

- **key** (*str*) – mask key
- **mask** (*np.ndarray*) – mask array

add_tile(*tile*)

Add a tile to h5path.

Parameters:

tile (*pathml.core.tile.Tile*) – Tile object

get_mask(*item*, *slicer=None*)

get_slidetype()

get_tile(*item*)

Retrieve tile from h5manager by key or index.

Parameters:

item (*int, str, tuple*) – key or index of tile to be retrieved

Returns:

Tile(pathml.core.tile.Tile)

remove_mask(*key*)🔗

Remove mask by key.

Parameters:

key (*str*) – key indicating mask to be removed

remove_tile(*key*)🔗

Remove tile from self.h5 by key.

slice_masks(*slicer*)🔗

Generator slicing all tiles, extending numpy array slicing.

Parameters:

slicer – List where each element is an object of type slice https://docs.python.org/3/c-api/slice.html indicating how the corresponding dimension should be sliced. The list length should correspond to the dimension of the tile. For 2D H&E images, pass a length 2 list of slice objects.

Yields:

key(str) – mask key val(np.ndarray): mask

update_mask(*key*, *mask*)🔗

Update a mask.

Parameters:

- **key** (*str*) – key indicating mask to be updated
- **mask** (*np.ndarray*) – mask

Previous Next

Datasets API

Downloadable Datasets

class pathml.datasets.PanNukeDataModule(*data_dir, download=False, shuffle=True, transforms=None, nucleus_type_labels=False, split=None, batch_size=8, hovernet_preprocess=False*)

DataModule for the PanNuke Dataset. Contains 256px image patches from 19 tissue types with annotations for 5 nucleus types. For more information, see: https://warwick.ac.uk/fac/sci/dcs/research/tia/data/pannuke

Parameters:

- **data_dir** (*str*) – Path to directory where PanNuke data is
- **download** (*bool, optional*) – Whether to download the data. If `True`, checks whether data files exist in `data_dir` and downloads them to `data_dir` if not. If `False`, checks to make sure that data files exist in `data_dir`. Default `False`.
- **shuffle** (*bool, optional*) – Whether to shuffle images. Defaults to `True`.
- **transforms** (*optional*) – Data augmentation transforms to apply to images. Transform must accept two arguments: (mask and image) and return a dict with "image" and "mask" keys. See an example here: https://albumentations.ai/docs/getting_started/mask_augmentation/
- **nucleus_type_labels** (*bool, optional*) –

  Whether to provide nucleus type labels, or binary nucleus labels. If `True`, then masks will be returned with six channels, corresponding to

  0. Neoplastic cells
  1. Inflammatory
  2. Connective/Soft tissue cells
  3. Dead Cells
  4. Epithelial
  5. Background

  If `False`, then the returned mask will have a single channel, with zeros for background pixels and ones for nucleus pixels (i.e. the inverse of the Background mask). Defaults to `False`.

- **split** (*int, optional*) –

  How to divide the three folds into train, test, and validation splits. Must be one of {1, 2, 3, None} corresponding to the following splits:

  1. Training: Fold 1; Validation: Fold 2; Testing: Fold 3
  2. Training: Fold 2; Validation: Fold 1; Testing: Fold 3
  3. Training: Fold 3; Validation: Fold 2; Testing: Fold 1

  If `None`, then the entire PanNuke dataset will be used. Defaults to `None`.

- **batch_size** (*int, optional*) – batch size for dataloaders. Defaults to 8.
- **hovernet_preprocess** (*bool*) – Whether to perform preprocessing specific to HoVer-Net architecture. If `True`, the center of mass of each nucleus will be computed, and an additional mask will be returned with the distance of each nuclear pixel to its center of mass in the horizontal and vertical dimensions. This corresponds to Gamma(I) from the HoVer-Net paper. Defaults to `False`.

References

Gamper, J., Koohbanani, N.A., Benet, K., Khuram, A. and Rajpoot, N., 2019, April. PanNuke: an open pan-cancer histology dataset for nuclei instance segmentation and classification. In European Congress on Digital Pathology (pp. 11-19). Springer, Cham.

Gamper, J., Koohbanani, N.A., Graham, S., Jahanifar, M., Khurram, S.A., Azam, A., Hewitt, K. and Rajpoot, N., 2020. PanNuke Dataset Extension, Insights and Baselines. arXiv preprint arXiv:2003.10778.

property test_dataloader🔗

Dataloader for test set. Yields (image, mask, tissue_type), or (image, mask, hv, tissue_type) for HoVer-Net

property train_dataloader🔗

Dataloader for training set. Yields (image, mask, tissue_type), or (image, mask, hv, tissue_type) for HoVer-Net

property valid_dataloader🔗

Dataloader for validation set. Yields (image, mask, tissue_type), or (image, mask, hv, tissue_type) for HoVer-Net

class pathml.datasets.DeepFocusDataModule(*data_dir*, *download=False*, *shuffle=True*, *transforms=None*, *batch_size=8*)🔗

DataModule for the DeepFocus dataset. The DeepFocus dataset comprises four slides from different patients, each with four different stains (H&E, Ki67, CD21, and CD10) for a total of 16 whole-slide images. For each slide, a region of interest (ROI) of approx 6mm^2 was scanned at 40x magnification with an Aperio ScanScope on nine different focal planes, generating 216,000 samples with varying amounts of blurriness. Tiles with offset values between [-0.5µm, 0.5µm] are labeled as in-focus and the rest of the images are labeled as blurry.

See: https://github.com/cialab/DeepFocus

Parameters:

- **data_dir** (*str*) – file path to directory containing data.
- **download** (*bool, optional*) – Whether to download the data. If `True`, checks whether data files exist in `data_dir` and downloads them to `data_dir` if not. If `False`, checks to make sure that data files exist in `data_dir`. Default `False`.
- **shuffle** (*bool, optional*) – Whether to shuffle images. Defaults to `True`.
- **transforms** (*optional*) – Data augmentation transforms to apply to images.
- **batch_size** (*int, optional*) – batch size for dataloaders. Defaults to 8.

Reference:

Senaras, C., Niazi, M.K.K., Lozanski, G. and Gurcan, M.N., 2018. DeepFocus: detection of out-of-focus regions in whole slide digital images using deep learning. PloS one, 13(10), p.e0205387.

property test_dataloader🔗

property train_dataloader🔗

property valid_dataloader🔗

ML Dataset classes

class pathml.datasets.TileDataset(*file_path*)

PyTorch Dataset class for h5path files

Each item is a tuple of (`tile_image`, `tile_masks`, `tile_labels`, `slide_labels`) where:

- `tile_image` is a torch.Tensor of shape (C, H, W) or (T, Z, C, H, W)
- `tile_masks` is a torch.Tensor of shape (n_masks, tile_height, tile_width)
- `tile_labels` is a dict
- `slide_labels` is a dict

This is designed to be wrapped in a PyTorch DataLoader for feeding tiles into ML models. Note that label dictionaries are not standardized, as users are free to store whatever labels they want. For that reason, PyTorch cannot automatically stack labels into batches. When creating a DataLoader from a TileDataset, it may therefore be necessary to create a custom `collate_fn` to specify how to create batches of labels. See: https://discuss.pytorch.org/t/how-to-use-collate-fn/27181

Parameters:

file_path (*str*) – Path to .h5path file on disk

class pathml.datasets.EntityDataset(*cell_dir=None*, *tissue_dir=None*, *assign_dir=None*)

Torch Geometric Dataset class for storing cell or tissue graphs. Each item returns a pathml.graph.utils.HACTPairData object.

Parameters:

- **cell_dir** (*str*) – Path to folder containing cell graphs
- **tissue_dir** (*str*) – Path to folder containing tissue graphs
- **assign_dir** (*str*) – Path to folder containing assignment matrices

Previous Next

PathML

-
- Graph API
- View PathML on GitHub

Previous Next

Graph API

Graph Building

class pathml.graph.preprocessing.BaseGraphBuilder(*nr_annotation_classes: int = 5*, *annotation_background_class=None*, *add_loc_feats=False*, *return_networkx=False*, ***kwargs*)

Base interface class for graph building.

Parameters:

- **nr_annotation_classes** (*int*) – Number of classes in annotation. Used only if setting node labels.
- **annotation_background_class** (*int*) – Background class label in annotation. Used only if setting node labels.
- **add_loc_feats** (*bool*) – Flag to include location-based features (ie normalized centroids) in node feature representation. Defaults to False.
- **return_networkx** (*bool*) – Whether to return as a networkx graph object. Deafults to returning a Pytorvh Geometric Data object.

References

[1] https://github.com/BiomedSciAI/histocartography/tree/main with an AGPL-3.0 license (https://www.gnu.org/licenses/) [2] Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI Workshop on Computational Pathology (pp. 117-128). PMLR.

process(*instance_map*, *features=None*, *annotation=None*, *target=None*)

Generates a graph from a given instance_map and features

process_with_centroids(*centroids*, *features=None*, *image_size=None*, *annotation=None*,

target=None)

Generates a graph from a given node centroids and features

class pathml.graph.preprocessing.KNNGraphBuilder(*k=5*, *thresh=None*, ***kwargs*)

k-Nearest Neighbors Graph class for graph building.

Parameters:

- **k** (*int, optional*) – Number of neighbors. Defaults to 5.
- **thresh** (*int, optional*) – Maximum allowed distance between 2 nodes. Defaults to None (no thresholding).

Returns:

A pathml.graph.utils.Graph object containing node and edge information.

References

[1] https://github.com/BiomedSciAI/histocartography/tree/main with an AGPL-3.0 license (https://www.gnu.org/licenses/) [2] Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI Workshop on Computational Pathology (pp. 117-128). PMLR.

class pathml.graph.preprocessing.RAGGraphBuilder(*kernel_size=3*, *hops=1*, ***kwargs*)

Region Adjacency Graph builder class.

Parameters:

- **kernel_size** (*int, optional*) – Size of the kernel to detect connectivity. Defaults to 5.
- **hops** (*int, optional*) – Number of hops in a multi-hop neighbourhood. Defaults to 1.

Returns:

A pathml.graph.utils.Graph object containing node and edge information.

References

[1] https://github.com/BiomedSciAI/histocartography/tree/main with an AGPL-3.0 license (https://www.gnu.org/licenses/) [2] Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI Workshop on Computational Pathology (pp. 117-128). PMLR.

class pathml.graph.preprocessing.MSTGraphBuilder(*k=5*, *thresh=None*, ***kwargs*)

Minimum Spanning Tree Graph class for graph building.

Parameters:

- **k** (*int, optional*) – Number of neighbors. Defaults to 5.
- **thresh** (*int, optional*) – Maximum allowed distance between 2 nodes. Defaults to None (no thresholding).

Returns:

A pathml.graph.utils.Graph object containing node and edge information.

Tissue Extraction

class pathml.graph.preprocessing.SuperpixelExtractor(*nr_superpixels: int = None*, *superpixel_size: int = None*, *max_nr_superpixels=None*, *blur_kernel_size=1*, *compactness=20*, *max_iterations=10*, *threshold=0.03*, *connectivity=2*, *color_space='rgb'*, *downsampling_factor=1*, ***kwargs*)

Helper class to extract superpixels from images

Parameters:

- **nr_superpixels** (*None, int*) – The number of super pixels before any merging.
- **superpixel_size** (*None, int*) – The size of super pixels before any merging.
- **max_nr_superpixels** (*int, optional*) – Upper bound for the number of super pixels. Useful when providing a superpixel size.
- **blur_kernel_size** (*float, optional*) – Size of the blur kernel. Defaults to 0.
- **compactness** (*int, optional*) – Compactness of the superpixels. Defaults to 30.
- **max_iterations** (*int, optional*) – Number of iterations of the slic algorithm. Defaults to 10.
- **threshold** (*float, optional*) – Connectivity threshold. Defaults to 0.03.
- **connectivity** (*int, optional*) – Connectivity for merging graph. Defaults to 2.
- **downsampling_factor** (*int, optional*) – Downsampling factor from the input image resolution. Defaults to 1.

References

[1] https://github.com/BiomedSciAI/histocartography/tree/main with an AGPL-3.0 license (https://www.gnu.org/licenses/) [2] Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI

Workshop on Computational Pathology (pp. 117-128). PMLR.

process(*input_image*, *tissue_mask=None*)

Return the superpixels of a given input image

class pathml.graph.preprocessing.SLICSuperpixelExtractor(***kwargs*)

Use the SLIC algorithm to extract superpixels.

References

[1] https://github.com/BiomedSciAI/histocartography/tree/main with an AGPL-3.0 license (https://www.gnu.org/licenses/) [2] Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI Workshop on Computational Pathology (pp. 117-128). PMLR.

class pathml.graph.preprocessing.MergedSuperpixelExtractor(***kwargs*)

Use the SLIC algorithm to extract superpixels and a merging function to merge superpixels

References

[1] https://github.com/BiomedSciAI/histocartography/tree/main with an AGPL-3.0 license (https://www.gnu.org/licenses/) [2] Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI Workshop on Computational Pathology (pp. 117-128). PMLR.

process(*input_image*, *tissue_mask=None*)

Return the superpixels of a given input image

class pathml.graph.preprocessing.ColorMergedSuperpixelExtractor(*w_hist: float = 0.5*, *w_mean: float = 0.5*, ***kwargs*)

Superpixel merger based on color attibutes taken from the HACT-Net Implementation :param w_hist: Weight of the histogram features for merging. Defaults to 0.5. :type w_hist: float, optional :param w_mean: Weight of the mean features for merging. Defaults to 0.5. :type w_mean: float, optional

References

[1] https://github.com/BiomedSciAI/histocartography/tree/main with an AGPL-3.0 license (https://www.gnu.org/licenses/) [2] Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI Workshop on Computational Pathology (pp. 117-128). PMLR.

Graph Feature Extraction

class pathml.graph.preprocessing.GraphFeatureExtractor(*use_weight=False*, *alpha=0.85*)

Extracts features from a networkx graph object.

Parameters:

- **use_weight** (*bool, optional*) – Whether to use edge weights for feature computation. Defaults to False.
- **alpha** (*float, optional*) – Alpha value for personalized page-rank. Defaults to 0.85.

Returns:

Dictionary of keys as feature type and values as features

get_stats(*dct*, *prefix='add_pre'*)

process(*G*)

Previous Next

PathML

-
- Inference API
- View PathML on GitHub

Previous Next

Inference API

Base Inference Class

class pathml.inference.InferenceBase

Base class for all ONNX Models. Each transform must operate on a Tile.

abstract F(*target*)

functional implementation

abstract apply(*tile*)

modify Tile object in-place

get_model_card()

Returns model card.

reshape(*image*)

standard reshaping of tile image

set_citation(*citation*)

Sets the “citation” parameter in the model card.

Parameters:

citation (*str*) – Citation for the model

set_model_input_notes(*note*)

Sets the “model_input_notes” parameter in the model card.

Parameters:

note (*str*) – Comments on the model input

set_model_output_notes(*note*)

Sets the “model_output_notes” parameter in the model card.

Parameters:

note (*str*) – Comments on the model output

set_model_type(*model_type*)

Sets the “model_type” parameter in the model card.

Parameters:

model_type (*str*) – Type of model, e.g. “segmentation”

set_name(*name*)

Sets the “name” parameter in the model card.

Parameters:

name (*str*) – Name for the model

set_notes(*note*)

Sets the “notes” parameter in the model card.

Parameters:

note (*str*) – Any extra information you want to put in the model card

set_num_classes(*num*)

Sets the “num_classes” parameter in the model card.

Parameters:

num (*int*) – Number of classes your model predicts

Inference Class

class pathml.inference.Inference(*model_path=None*, *input_name='data'*, *num_classes=None*, *model_type=None*, *local=True*)

Transformation to run inferrence on ONNX model.

Assumptions:

- The ONNX model has been cleaned by *remove_initializer_from_input* first

Parameters:

- **model_path** (*str*) – path to ONNX model w/o initializers,
- **input_name** (*str*) – name of the input the ONNX model accepts, default = “data”
- **num_classes** (*int*) – number of classes you are predicting
- **model_type** (*str*) – type of model, e.g. “segmentation”
- **local** (*bool*) – True if the model is stored locally, default = “True”

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

inference(*image*)

HaloAI Inference Class

class pathml.inference.HaloAIInference(*model_path=None*, *input_name='data'*, *num_classes=None*, *model_type=None*, *local=True*)

Transformation to run inferrence on HALO AI ONNX model.

Assumptions:

- Assumes that the ONNX model returns a tensor in which there is one prediction map for each class
- For example, if there are 5 classes, the ONNX model will output a (1, 5, Height, Weight) tensor
- If you select to argmax the classes, the class assumes a softmax or sigmoid has already been applied
- HaloAI ONNX models always have 20 class maps so you need to index into the first x maps if you have x classes

Parameters:

- **model_path** (*str*) – path to HaloAI ONNX model w/o initializers,
- **input_name** (*str*) – name of the input the ONNX model accepts, default = “data”
- **num_classes** (*int*) – number of classes you are predicting
- **model_type** (*str*) – type of model, e.g. “segmentation”
- **local** (*bool*) – True if the model is stored locally, default = “True”

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

RemoteTestHoverNet Class

class pathml.inference.RemoteTestHoverNet(*model_path='temp.onnx'*, *input_name='data'*, *num_classes=5*, *model_type='Segmentation'*, *local=False*)

Transformation to run inference on ONNX model.

Citation for model: Pocock J, Graham S, Vu QD, Jahanifar M, Deshpande S, Hadjigeorghiou G, Shephard A, Bashir RM, Bilal M, Lu W, Epstein D. TIAToolbox as an end-to-end library for advanced tissue image analytics. Communications medicine. 2022 Sep 24;2(1):120.

Parameters:

- **model_path** (*str*) – temp file name to download onnx from huggingface, do not change
- **input_name** (*str*) – name of the input the ONNX model accepts, default = “data”, do not change
- **num_classes** (*int*) – number of classes you are predicting, do not change
- **model_type** (*str*) – type of model, e.g. “segmentation”, do not change
- **local** (*bool*) – True if the model is stored locally, default = “True”, do not change

apply(*tile*)

modify Tile object in-place

remove()

RemoteMesmer Class

class pathml.inference.RemoteMesmer(*model_path='temp.onnx'*, *input_name='data'*, *num_classes=3*, *model_type='Segmentation'*, *local=False*, *nuclear_channel=None*, *cytoplasm_channel=None*, *image_resolution=0.5*, *preprocess_kwargs=None*, *postprocess_kwargs_nuclear=None*, *postprocess_kwargs_whole_cell=None*)

Transformation to run inference on ONNX Mesmer model.

Citation for model: Greenwald NF, Miller G, Moen E, Kong A, Kagel A, Dougherty T, Fullaway CC, McIntosh BJ, Leow KX, Schwartz MS, Pavelchek C. Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning. Nature biotechnology. 2022 Apr;40(4):555-65.

Parameters:

- **model_path** (*str*) – temp file name to download onnx from huggingface, do not change
- **input_name** (*str*) – name of the input the ONNX model accepts, default = “data”, do not change
- **num_classes** (*int*) – number of classes you are predicting, do not change
- **model_type** (*str*) – type of model, e.g. “segmentation”, do not change
- **local** (*bool*) – True if the model is stored locally, default = “True”, do not change
- **nuclear_channel** (*int*) – channel that defines cell nucleus
- **cytoplasm_channel** (*int*) – channel that defines cell membrane or cytoplasm
- **image_resolution** (*float*) – pixel resolution of image in microns. Currently only supports 0.5
- **preprocess_kwargs** (*dict*) – keyword arguemnts to pass to pre-processing function
- **postprocess_kwargs_nuclear** (*dict*) – keyword arguments to pass to post-processing function
- **postprocess_kwargs_whole_cell** (*dict*) – keyword arguments to pass to post-processing function

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

inference(*image*)

remove()

Helper functions

pathml.inference.remove_initializer_from_input(*model_path*, *new_path*)

Removes initializers from HaloAI ONNX models Taken from https://github.com/microsoft/onnxruntime/blob/main/tools/python/remove_initializer_from_input.py

Parameters:

- **model_path** (*str*) – path to ONNX model,
- **new_path** (*str*) – path to save adjusted model w/o initializers,

Returns:

ONNX model w/o initializers to run inference using PathML

pathml.inference.check_onnx_clean(*model_path*)

Checks if the model has had it's initalizers removed from input graph. Adapted from from https://github.com/microsoft/onnxruntime/blob/main/tools/python/remove_initializer_from_input.py

Parameters:

model_path (*str*) – path to ONNX model,

Returns:

Boolean if there are initializers in input graph.

pathml.inference.convert_pytorch_onnx(*model*, *dummy_tensor*, *model_name*, *opset_version=10*, *input_name='data'*)

Converts a Pytorch Model to ONNX Adjusted from https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html

You need to define the model class and load the weights before exporting. See URL above for full steps.

Parameters:

- **model_path** (*torch.nn.Module Model*) – Pytorch model to be converted,

- **dummy_tensor** (*torch.tensor*) – dummy input tensor that is an example of what will be passed into the model,
- **model_name** (*str*) – name of ONNX model created with .onnx at the end,
- **opset_version** (*int*) – which opset version you want to use to export
- **input_name** (*str*) – name assigned to dummy_tensor

Returns:

Exports ONNX model converted from Pytorch

Previous Next

Contributing

ML API

Models

class pathml.ml.models.hovernet.HoVerNet(*n_classes=None*)

Model for simultaneous segmentation and classification based on HoVer-Net. Can also be used for segmentation only, if class labels are not supplied. Each branch returns logits.

Parameters:

n_classes (*int*) – Number of classes for classification task. If `None` then the classification branch is not used.

References

Graham, S., Vu, Q.D., Raza, S.E.A., Azam, A., Tsang, Y.W., Kwak, J.T. and Rajpoot, N., 2019. Hover-Net: Simultaneous segmentation and classification of nuclei in multi-tissue histology images. Medical Image Analysis, 58, p.101563.

forward(*inputs*)

class pathml.ml.models.hactnet.HACTNet(*cell_params*, *tissue_params*, *classifier_params*)

Hierarchical cell-to-tissue model for supervised prediction using cell and tissue graphs.

Parameters:

- **cell_params** (*dict*) – Dictionary containing parameters for cell graph GNN.
- **tissue_params** (*dict*) – Dictionary containing parameters for tissue graph GNN.
- **classifier_params** (*dict*) – Dictionary containing parameters for prediction MLP.

References

Pati, P., Jaume, G., Foncubierta-Rodriguez, A., Feroce, F., Anniciello, A.M., Scognamiglio, G., Brancati, N., Fiche, M., Dubruc, E., Riccio, D. and Di Bonito, M., 2022. Hierarchical graph representations in digital pathology. Medical image analysis, 75, p.102264.

forward(*batch*)

Layers

class pathml.ml.layers.GNNLayer(*layer*, *in_channels*, *hidden_channels*, *num_layers*, *out_channels*, *readout_op*, *readout_type*, *kwargs*)

GNN layer for processing graph structures.

Parameters:

- **layer** (*str*) – Type of torch_geometric GNN layer to be used. See https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#convolutional-layers for all available options.
- **in_channels** (*int*) – Number of input features supplied to the model.
- **hidden_channels** (*int*) – Number of hidden channels used in each layer of the GNN model.
- **num_layers** (*int*) – Number of message-passing layers in the model.
- **out_channels** (*int*) – Number of output features returned by the model.
- **readout_op** (*str*) – Readout operation to summarize features from each layer. Supports 'lstm' and 'concat'.
- **readout_type** (*str*) – Type of readout to aggregate node embeddings. Supports 'mean'.
- **kwargs** (*dict*) – Extra layer-specific arguments. Must have required keyword arguments of layer from https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#convolutional-layers.

forward(*x*, *edge_index*, *batch*, *with_readout=True*)

Helper functions

pathml.ml.models.hovernet.compute_hv_map(*mask*)

Preprocessing step for HoVer-Net architecture. Compute center of mass for each nucleus, then compute distance of each nuclear pixel to its corresponding center of mass. Nuclear pixel distances are normalized to (-1, 1). Background pixels are left as 0. Operates on a single mask. Can be used in Dataset object to make Dataloader compatible with HoVer-Net.

Based on https://github.com/vqdang/hover_net/blob/195ed9b6cc67b12f908285492796fb5c6c15a000/src/loader/augs.py#L192

Parameters:

mask (*np.ndarray*) – Mask indicating individual nuclei. Array of shape (H, W), where each pixel is in {0, …, n} with 0 indicating background pixels and {1, …, n} indicating n unique nuclei.

Returns:

array of hv maps of shape (2, H, W). First channel corresponds to horizontal and second vertical.

Return type:

np.ndarray

pathml.ml.models.hovernet.loss_hovernet(*outputs*, *ground_truth*, *n_classes=None*)

Compute loss for HoVer-Net. Equation (1) in Graham et al.

Parameters:

- **outputs** –

  Output of HoVer-Net. Should be a list of [np, hv] if n_classes is None, or a list of [np, hv, nc] if n_classes is not None. Shapes of each should be:

  - np: (B, 2, H, W)
  - hv: (B, 2, H, W)
  - nc: (B, n_classes, H, W)

- **ground_truth** – True labels. Should be a list of [mask, hv], where mask is a Tensor of shape (B, 1, H, W) if n_classes is `None` or (B, n_classes, H, W) if n_classes is not `None`. hv is a tensor of precomputed horizontal and vertical distances of nuclear pixels to their corresponding centers of mass, and is of shape (B, 2, H, W).
- **n_classes** (*int*) – Number of classes for classification task. If `None` then the classification branch is not used.

References

Graham, S., Vu, Q.D., Raza, S.E.A., Azam, A., Tsang, Y.W., Kwak, J.T. and Rajpoot, N., 2019. Hover-Net: Simultaneous segmentation and classification of nuclei in multi-tissue histology images. Medical Image Analysis, 58, p.101563.

pathml.ml.models.hovernet.remove_small_objs(*array_in*, *min_size*)

Removes small foreground regions from binary array, leaving only the contiguous regions which are above the size threshold. Pixels in regions below the size threshold are zeroed out.

Parameters:

- **array_in** (*np.ndarray*) – Input array. Must be binary array with dtype=np.uint8.
- **min_size** (*int*) – Minimum size of each region.

Returns:

Array of labels for regions above the threshold. Each separate contiguous region is labelled with

a different integer from 1 to n, where n is the number of total distinct contiguous regions

Return type:

np.ndarray

pathml.ml.models.hovernet.post_process_batch_hovernet(*outputs*, *n_classes*, *small_obj_size_thresh=10*, *kernel_size=21*, *h=0.5*, *k=0.5*, *return_nc_out_preds=False*)

Post-process HoVer-Net outputs to get a final predicted mask. See: Section B of HoVer-Net article and https://github.com/vqdang/hover_net/blob/14c5996fa61ede4691e87905775e8f4243da6a62/models/hovernet/post_proc.py#L27

Parameters:

- **outputs** (*list*) –

  Outputs of HoVer-Net model. List of [np_out, hv_out], or [np_out, hv_out, nc_out] depending on whether model is predicting classification or not.

  - np_out is a Tensor of shape (B, 2, H, W) of logit predictions for binary classification
  - hv_out is a Tensor of shape (B, 2, H, W) of predictions for horizontal/vertical maps
  - nc_out is a Tensor of shape (B, n_classes, H, W) of logits for classification

- **n_classes** (*int*) – Number of classes for classification task. If `None` then only segmentation is performed.
- **small_obj_size_thresh** (*int*) – Minimum number of pixels in regions. Defaults to 10.
- **kernel_size** (*int*) – Width of Sobel kernel used to compute horizontal and vertical gradients.
- **h** (*float*) – hyperparameter for thresholding nucleus probabilities. Defaults to 0.5.
- **k** (*float*) – hyperparameter for thresholding energy landscape to create markers for watershed segmentation. Defaults to 0.5.

Returns:

If n_classes is None, returns det_out. In classification setting, returns (det_out, class_out).

- det_out is np.ndarray of shape (B, H, W)
- class_out is np.ndarray of shape (B, n_classes, H, W)

Each pixel is labelled from 0 to n, where n is the number of individual nuclei detected. 0 pixels indicate background. Pixel values i indicate that the pixel belongs to the ith nucleus.

Return type:

np.ndarray

Modified previous method to output nc_out_preds.

Previous Next

PathML

-
- Preprocessing API
- View PathML on GitHub

Previous Next

Preprocessing API

Pipeline

class pathml.preprocessing.Pipeline(*transform_sequence=None*)

Compose a sequence of Transforms

Parameters:

transform_sequence (*list*) – sequence of transforms to be consecutively applied. List of *pathml.core.Transform* objects

apply(*tile*)

modify Tile object in-place

save(*filename*)

save pipeline to disk

Parameters:

filename (*str*) – save path on disk

Transforms

class pathml.preprocessing.MedianBlur(*kernel_size=5*)

Median blur kernel.

Parameters:

kernel_size (*int*) – Width of kernel. Must be an odd number. Defaults to 5.

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.GaussianBlur(*kernel_size=5*, *sigma=5*)

Gaussian blur kernel.

Parameters:

- **kernel_size** (*int*) – Width of kernel. Must be an odd number. Defaults to 5.
- **sigma** (*float*) – Variance of Gaussian kernel. Variance is assumed to be equal in X and Y axes. Defaults to 5.

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.BoxBlur(*kernel_size=5*)

Box (average) blur kernel.

Parameters:

kernel_size (*int*) – Width of kernel. Defaults to 5.

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.BinaryThreshold(*mask_name=None*, *use_otsu=True*, *threshold=0*, *inverse=False*)

Binary thresholding transform to create a binary mask. If input image is RGB it is first converted to greyscale, otherwise the input must have 1 channel.

Parameters:

- **mask_name** (*str*) – Name of mask that is created.
- **use_otsu** (*bool*) – Whether to use Otsu's method to automatically determine optimal threshold. Defaults to True.
- **threshold** (*int*) – Specified threshold. Ignored if `use_otsu is True`. Defaults to 0.
- **inverse** (*bool*) – Whether to use inverse threshold. If using inverse threshold, pixels below the threshold will be returned as 1. Otherwise pixels below the threshold will be returned as 0. Defaults to `False`.

References

Otsu, N., 1979. A threshold selection method from gray-level histograms. IEEE transactions on systems, man, and cybernetics, 9(1), pp.62-66.

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.MorphOpen(*mask_name=None*, *kernel_size=5*, *n_iterations=1*)

Morphological opening. First applies erosion operation, then dilation. Reduces noise by removing small objects from the background. Operates on a binary mask.

Parameters:

- **mask_name** (*str*) – Name of mask on which to apply transform
- **kernel_size** (*int*) – Size of kernel for default square kernel. Ignored if a custom kernel is specified. Defaults to 5.
- **n_iterations** (*int*) – Number of opening operations to perform. Defaults to 1.

F(*mask*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.MorphClose(*mask_name=None*, *kernel_size=5*, *n_iterations=1*)

Morphological closing. First applies dilation operation, then erosion. Reduces noise by closing small

holes in the foreground. Operates on a binary mask.

Parameters:

- **mask_name** (*str*) – Name of mask on which to apply transform
- **kernel_size** (*int*) – Size of kernel for default square kernel. Ignored if a custom kernel is specified. Defaults to 5.
- **n_iterations** (*int*) – Number of opening operations to perform. Defaults to 1.

F(*mask*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.ForegroundDetection(*mask_name=None*, *min_region_size=5000*, *max_hole_size=1500*, *outer_contours_only=False*)

Foreground detection for binary masks. Identifies regions that have a total area greater than specified threshold. Supports including holes within foreground regions, or excluding holes above a specified area threshold.

Parameters:

- **min_region_size** (*int*) – Minimum area of detected foreground regions, in pixels. Defaults to 5000.
- **max_hole_size** (*int*) – Maximum size of allowed holes in foreground regions, in pixels. Ignored if `outer_contours_only is True`. Defaults to 1500.
- **outer_contours_only** (*bool*) – If true, ignore holes in detected foreground regions. Defaults to False.
- **mask_name** (*str*) – Name of mask on which to apply transform

References

Lu, M.Y., Williamson, D.F., Chen, T.Y., Chen, R.J., Barbieri, M. and Mahmood, F., 2020. Data Efficient and Weakly Supervised Computational Pathology on Whole Slide Images. arXiv preprint arXiv:2004.09666.

F(*mask*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.SuperpixelInterpolation(*region_size=10*, *n_iter=30*)

Divide input image into superpixels using SLIC algorithm, then interpolate each superpixel with average color. SLIC superpixel algorithm described in Achanta et al. 2012.

Parameters:

- **region_size** (*int*) – region_size parameter used for superpixel creation. Defaults to 10.
- **n_iter** (*int*) – Number of iterations to run SLIC algorithm. Defaults to 30.

References

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P. and Süsstrunk, S., 2012. SLIC superpixels compared to state-of-the-art superpixel methods. IEEE transactions on pattern analysis and machine intelligence, 34(11), pp.2274-2282.

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.StainNormalizationHE(*target='normalize'*, *stain_estimation_method='macenko'*, *optical_density_threshold=0.15*, *regularizer=0.1*, *angular_percentile=0.01*, *background_intensity=245*, *stain_matrix_target_od=np.array([[0.5626, 0.2159], [0.7201, 0.8012], [0.4062, 0.5581]]).T*, *max_c_target=np.array([[1.9705, 1.0308]])*)

Normalize H&E stained images to a reference slide. Also can be used to separate hematoxylin and eosin channels.

H&E images are assumed to be composed of two stains, each one having a vector of its characteristic RGB values. The stain matrix is a 2x3 matrix where the first row corresponds to the hematoxylin stain vector and the second corresponds to eosin stain vector. The stain matrix can be estimated from a reference image in a number of ways; here we provide implementations of two such algorithms from Macenko et al. and Vahadane et al.

After estimating the stain matrix for an image, the next step is to assign stain concentrations to each pixel. Each pixel is assumed to be a linear combination of the two stain vectors, where the coefficients are the intensities of each stain vector at that pixel. To solve for the intensities, we use least squares in Macenko method and lasso in vahadane method.

The image can then be reconstructed by applying those pixel intensities to a stain matrix. This allows

you to standardize the appearance of an image by reconstructing it using a reference stain matrix. Using this method of normalization may help account for differences in slide appearance arising from variations in staining procedure, differences between scanners, etc. Images can also be reconstructed using only a single stain vector, e.g. to separate the hematoxylin and eosin channels of an H&E image.

This code is based in part on StainTools: https://github.com/Peter554/StainTools

Parameters:

- **target** (*str*) – one of 'normalize', 'hematoxylin', or 'eosin'. Defaults to 'normalize'
- **stain_estimation_method** (*str*) – method for estimating stain matrix. Must be one of 'macenko' or 'vahadane'. Defaults to 'macenko'.
- **optical_density_threshold** (*float*) – Threshold for removing low-optical density pixels when estimating stain vectors. Defaults to 0.15
- **regularizer** (*float*) – Regularization parameter for dictionary learning when estimating stain vector using vahadane method. Ignored if `concentration_estimation_method != 'vahadane'`. Defaults to 0.1
- **angular_percentile** (*float*) – Percentile for stain vector selection when estimating stain vector using Macenko method. Ignored if `concentration_estimation_method != 'macenko'`. Defaults to 0.01
- **background_intensity** (*int*) – Intensity of background light. Must be an integer between 0 and 255. Defaults to 245.
- **stain_matrix_target_od** (*np.ndarray*) – Stain matrix for reference slide. Matrix of H and E stain vectors in optical density (OD) space. Stain matrix is (2, 3) and first row corresponds to hematoxylin. Default stain matrix can be used, or you can also fit to a reference slide of your choosing by calling `fit_to_reference()`.
- **max_c_target** (*np.ndarray*) – Maximum concentrations of each stain in reference slide. Default can be used, or you can also fit to a reference slide of your choosing by calling `fit_to_reference()`.

References

Macenko, M., Niethammer, M., Marron, J.S., Borland, D., Woosley, J.T., Guan, X., Schmitt, C. and Thomas, N.E., 2009, June. A method for normalizing histology slides for quantitative analysis. In 2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro (pp. 1107-1110). IEEE.

Vahadane, A., Peng, T., Sethi, A., Albarqouni, S., Wang, L., Baust, M., Steiger, K., Schlitter, A.M., Esposito, I. and Navab, N., 2016. Structure-preserving color normalization and sparse stain separation for histological images. IEEE transactions on medical imaging, 35(8), pp.1962-1971.

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

fit_to_reference(*target*)

class pathml.preprocessing.NucleusDetectionHE(*mask_name=None*, *stain_estimation_method='vahadane'*, *superpixel_region_size=10*, *n_iter=30*, ***stain_kwargs*)

Simple nucleus detection algorithm for H&E stained images. Works by first separating hematoxylin channel, then doing interpolation using superpixels, and finally using Otsu's method for binary thresholding.

Parameters:

- **stain_estimation_method** (*str*) – Method for estimating stain matrix. Defaults to "vahadane"
- **superpixel_region_size** (*int*) – region_size parameter used for superpixel creation. Defaults to 10.
- **n_iter** (*int*) – Number of iterations to run SLIC superpixel algorithm. Defaults to 30.
- **mask_name** (*str*) – Name of mask that is created.
- **stain_kwargs** (*dict*) – other arguments passed to `StainNormalizationHE()`

References

Hu, B., Tang, Y., Eric, I., Chang, C., Fan, Y., Lai, M. and Xu, Y., 2018. Unsupervised learning for cell-level visual representation in histopathology images with generative adversarial networks. IEEE journal of biomedical and health informatics, 23(3), pp.1316-1328.

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.TissueDetectionHE(*mask_name=None*, *use_saturation=True*, *blur_ksize=17*, *threshold=None*, *morph_n_iter=3*, *morph_k_size=7*, *min_region_size=5000*, *max_hole_size=1500*, *outer_contours_only=False*)

Detect tissue regions from H&E stained slide. First applies a median blur, then binary thresholding, then morphological opening and closing, and finally foreground detection.

Parameters:

- **use_saturation** (*bool*) – Whether to convert to HSV and use saturation channel for tissue detection. If False, convert from RGB to greyscale and use greyscale image_ref for tissue detection. Defaults to True.
- **blur_ksize** (*int*) – kernel size used to apply median blurring. Defaults to 15.
- **threshold** (*int*) – threshold for binary thresholding. If None, uses Otsu's method. Defaults to None.
- **morph_n_iter** (*int*) – number of iterations of morphological opening and closing to apply. Defaults to 3.
- **morph_k_size** (*int*) – kernel size for morphological opening and closing. Defaults to 7.
- **min_region_size** (*int*) – Minimum area of detected foreground regions, in pixels. Defaults to 5000.
- **max_hole_size** (*int*) – Maximum size of allowed holes in foreground regions, in pixels. Ignored if outer_contours_only=True. Defaults to 1500.
- **outer_contours_only** (*bool*) – If true, ignore holes in detected foreground regions. Defaults to False.
- **mask_name** (*str*) – name for new mask

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.LabelArtifactTileHE(*label_name=None*)

Applies a rule-based method to identify whether or not an image contains artifacts (e.g. pen marks). Based on criteria from Kothari et al. 2012 ACM-BCB 218-225.

Parameters:

label_name (*str*) – name for new mask

References

Kothari, S., Phan, J.H., Osunkoya, A.O. and Wang, M.D., 2012, October. Biological interpretation of morphological patterns in histopathological whole-slide images. In Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine (pp. 218-225).

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.LabelWhiteSpaceHE(*label_name=None*, *greyscale_threshold=230*, *proportion_threshold=0.5*)

Simple threshold method to label an image as majority whitespace. Converts image to greyscale. If the proportion of pixels exceeding the greyscale threshold is greater than the proportion threshold, then the image is labelled as whitespace.

Parameters:

label_name (*str*) – name for new mask

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.SegmentMIF(*model='mesmer'*, *nuclear_channel=None*, *cytoplasm_channel=None*, *image_resolution=0.5*, *preprocess_kwargs=None*, *postprocess_kwargs_nuclear=None*, *postprocess_kwargs_whole_cell=None*)

Transform applying segmentation to MIF images.

Input image must be formatted (c, x, y) or (batch, c, x, y). z and t dimensions must be selected before calling SegmentMIF

Supported models:

- **Mesmer**: Mesmer uses human-in-the-loop pipeline to train a ResNet50 backbone w/ Feature Pyramid Network segmentation model on 1.3 million cell annotations and 1.2 million nuclear annotations (TissueNet dataset). Model outputs predictions for centroid and boundary of every nucleus and cell, then centroid and boundary predictions are used as inputs to a watershed algorithm that creates segmentation masks.

Note

Mesmer model requires installation of deepcell dependency: `pip install deepcell`

Parameters:

- **model** (*str*) – string indicating which segmentation model to use. Currently only 'mesmer' is supported.
- **nuclear_channel** (*int*) – channel that defines cell nucleus
- **cytoplasm_channel** (*int*) – channel that defines cell membrane or cytoplasm
- **image_resolution** (*float*) – pixel resolution of image in microns
- **preprocess_kwargs** (*dict*) – keyword arguemnts to pass to pre-processing function
- **postprocess_kwargs_nuclear** (*dict*) – keyword arguments to pass to post-processing function
- **postprocess_kwargs_whole_cell** (*dict*) – keyword arguments to pass to post-processing function

References

Greenwald, N.F., Miller, G., Moen, E. et al. Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning. Nat Biotechnol (2021). https://doi.org/10.1038/s41587-021-01094-0

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.SegmentMIFRemote(*model_path='temp.onnx'*, *nuclear_channel=None*, *cytoplasm_channel=None*, *image_resolution=0.5*, *preprocess_kwargs=None*, *postprocess_kwargs_nuclear=None*, *postprocess_kwargs_whole_cell=None*)

Transform applying segmentation to MIF images using a Mesmer model. Mesmer uses human-in-the-loop pipeline to train a ResNet50 backbone w/ Feature Pyramid Network segmentation model on 1.3 million cell annotations and 1.2 million nuclear annotations (TissueNet dataset). Model outputs predictions for centroid and boundary of every nucleus and cell, then centroid and boundary predictions are used as inputs to a watershed algorithm that creates segmentation masks.

Implements *pathml.inference.RemoteMesmer* in the backend.

Input image must be formatted (c, x, y) or (batch, c, x, y). z and t dimensions must be selected before calling SegmentMIF

Parameters:

- **model_path** (*str*) – path where the ONNX model is downloaded
- **nuclear_channel** (*int*) – channel that defines cell nucleus
- **cytoplasm_channel** (*int*) – channel that defines cell membrane or cytoplasm
- **image_resolution** (*float*) – pixel resolution of image in microns. Currently only supports 0.5
- **preprocess_kwargs** (*dict*) – keyword arguemnts to pass to pre-processing function
- **postprocess_kwargs_nuclear** (*dict*) – keyword arguments to pass to post-processing function
- **postprocess_kwargs_whole_cell** (*dict*) – keyword arguments to pass to post-processing function

References

Greenwald, N.F., Miller, G., Moen, E. et al. Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning. Nat Biotechnol (2021). https://doi.org/10.1038/s41587-021-01094-0

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.QuantifyMIF(*segmentation_mask*)

Convert segmented image into anndata.AnnData counts object AnnData. Counts objects are used to interface with the Python single cell analysis ecosystem Scanpy. The counts object contains a summary of channel statistics in each cell along with its coordinate.

Parameters:

segmentation_mask (*str*) – key indicating which mask to use as label image

F(*img*, *segmentation*, *coords_offset=(0, 0)*)

Functional implementation

Parameters:

- **img** (*np.ndarray*) – Input image of shape (i, j, n_channels)

- **segmentation** (*np.ndarray*) – Segmentation map of shape (i, j) or (i, j, 1). Zeros are background. Regions should be labelled with unique integers.
- **coords_offset** (*tuple, optional*) – Coordinates (i, j) used to convert tile-level coordinates to slide-level. Defaults to (0, 0) for no offset.

Returns:

Counts matrix

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.CollapseRunsVectra

Coerce Vectra output to standard format. For compatibility with transforms, tiles need to have their shape collapsed to (x, y, c)

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.CollapseRunsCODEX(*z*)

Coerce CODEX output to standard format. CODEX format is (x, y, z, c, t) where c=4 (4 runs per cycle) and t is the number of cycles. Output format is (x, y, c) where all cycles are collapsed into c (c = 4 * # of cycles).

Parameters:

z (*int*) – in-focus z-plane

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.RescaleIntensity(*in_range='image', out_range='dtype'*)

Return image after stretching or shrinking its intensity levels. The desired intensity range of the input and output, in_range and out_range respectively, are used to stretch or shrink the intensity range of the

input image This function is a wrapper for 'rescale_intensity' function from scikit-image: https://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.rescale_intensity

Parameters:

- **in_range** (*str or 2-tuple, optional*) – Min and max intensity values of input image. The possible values for this parameter are enumerated below. 'image' : Use image min/max as the intensity range. 'dtype' : Use min/max of the image's dtype as the intensity range. 'dtype-name' : Use intensity range based on desired dtype. Must be valid key in DTYPE_RANGE. '2-tuple' : Use range_values as explicit min/max intensities.
- **out_range** (*str or 2-tuple, optional*) – Min and max intensity values of output image. The possible values for this parameter are enumerated below. 'image' : Use image min/max as the intensity range. 'dtype' : Use min/max of the image's dtype as the intensity range. 'dtype-name' : Use intensity range based on desired dtype. Must be valid key in DTYPE_RANGE. '2-tuple' : Use range_values as explicit min/max intensities.

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.HistogramEqualization(*nbins=256*, *mask=None*)

Return image after histogram equalization. This function is a wrapper for 'equalize_hist' function from scikit-image: https://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.equalize_hist

Parameters:

- **nbins** (*int, optional*) – Number of gray bins for histogram. Note: this argument is ignored for integer images, for which each integer is its own bin.
- **mask** (*ndarray of bools or 0s and 1s, optional*) – Array of same shape as image. Only points at which mask == True are used for the equalization, which is applied to the whole image.

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

class pathml.preprocessing.AdaptiveHistogramEqualization(*kernel_size=None*, *clip_limit=0.3*, *nbins=256*)

Contrast Limited Adaptive Histogram Equalization (CLAHE). An algorithm for local contrast enhancement, that uses histograms computed over different tile regions of the image. Local details can therefore be enhanced even in regions that are darker or lighter than most of the image. This function is a wrapper for 'equalize_adapthist' function from scikit-image: https://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.equalize_adapthist

Parameters:

- **kernel_size** (*int or array_like, optional*) – Defines the shape of contextual regions used in the algorithm. If iterable is passed, it must have the same number of elements as image.ndim (without color channel). If integer, it is broadcasted to each image dimension. By default, kernel_size is 1/8 of image height by 1/8 of its width.
- **clip_limit** (*float*) – Clipping limit, normalized between 0 and 1 (higher values give more contrast).
- **nbins** (*int*) – Number of gray bins for histogram ("data range").

F(*image*)

functional implementation

apply(*tile*)

modify Tile object in-place

TileStitching

This section covers the *TileStitcher* class, which is specialized for stitching tiled images, particularly useful in digital pathology.

class pathml.preprocessing.tilestitcher.TileStitcher(*qupath_jarpath=[]*, *java_path=None*, *memory='40g'*, *bfconvert_dir='./'*)

A Python class for stitching tiled images, specifically designed for spectrally unmixed images in a pyramidal OME-TIFF format.

This class is a Python implementation of Pete Bankhead's script for image stitching, available at available at https://gist.github.com/petebankhead/b5a86caa333de1fdcff6bdee72a20abe. It requires QuPath and JDK to be installed prior to use.

Parameters:

- **qupath_jarpath** (*list*) – Paths to QuPath JAR files.
- **java_path** (*str*) – Path to Java installation.
- **memory** (*str*) – Memory allocation for the JVM.

- **bfconvert_dir** (*str*) – Directory for Bio-Formats conversion tools.

checkTIFF(*file*)

Check if a given file is a valid TIFF file.

This method reads the first few bytes of the file to determine if it conforms to TIFF specifications.

Parameters:

file (*str*) – Path to the file to be checked.

Returns:

True if the file is a valid TIFF file, False otherwise.

Return type:

bool

static format_jvm_options(*qupath_jars*, *memory*)

is_bfconvert_available()

Check if bfconvert is available.

parseRegion(*file*, *z=0*, *t=0*)

Parse an image region from a given TIFF file.

Parameters:

- **file** (*str*) – Path to the TIFF file.
- **z** (*int, optional*) – Z-position of the image. Defaults to 0.
- **t** (*int, optional*) – Time point of the image. Defaults to 0.

Returns:

An ImageRegion object representing the parsed region.

Return type:

ImageRegion

parse_regions(*infiles*)

Parse image regions from a list of TIFF files and build a sparse image server.

Parameters:

infiles (*list*) – List of paths to TIFF files.

Returns:

A server containing the parsed image regions.

Return type:

SparseImageServer

run_bfconvert(*stitched_image_path*, *bfconverted_path=None*, *delete_original=True*)

Run the Bio-Formats conversion tool on a stitched image.

Parameters:

- **stitched_image_path** (*str*) – Path to the stitched image.
- **bfconverted_path** (*str, optional*) – Path for the converted image. If None, a default path is generated.
- **delete_original** (*bool*) – If True, delete the original stitched image after conversion.

run_image_stitching(*input_dir*, *output_filename*, *downsamples=[1, 8]*, *separate_series=False*)

Perform image stitching on the provided TIFF files and output a stitched OME-TIFF image.

Parameters:

- **input_dir** (*str*) – Directory containing the input TIFF files.
- **output_filename** (*str*) – Filename for the output stitched image.
- **downsamples** (*list, optional*) – List of downsample levels. Defaults to [1, 8].
- **separate_series** (*bool, optional*) – Whether to separate the series. Defaults to False.

setup_bfconvert(*bfconvert_dir*)

Set up Bio-Formats conversion tool (bfconvert) in the given directory.

Parameters:

bfconvert_dir (*str*) – Directory path for setting up bfconvert.

Returns:

Path to the bfconvert tool.

Return type:

str

shutdown()

Shut down the Java Virtual Machine (JVM) if it's running.

toShort(*b1*, *b2*)

Convert two bytes to a short integer.

This helper function is used for interpreting the binary data in file headers, particularly for TIFF files.

Parameters:

- **b1** (*byte*) – The first byte.
- **b2** (*byte*) – The second byte.

Returns:

The short integer represented by the two bytes.

Return type:

int

Previous Next

Previous Next

Utilities API

Documentation for various utilities from all modules.

Logging Utils

class pathml.PathMLLogger

Convenience methods for turning on or off and configuring logging for PathML. Note that this can also be achieved by interfacing with loguru directly

Example:

```
from pathml import PathMLLogger as pml

# turn on logging for PathML
pml.enable()

# turn off logging for PathML
pml.disable()

# turn on logging and output logs to a file named 'logs.txt', with colorization enabled
pml.enable(sink="logs.txt", colorize=True)
```

static disable()

Turn off logging for PathML

static enable(*sink=sys.stderr*, *level='DEBUG'*, *fmt='PathML:{level}:{time:HH:mm:ss} | {module}:{function}:{line} | {message}'*, ***kwargs*)

Turn on and configure logging for PathML

Parameters:

- **sink** (*str or io._io.TextIOWrapper*, *optional*) – Destination sink for log messages. Defaults to `sys.stderr`.
- **level** (*str*) – level of logs to capture. Defaults to 'DEBUG'.
- **fmt** (*str*) – Formatting for the log message. Defaults to: 'PathML:{level}:{time:HH:mm:ss} | {module}:{function}:{line} | {message}'
- ****kwargs** (*dict*, *optional*) – additional options passed to configure logger. See: loguru documentation

Core Utils

pathml.core.utils.readtupleh5(*h5*, *key*)

Read tuple from h5.

Parameters:

- **h5** (*h5py.Dataset or h5py.Group*) – h5 object that will be read from
- **key** (*str*) – key where data to read is stored

pathml.core.utils.writedataframeh5(*h5*, *name*, *df*)

Write dataframe as h5 dataset.

Parameters:

- **h5** (*h5py.Dataset*) – root of h5 object that df will be written into
- **name** (*str*) – name of dataset to be created
- **df** (*pd.DataFrame*) – dataframe to be written

pathml.core.utils.writedicth5(*h5*, *name*, *dic*)

Write dict as attributes of h5py.Group.

Parameters:

- **h5** (*h5py.Dataset*) – root of h5 object that dic will be written into
- **name** (*str*) – name of dataset to be created
- **dic** (*str*) – dict to be written

pathml.core.utils.writestringh5(*h5*, *name*, *st*)

Write string as h5 attribute.

Parameters:

- **h5** (*h5py.Dataset*) – root of h5 object that st will be written into
- **name** (*str*) – name of dataset to be created
- **st** (*str*) – string to be written

pathml.core.utils.writetupleh5(*h5*, *name*, *tup*)

Write tuple as h5 attribute.

Parameters:

- **h5** (*h5py.Dataset*) – root of h5 object that tup will be written into
- **name** (*str*) – name of dataset to be created
- **tup** (*str*) – tuple to be written

pathml.core.utils.readcounts(*h5*)

Read counts using anndata h5py.

Parameters:

h5 (*h5py.Dataset*) – h5 object that will be read

pathml.core.utils.writecounts(*h5*, *counts*)

Write counts using anndata h5py.

Parameters:

- **h5** (*h5py.Dataset*) – root of h5 object that counts will be written into
- **name** (*str*) – name of dataset to be created
- **tup** (*anndata.AnnData*) – anndata object to be written

Graph Utils

pathml.graph.utils.Graph(*node_centroids*, *edge_index*, *node_features=None*, *node_labels=None*, *edge_features=None*, *target=None*)

Constructs pytorch-geometric data object for saving and loading

Parameters:

- **node_centroids** (*torch.tensor*) – Coordinates of the centers of each entity (cell or tissue) in the graph
- **node_features** (*torch.tensor*) – Computed features of each entity (cell or tissue) in the graph
- **edge_index** (*torch.tensor*) – Edge index in sparse format between nodes in the graph
- **node_labels** (*torch.tensor*) – Node labels of each entity (cell or tissue) in the graph. Defaults to None.

- **target** (*torch.tensor*) – Target label if used in a supervised setting. Defaults to None.

pathml.graph.utils.HACTPairData(*x_cell*, *edge_index_cell*, *x_tissue*, *edge_index_tissue*, *assignment*, *target*)

Constructs pytorch-geometric data object for handling both cell and tissue data.

Parameters:

- **x_cell** (*torch.tensor*) – Computed features of each cell in the graph
- **edge_index_cell** (*torch.tensor*) – Edge index in sparse format between nodes in the cell graph
- **x_tissue** (*torch.tensor*) – Computed features of each tissue in the graph
- **edge_index_tissue** (*torch.tensor*) – Edge index in sparse format between nodes in the tissue graph
- **assignment** (*torch.tensor*) – Assigment matrix that contains mapping between cells and tissues.
- **target** (*torch.tensor*) – Target label if used in a supervised setting.

References

Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI Workshop on Computational Pathology (pp. 117-128). PMLR.

pathml.graph.utils.get_full_instance_map(*wsi*, *patch_size*, *mask_name='cell'*)

Generates and returns the normalized image, cell instance map and cell centroids from pathml SlideData object

Parameters:

- **wsi** (*pathml.core.SlideData*) – Normalized WSI object with detected cells in the 'masks' slot
- **patch_size** (*int*) – Patch size used for cell detection
- **mask_name** (*str*) – Name of the mask slot storing the detected cells. Defaults to 'cell'.

Returns:

The image in np.unint8 format, the instance map for the entity and the instance centroids for each entity in the instance map as numpy arrays.

pathml.graph.utils.build_assignment_matrix(*low_level_centroids*, *high_level_map*, *matrix=False*)

Builds an assignment matrix/mapping between low-level centroid locations and a high-level

segmentation map

Parameters:

- **low_level_centroids** (*numpy.array*) – The low-level centroid coordinates in x-y plane
- **map** (*high-level*) – The high-level map returned from regionprops
- **matrix** (*bool*) – Whether to return in a matrix format. If True, returns a N*L matrix where N is the number of low-level instances and L is the number of high-level instances. If False, returns this mapping in sparse format. Defaults to False.

Returns:

The assignment matrix as a numpy array.

References

[1] https://github.com/BiomedSciAI/histocartography/tree/main [2] Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI Workshop on Computational Pathology (pp. 117-128). PMLR.

pathml.graph.utils.two_hop(*edge_index*, *num_nodes*)

Calculates the two-hop graph. :param edge_index: The edge index in sparse form of the graph. :type edge_index: torch.tensor :param num_nodes: maximum number of nodes. :type num_nodes: int

Returns:

Output edge index tensor.

Return type:

torch.tensor

References

[1] https://github.com/BiomedSciAI/histocartography/tree/main [2] Jaume, G., Pati, P., Anklin, V., Foncubierta, A. and Gabrani, M., 2021, September. Histocartography: A toolkit for graph analytics in digital pathology. In MICCAI Workshop on Computational Pathology (pp. 117-128). PMLR.

pathml.graph.utils.two_hop_no_sparse(*edge_index*, *num_nodes*)

Calculates the two-hop graph without using sparse tensors, in case of M1/M2 chips. :param edge_index: The edge index in sparse form of the graph (2, E) :type edge_index: torch.tensor :param num_nodes: maximum number of nodes. :type num_nodes: int

Returns:

Output edge index tensor.

Return type:

torch.tensor

Datasets Utils

class pathml.datasets.utils.DeepPatchFeatureExtractor(*patch_size*, *batch_size*, *architecture*, *device='cpu'*, *entity='cell'*, *fill_value=255*, *threshold=0.2*, *resize_size=224*, *with_instance_masking=False*, *extraction_layer=None*)

Patch feature extracter of a given architecture and put it on GPU if available using Pathml.datasets.InstanceMapPatchDataset.

Parameters:

- **patch_size** (*int*) – Desired size of patch.
- **batch_size** (*int*) – Desired size of batch.
- **architecture** (*str or nn.Module*) – String of architecture. According to torchvision.models syntax, or nn.Module class directly.
- **entity** (*str*) – Entity to be processed. Must be one of 'cell' or 'tissue'. Defaults to 'cell'.
- **device** (*torch.device*) – Torch Device used for inference.
- **fill_value** (*int*) – Value to fill outside the instance maps. Defaults to 255.
- **threshold** (*float*) – Threshold for processing a patch or not.
- **resize_size** (*int*) – Desired resized size to input the network. If None, no resizing is done and the patches of size patch_size are provided to the network. Defaults to None.
- **with_instance_masking** (*bool*) – If pixels outside instance should be masked. Defaults to False.
- **extraction_layer** (*str*) – Name of the network module from where the features are extracted.

Returns:

Tensor of features computed for each entity.

process(*input_image*, *instance_map*)

Main processing function that takes in an input image and an instance map and returns features for all entities in the instance map

pathml.datasets.utils.pannuke_multiclass_mask_to_nucleus_mask(*multiclass_mask*)

Convert multiclass mask from PanNuke to a single channel nucleus mask. Assumes each pixel is assigned to one and only one class. Sums across channels, except the last mask channel which indicates background pixels in PanNuke. Operates on a single mask.

Parameters:

multiclass_mask (*torch.Tensor*) – Mask from PanNuke, in classification setting. (i.e. `nucleus_type_labels=True`). Tensor of shape (6, 256, 256).

Returns:

Tensor of shape (256, 256).

pathml.datasets.utils._remove_modules(*model*, *last_layer*)

Remove all modules in the model that come after a given layer.

Parameters:

- **model** (*nn.Module*) – A PyTorch model.
- **last_layer** (*str*) – Last layer to keep in the model.

Returns:

Model (nn.Module) without pruned modules.

ML Utils

pathml.ml.utils.center_crop_im_batch(*batch*, *dims*, *batch_order='BCHW'*)

Center crop images in a batch.

Parameters:

- **batch** – The batch of images to be cropped
- **dims** – Amount to be cropped (tuple for H, W)

pathml.ml.utils.dice_loss(*true*, *logits*, *eps=0.001*)

Computes the Sørensen–Dice loss. Note that PyTorch optimizers minimize a loss. In this case, we would like to maximize the dice loss so we return 1 - dice loss. From: https://github.com/kevinzakka/pytorch-goodies/blob/c039691f349be9f21527bb38b907a940bfc5e8f3/losses.py#L54

Parameters:

- **true** – a tensor of shape [B, 1, H, W].

- **logits** – a tensor of shape [B, C, H, W]. Corresponds to the raw output or logits of the model.
- **eps** – added to the denominator for numerical stability.

Returns:

the Sørensen–Dice loss.

Return type:

dice_loss

pathml.ml.utils.dice_score(*pred*, *truth*, *eps=0.001*)

Calculate dice score for two tensors of the same shape. If tensors are not already binary, they are converted to bool by zero/non-zero.

Parameters:

- **pred** (*np.ndarray*) – Predictions
- **truth** (*np.ndarray*) – ground truth
- **eps** (*float, optional*) – Constant used for numerical stability to avoid divide-by-zero errors. Defaults to 1e-3.

Returns:

Dice score

Return type:

float

pathml.ml.utils.get_sobel_kernels(*size*, *dt=torch.float32*)

Create horizontal and vertical Sobel kernels for approximating gradients Returned kernels will be of shape (size, size)

pathml.ml.utils.wrap_transform_multichannel(*transform*)

Wrapper to make albumentations transform compatible with a multichannel mask. Channel should be in first dimension, i.e. (n_mask_channels, H, W)

Parameters:

transform – Albumentations transform. Must have 'additional_targets' parameter specified with a total of *n_channels* key,value pairs. All values must be 'mask' but the keys don't matter. e.g. for a mask with 3 channels, you could use: *additional targets = {'mask1' : 'mask', 'mask2' : 'mask',*

'pathml' : 'mask'}

Returns:

function that can be called with a multichannel mask argument

pathml.ml.utils.scatter_sum(*src*, *index*, *dim*, *out=None*, *dim_size=None*)

Reduces all values from the `src` tensor into `out` at the indices specified in the `index` tensor along a given axis `dim`.

For each value in `src`, its output index is specified by its index in `src` for dimensions outside of `dim` and by the corresponding value in `index` for dimension `dim`. The applied reduction is defined via the `reduce` argument.

Parameters:

- **src** – The source tensor.
- **index** – The indices of elements to scatter.
- **dim** – The axis along which to index. Default is -1.
- **out** – The destination tensor.
- **dim_size** – If *out* is not given, automatically create output with size *dim_size* at dimension *dim*.

Reference:

https://pytorch-scatter.readthedocs.io/en/latest/_modules/torch_scatter/scatter.html#scatter

pathml.ml.utils.broadcast(*src*, *other*, *dim*)

Broadcast tensors to match output tensor dimension.

pathml.ml.utils.get_degree_histogram(*loader*, *edge_index_str*, *x_str*)

Returns the degree histogram to be used as input for the *deg* argument in *PNAConv*.

pathml.ml.utils.get_class_weights(*loader*)

Returns the per-class weights to be used in weighted loss functions.

Miscellaneous Utils

pathml.utils.upsample_array(*arr*, *factor*)

Upsample array by a factor. Each element in input array will become a CxC block in the upsampled

array, where C is the constant upsampling factor. From https://stackoverflow.com/a/32848377

Parameters:

- **arr** (*np.ndarray*) – input array to be upsampled
- **factor** (*int*) – Upsampling factor

Returns:

np.ndarray

pathml.utils.pil_to_rgb(*image_array_pil*)

Convert PIL RGBA Image to numpy RGB array

pathml.utils.segmentation_lines(*mask_in*)

Generate coords of points bordering segmentations from a given mask. Useful for plotting results of tissue detection or other segmentation.

pathml.utils.plot_mask(*im*, *mask_in*, *ax=None*, *color='red'*, *downsample_factor=None*)

plot results of segmentation, overlaying on original image_ref

Parameters:

- **im** (*np.ndarray*) – Original RGB image_ref
- **mask_in** (*np.ndarray*) – Boolean array of segmentation mask, with True values for masked pixels. Must be same shape as im.
- **ax** – Matplotlib axes object to plot on. If None, creates a new plot. Defaults to None.
- **color** – Color to plot outlines of mask. Defaults to "red". Must be recognized by matplotlib.
- **downsample_factor** – Downsample factor for image_ref and mask to speed up plotting for big images

pathml.utils.contour_centroid(*contour*)

Return the centroid of a contour, calculated using moments. From OpenCV implementation

Parameters:

contour (*np.array*) – Contour array as returned by cv2.findContours

Returns:

(x, y) coordinates of centroid.

Return type:

tuple

pathml.utils.sort_points_clockwise(*points*)

Sort a list of points into clockwise order around centroid, ordering by angle with centroid and x-axis. After sorting, we can pass the points to cv2 as a contour. Centroid is defined as center of bounding box around points.

Parameters:

points (*np.ndarray*) – Array of points (N x 2)

Returns:

Array of points, sorted in order by angle with centroid (N x 2)

Return type:

np.ndarray

Return sorted points

pathml.utils.pad_or_crop(*array*, *target_shape*)

Make dimensions of input array match target shape by either zero-padding or cropping each axis.

Parameters:

- **array** (*np.ndarray*) – Input array
- **target_shape** (*tuple*) – Target shape of output

Returns:

Input array cropped/padded to match target_shape

Return type:

np.ndarray

pathml.utils.RGB_to_HSI(*imarr*)

Convert imarr from RGB to HSI colorspace.

Parameters:

imarr (*np.ndarray*) – numpy array of RGB image_ref (m, n, 3)

Returns:

numpy array of HSI image_ref (m, n, 3)

Return type:

np.ndarray

References

http://eng.usf.edu/~hady/courses/cap5400/rgb-to-hsi.pdf

pathml.utils.RGB_to_OD(*imarr*)

Convert input image from RGB space to optical density (OD) space. *OD = -log(I)*, where I is the input image in RGB space.

Parameters:

imarr (*numpy.ndarray*) – Image array, RGB format

Returns:

Image array, OD format

Return type:

numpy.ndarray

pathml.utils.RGB_to_HSV(*imarr*)

convert image from RGB to HSV

pathml.utils.RGB_to_LAB(*imarr*)

convert image from RGB to LAB color space

pathml.utils.RGB_to_GREY(*imarr*)

convert image_ref from RGB to HSV

pathml.utils.normalize_matrix_rows(*A*)

Normalize the rows of an array.

Parameters:

A (*np.ndarray*) – Input array.

Returns:

Array with rows normalized.

Return type:

np.ndarray

pathml.utils.normalize_matrix_cols(*A*)

Normalize the columns of an array.

Parameters:

A (*np.ndarray*) – An array

Returns:

Array with columns normalized

Return type:

np.ndarray

pathml.utils.plot_segmentation(*ax*, *masks*, *palette=None*, *markersize=5*)

Plot segmentation contours. Supports multi-class masks.

Parameters:

- **ax** – matplotlib axis
- **masks** (*np.ndarray*) – Mask array of shape (n_masks, H, W). Zeroes are background pixels.
- **palette** – color palette to use. if None, defaults to matplotlib.colors.TABLEAU_COLORS
- **markersize** (*int*) – Size of markers used on plot. Defaults to 5

Previous Next
