wsipipe Documentation

Release 0.1.6

David Morrison, Christina Fell

Mar 24, 2023

CONTENTS:

CHAPTER

ONE

GETTING STARTED

1.1 wsipipe

A set of tools for processing pathology whole slide images for deep learning.

- Free software: MIT license
- Documentation: https://wsipipe.readthedocs.io.

1.1.1 Features

- TODO

1.1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

1.2 Installation

1.2.1 Stable release

To install wsipipe, run this command in your terminal:

```
$ pip install wsipipe
```

This is the preferred method to install wsipipe, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.2.2 From sources

The sources for wsipipe can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/StAndrewsMedTech/wsipipe
```

Or download the tarball:

```
$ curl -OJL https://github.com/StAndrewsMedTech/wsipipe/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

1.2.3 Openslide

Wsipipe requires openslide this cannot be installed by pip. Please follow the instructions on https://openslide.org/api/python/ for your operating system.

for example on ubuntu:

```
apt install -y openslide-tools
```

1.3 Usage

To use wsipipe in a project:

```
import wsipipe
```

1.4 Tutorial

Some basics to get you started using wsipipe. Wsipipe is structured around:

- datasets, which contain the details of where files are stored.
- patchsets, which contain details of where patches are within an WSI.

1.4.1 Specifying slide and annotation information

To get started we need to define a set of data we are going to use. A dataset is stored in a pandas DataFrame. Each row contains the information for a single slide. The dataframe should have four columns, slide, annotation, label, and tags.

- slide, contains the path the WSI.
- annotation, contains a path to an annotation file.
- label, contains a slide level label.
- tags, can contain any other information you want to store about the slide.

You can create these dataframes or read from disk. WSIpipe also has some datasets predefined, for example camelyon16.

If you have downloaded the camelyon16 data from and stored it in a local folder: https://camelyon17.grand-challenge.org/Data/ . It is assumed the local folder (path_to_local_folder) is structured in the dame way as the Camleyon16 google drive, that is it should contain two folders named training and testing.

The code to create the wsipipe dataset dataframe is:

```
from wsipipe.datasets import camelyon16

train_dset = camelyon16.training(cam16_path = path_to_local_folder)
```

We only want to use a few slides for the examples in this tutorial so we can cut down the size using sample_dataset. For example if we want to randomly select 2 slides of each label category from the dataset:

```
from wsipipe.datasets.dataset_utils import sample_dataset

small_train_dset = sample_dataset(train_dset, 2)
```

As the dataset is just a pandas dataframe we can access information for an individual slide by specifying the row.:

```
row = small_train_dset.iloc[0]
```

1.4.2 Specifying how to load a dataset

Our dataset has now stored the location of the WSI, annotations and other information. Now we need to specify how these files are to be loaded as not all WSI formats and annotations can be loaded using the same libraries. This is done using dataset loader classes, each of which specifies how to load annotations and slides, as well as the allowable slide labels. A selection of slide and annotation loaders are included in wsipipe. The Camleyon16 dataset loader class is specifed as::

```
from wsipipe.load.datasets.camelyon16 import Camelyon16Loader

dset_loader = Camelyon16Loader()
```

1.4.3 Viewing a slide

Now we have defined where the WSI files are and how to load them, we can open a slide and return the whole slide at a given level in the image pyramid as a numpy array. Depending on the size of the WSI it may not be possible to do this at the lowest levels (highest magnification) of the image pyramid due to lack of memory. In the example we are extracting the thumbnail at level 5.:

```
with dset_loader.load_slide(row.slide) as slide:
    thumb = slide.get_thumbnail(5)
```

This code returns a numpy array, if you want to for example display it as a PIL image in a jupyter notebook.:

```
from wsipipe.utils import np_to_pil

np_to_pil(thumb)
```

1.4.4 Viewing an annotation

We can also read and view the annotations, here we render them at level 5. The annotations for camleyon are read in as labels 1 or 2, in the code below they are mulitplied by 100 to make them visible when displayed.:

```
from wsipipe.load.annotations import visualise_annotations

labelled_image = visualise_annotations(
    row.annotation,
    row.slide,
    dset_loader,
    5,
    row.label
)
np_to_pil(labelled_image*100)
```

1.4.5 Applying background subtraction

Often large parts of WSI are background that contain nothing of interest, therefore we want to split the background from the tissue so we know which are the areas of interest on the slide. There different types of tissue detectors specfied in wsipipe. Here we use a basic Greyscale version. Firstly we specify our tissue detector and define the parameters, then we apply it to a thumbnail of the WSI. This returns a binary mask where True/1/white is tissue and False/0/black is background.:

```
from wsipipe.preprocess.tissue_detection import TissueDetectorGreyScale

tisdet = TissueDetectorGreyScale(grey_level=0.85)
tissmask = tisdet(thumb)
np_to_pil(tissmask)
```

We can also apply filters or morphological operations as part of the tissue detection.:

```
from wsipipe.preprocess.tissue_detection import SimpleClosingTransform,␣
↪SimpleOpeningTransform, GaussianBlur

prefilt = GaussianBlur(sigma=2)
morph = [SimpleOpeningTransform(), SimpleClosingTransform()]
tisdet = TissueDetectorGreyScale(
    grey_level=0.75,
    morph_transform = morph,
    pre_filter = prefilt
)
tissmask = tisdet(thumb)
np_to_pil(tissmask)
```

We can also visualise the mask overlaid on the thumbnail.:

```
from wsipipe.preprocess.tissue_detection import visualise_tissue_detection_for_slide

visualise_tissue_detection_for_slide(row.slide, dset_loader, 5, tisdet)
```

1.4.6 Creating a patchset for a slide

Next we define the location of patches to extract from the slide, which we refer to as a patchset. Here we specify we want to create 256 pixels patches on a regular grid with stride 256 pixels. The patches are extracted at level 0. This will be calculated based on thumbnails and annotations rendered at level 5.:

```
from wsipipe.preprocess.patching import GridPatchFinder, make_patchset_for_slide

patchfinder = GridPatchFinder(patch_level=1, patch_size=512, stride=512, labels_level=5)
pset = make_patchset_for_slide(row.slide, row.annotation, dset_loader, tisdet,␣
↪patchfinder)
```

The patchset is datafrom with the top left position and label for each patch, plus a settings object which stores information which is used for multiple patches such as the patch size and slide path. You can combine multiple settings within one patchset, so the dataframe also records which setting to apply to a patch. We can then use the patchset to visualise the patches overlaid on the slide.:

```
from wsipipe.preprocess.patching import visualise_patches_on_slide

visualise_patches_on_slide(pset, vis_level = 5)
```

There is also a random patch finder available, which extracts a given number of patches at random locations within the tissue area.

1.4.7 Creating patchsets for a dataset

We can also create patchsets for the whole dataset. This simply returns a list of patchsets for each slide in the dataset.:

```
from wsipipe.preprocess.patching import make_patchsets_for_dataset

psets_for_dset = make_patchsets_for_dataset(
    dataset = small_train_dset,
    loader = dset_loader,
    tissue_detector = tisdet,
    patch_finder = patchfinder
)
```

1.4.8 Saving and loading patchsets

For large datasets, this can take a long time and a problem in one file can cause this not to complete. It is frustrating to have to remake the patchsets for all the other slides. Therefore there is also a function to save each patchset individually as it makes them. When the function is rerun it then checks if the patchsets already exists, if so it skips creating it. This function saves each patchset in a separate subdirectory of the output directory.:

```
from wsipipe.preprocess.patching import make_and_save_patchsets_for_dataset

psets_for_dset = make_and_save_patchsets_for_dataset(
    dataset = small_train_dset,
    loader = dset_loader,
    tissue_detector = tisdet,
    patch_finder = patchfinder,
```

(continues on next page)

(continued from previous page)

```
    output_dir = path_to_pset_folder
)
```

You can also load datasets created with the same folder structure.:

```
from wsipipe.preprocess.patching import load_patchsets_from_directory

psets_for_dset = load_patchsets_from_directory(patchsets_dir = path_to_pset_folder)
```

1.4.9 Combining patchsets

You can combine multiple patchsets into one big patchset, for example to combine all the patchsets in a dataset.:

```
from wsipipe.preprocess.patching import combine

all_patches_in_dset = combine(psets_for_dset)
```

1.4.10 Sampling patchsets

You can sample patches from a patchset, there are various samplers available that can be used to create balanced sets, weighted sets etc. The balanced sample will sample num_samples without replacement from each category. If there are fewer than num_samples of one category it will sample the number of samples of the smallest category. If the smallest category is less than floor_samples, it will sample floor_samples from the other categories and all the samples from the smallest category. The sampler returns a patchset.:

```
from wsipipe.preprocess.sample import balanced_sample

sampled_patches = balanced_sample(
    patches = all_patches_in_dset,
    num_samples = 500,
    floor_samples = 100
)
```

1.4.11 Creating patches

Once you have a patchset (an individual slide, a combined patchset or a sampled patchset) it is simple to create the patches from it.:

```
sampled_patches.export_patches(path_to_folder_for_patches)
```

You now have your patches ready for training the deep learning model of your choice.

CHAPTER

TWO

WSIPIPE

2.1 wsipipe package

2.1.1 wsipipe.datasets package

Datasets contain information on sets of data, e.g file locations, number of slides, labels etc A dataset is a dataframe with columns slide, annotation, label and tags

- slide contains WSI path
- annotation contains path to annotation file or slide label
- label contains slide level labels
- tags is any other infomation about the slide (multiple pieces of data are separated by semi colons).

camelyon16 module

This module creates the dataframe for the camelyon 16 dataset with the follwing columns:

- The slide column stores the paths on disk of the whole slide images.
- The annotation column records a path to the annotation files.
- The label column is the slide level label.
- The tags column is blank for camelyon 16.

This assumes there is a folder on disk structured the same as downloading from the camelyon grand challenge Camelyon 16 google drive: https://camelyon17.grand-challenge.org/Data/

testing(*cam16_path=PosixPath('data/camelyon16')*, *project_root=None*)

Create Camleyon 16 testing dataset

This function goes through the input directories for the testing slides, and matches up the annotations and slides. It creates a dataframe with slide path with matching annotation path, and slide label. There is an empty tags column that is not used for this dataset

Parameters

- **cam16_path** (*Path, optional*) – a path relative to the project root that is the location of the Camelyon 16 data. Defaults to data/camelyon16.
- **project_root** (*Optional[Path]*) –

Returns

A dataframe with columns slide, annotation, label and tags

Return type
df (pd.DataFrame)

`training`(*cam16_path=PosixPath('data/camelyon16')*, *project_root=None*)

Create Camleyon 16 training dataset

This function goes through the input directories for the training slides, and matches up the annotations and slides. It creates a dataframe with slide path with matching annotation path, and slide label. There is an empty tags column that is not used for this dataset

Parameters

- **cam16_path** (*Path, optional*) – a path relative to the project root that is the location of the Camelyon 16 data. Defaults to data/camelyon16.
- **project_root** (*Optional[Path]*) –

Returns
A dataframe with columns slide, annotation, label and tags

Return type
df (pd.DataFrame)

stripai module

This module creates the dataframe for the STRIP AI dataset with the following columns:

- The slide column stores the paths on disk of the whole slide images
- The annotation column records a string with the slide label
- The label column is the slide level label
- The tags column contains the center and patient for each slide

This assumes there is a folder on disk structured the same as downloading from the kaggle website https://www.kaggle.com/competitions/mayo-clinic-strip-ai/data

`convert_to_pyramids`(*data_root=PosixPath('data/mayo-clinic-strip-ai')*, *out_root=PosixPath('experiments/mayo_pyramids')*, *project_root=None*)

Create pyramids for whole slide images

The whole slide images as downloaded only contain data at level 0, no other levels are present. This can make it slow to access the slides. This function will run over all the slides in the the dataset and write out copies that contain a pyramid of levels. Files are written to folder experiments/pyramids/

Parameters

- **mayo_path** (*Path, optional*) – a path relative to the project root that is the location of the strip ai data. Defaults to data/mayo-clinic-strip-ai.
- **data_root** (*Path*) –
- **out_root** (*Path*) –
- **project_root** (*Optional[Path]*) –

`training`(*data_root=PosixPath('data/mayo-clinic-strip-ai')*, *project_root=None*)

Create Strip AI training dataset

This function goes through the input directories for the training slides, and matches up the slide paths with infomation in the csv It creates a dataframe with slide path with matching slide label stored for both label and annotation. The tags column stores the patient id and center id.

Parameters

- **mayo_path** (*Path, optional*) – a path relative to the project root that is the location of the stripai data. Defaults to data/mayo-clinic-strip-ai.
- **data_root** (*Path*) –
- **project_root** (*Optional[Path]*) –

Returns
A dataframe with columns slide, annotation, label and tags

Return type
df (pd.DataFrame)

dataset_utils module

sample_dataset(*df*, *samples_per_class*)

Create a subset of a dataset dataframe This function will create a smaller dataframe that only includes n slides per class. This can be used to create smaller datasets for example for debugging pipelines

Parameters

- **df** (*pd.DataFrame*) – A dataframe containing a column called label
- **samples_per_class** (*str*) – The number of slides per class to return

Returns

A copy of the dataframe with samples_per_class rows
for each label

Return type
df (pd.DataFrame)

2.1.2 wsipipe.load package

Contains functionality to load slides, annotations or entire datasets

Subpackages

wsipipe.load.annotations package

All annotations are loaded in the generic annotation format. Individual modules convert specific annotation types to the generic

annotation module

Parent classes that contain functionality for reading annotations. These are used to render different types of annotations into a common format

class Annotation(*name*, *annotation_type*, *label*, *vertices*)

Bases: `object`

Class for a single annotation.

There can be multiple annotations on a slide

Parameters

- **name** (*str*) – Name of the annotation.
- **type** (*str*) – One of Dot, Polygon, Spline or Rectangle
- **label** (*str*) – What label should be given to the annotation
- **vertices** (*List[PointF]*) – A list of vertices, each of which is an PointF object, a named tuple (x, y) of floats.
- **annotation_type** (*str*) –

draw(*image*, *labels*, *factor*)

Renders the annotation into the image.

Parameters

- **image** (*np.array*) – Array to write the annotations into, must have dtype float.
- **labels** (*Dict[str, int]*) – The value to write into the image for each type of label.
- **factor** (*float*) – How much to scale (by divison) each vertex by.

class AnnotationSet(*annotations*, *labels*, *labels_order*, *fill_label*)

Bases: object

Class for all annotations on a slide.

Parameters

- **annotations** (*List[Annotation]*) – A list of all Annotations on a slide
- **labels** (*Dict[str, int]*) – A dictionary where the keys are the names of labels, with the integer values with which the string should be replaced.
- **labels_order** (*List[str]*) – An order the labels should be plotted in. Where annotations overlap they will be drawn in this order, so the final label will be on top
- **fill_label** (*str*) – The label given to any unannotated areas.

render(*shape*, *factor*)

Creates a labelled image containing annotations

This creates an array of size = shape, that is factor times smaller than the level at which the annotation vertexes are specified. Annotations vertex positions are assumed to be specified at level 0, and therefore for many WSI a np.array of the same size as level 0 would not fit in memory. Therefore one factor times smaller is created.

Parameters

- **shape** (Shape) – size of numpy array to create
- **factor** (*float*) – How much to scale (by divison) each vertex by.

Return type
numpy.array

visualise_annotations(*annot_path*, *slide_path*, *loader*, *level*, *slide_label*)

Creates a image render of the annotations of a slide

Converts annotations from level zero to the specified level. Requires slide path to find the correct dimensions of the output image. Returns a numpy array

Parameters

- **annot_path** (*Path*) – A path to the annotation file
- **slide_path** (*Path*) – A path to the WSI file
- **loader** – The loader to use for slides and annots
- **level** (*int*) – the level to create the numpy array
- **slide_label** (*str*) –

Returns
An array the same size as the WSI at level with the annotation labels plotted in it.

Return type
labels_image (np.array)

asapxml module

Functions to load annotations stored in asapxml formats and convert to Annotation class formats

annotation_from_tag(*tag*, *group_labels*)

Convert an asapxml element to annotation format.

Parameters

- **tag** (*Element*) – An element from the xml Element tree
- **group_labels** (*Dict[str, str]*) – A dictionary of group labels that convert values stored in xml PartOfGroup to required label. e.g {"Tumor": "tumor", "Metastasis": "tumor", "Normal": "normal", "Tissue": "normal"}

Return type
Annotation

load_annotations_asapxml(*xml_file_path*, *group_labels*)

Read xml file and create annotations

Parameters

- **xml_file_path** (*Path*) – PAth to xml file to read
- **group_labels** (*Dict[str, str]*) – A dictionary of group labels that convert values stored in xml PartOfGroup (keys) to required label (values). e.g {"Tumor": "tumor", "Metastasis": "tumor", "Normal": "normal", "Tissue": "normal"}

Return type
List[Annotation]

wsipipe.load.datasets package

loader module

camelyon16 module

stripai module

registry module

wsipipe.load.slides package

slide module

SlideBase is a parent class that contains functionality for reading slides. This is used to render different types of slides into a common format

class SlideBase

Bases: object

Generic base class for slide loaders.

open()

opens a slide

Return type
None

close()

closes a slide

Return type
None

path()

returns the filepath to the slide

dimensions()

returns a list of the slide dimensions in pixels

for each level present in the WSI pyramid

read_region()

returns a specified region of the slide as a PIL image

Parameters
region (Region) –

Return type
Image

read_regions()

returns multiple regions as a list of PIL images

Parameters
regions (*List[Region]*) –

Return type
List[*Image*]

get_thumbnail()

returns the whole of the slide at a given level

Parameters
level (*int*) –

Return type
numpy.array

in the WSI pyramid as numpy array. This can run out of memory if

too low a level in the pyramid is selected

abstract close()

Return type
None

abstract property dimensions: List[*Size*]

Gets slide dimensions in pixels for all levels in pyramid :returns: A list of sizes :rtype: (List[Size])

get_thumbnail(*level*)

Get thumbnail of whole slide downsized to a level in the pyramid

Parameters
level (*int*) – Level at which to return thumbnail

Returns
thumbnail as an RGB numpy array

Return type
im (np.array)

abstract open()

Return type
None

abstract property path: Path

abstract read_region(*region*)

Read a region from a WSI

Returns a PIL image for the region

Parameters
region (Region) – A region of the image

Returns
A PIL Image of the specified region

Return type
image (Image)

abstract read_regions(*regions*)

Read multiple regions of a WSI

Returns a PIL image for each region

Parameters
regions (*List[Region]*) – List of regions

Returns
List of Images

Return type
images (List[Image])

openslide module

class OSSlide(*path*)

Bases: *SlideBase*

Read slides to generic format using the openslide package. For example, to open OMETiff WSIs.

Parameters
path (*Path*) –

check_level(*region*)

Checks if level specified in region exists in pyramid :param region: A Region to check :type region: Region

Returns
True if level in region exists in pyramid

Return type
(bool)

Parameters
region (Region) –

close()

Return type
None

convert_region(*region*)

Creates a PIL image of a region by downsampling from lower level :param region: A Region to create :type region: Region

Returns
A downsampled PIL Image

Return type
image (Image)

Parameters
region (Region) –

property dimensions: List[*Size*]

Gets slide dimensions in pixels for all levels in pyramid

If fewer than 10 levels exist in the pyramid it calculates the extra sizes and adds them to the list

Returns
A list of sizes

Return type
sizelist (List[*Size*])

open()

Return type
None

property path: Path

read_region(*region*)

Read a region from a WSI

Checks if the specified level for the region exists in the pyramid. If not reads the region from the highest level that exists and downscales it

Parameters
region (Region) – A region of the image

Returns
A PIL Image of the specified region

Return type
image (Image)

`read_regions`(*regions*)

Read multiple regions of a WSI

Returns a PIL image for each region

Parameters
regions (*List[Region]*) – List of regions

Returns
List of Images

Return type
images (List[Image])

region module

`class Region`(*level*, *location*, *size*)

Bases: `tuple`

Class for a Region of a whole slide image :param level: Level to extract the region :type level: int :param location: x y tuple giving location of top left of region at that level :type location: Point :param size: width and height tuple giving size of region at that level :type size: Size

Parameters

- **level** (*int*) –
- **location** (Point) –
- **size** (Size) –

`as_values`()

Splits out location and size into separate values

Return type
Tuple[int, int, int, int, int]

`property level`

Alias for field number 0

`property location`

Alias for field number 1

`classmethod make`(*x*, *y*, *size*, *level*)

An alternate construction method for square region

Assumes a square region of width and height equal to size

Parameters

- **x** (*int*) – the pixel location of left of image at level
- **y** (*int*) – the pixel location of top of image at level

- **size** (*int*) – size of square region
- **level** (*int*) – Level to extract the region

property size

Alias for field number 2

2.1.3 wsipipe.preprocess package

Contains functionality to split slides into patches, sample patches and apply tissue detection.

Subpackages

wsipipe.preprocess.patching package

patch_finder module

Patch Finders describe how patches are created for a slide.

They work on a labelled image, that is a numpy array with integers giving the annotation category for each pixel.

The input labelled image can be at any level of the pyramid for which a numpy array for that size can fit into memory.

A patch finder will create a dataframe with columns x, y, label where x and y represents the top left corner of the patch and label is the label applied to the patch.

class GridPatchFinder(*labels_level*, *patch_level*, *patch_size*, *stride*, *border=0*, *jitter=0*, *remove_background=True*, *pool_mode='max'*)

Bases: *PatchFinder*

Parameters

- **labels_level** (*int*) –
- **patch_level** (*int*) –
- **patch_size** (*int*) –
- **stride** (*int*) –
- **border** (*int*) –
- **jitter** (*int*) –
- **remove_background** (*bool*) –
- **pool_mode** (*str*) –

labels_level()

class PatchFinder

Bases: `object`

Generic patch finder class

Parameters

- **labels_image** (*np.array*) – The whole slide image represented as a 2d numpy array, the classification is given by an integer. For example an image such as those output by AnnotationSet.render

- **slide_shape** (Size) – The size of the WSI at the level at which the labels are rendered. This may be different to the labels image shape, as the labels image may not include blank parts of the slide in the bottom right.

abstract property labels_level

class RandomPatchFinder(*labels_level*, *patch_level*, *patch_size*, *border=0*, *npatches=1000*, *pool_mode='mode'*)

Bases: PatchFinder

Parameters

- **labels_level** (*int*) –
- **patch_level** (*int*) –
- **patch_size** (*int*) –
- **border** (*int*) –
- **npatches** (*int*) –
- **pool_mode** (*str*) –

labels_level()

patchset module

patchset_utils module

wsipipe.preprocess.sample package

sampler module

wsipipe.preprocess.tissue_detection package

tissue_detector module

Tissue Detectors create a 2d array of booleans indicating if that area contains tissue or not.

The input is an RGB numpy array representing the slide. Usually a downsampled thumbnail image as whole slide images as level 0 are often too large to store in memory.

class TissueDetector(*pre_filter=<wsipipe.preprocess.tissue_detection.filters.NullBlur object>*, *morph_transform=<wsipipe.preprocess.tissue_detection.morphology_transforms.NullTransform object>*)

Bases: object

Generic tissue detector class

Parameters

- **pre_filter** (*Union[PreFilter, List[PreFilter]]*) – Any filters or transforms that are to be applied before the tissue detection. Can be lists of filters or individual filters. Defaults to NullBlur
- **()** (*morph_transform*) – Any filters or transforms that are to be applied after the tissue detection. Can be lists of transforms or individual transforms. Defaults to NullTransform

- **morph_transform** (*Union[MorphologyTransform, List[MorphologyTransform]]*) –

Returns
An ndarray of booleans with the same dimensions as the input image True means foreground, False means background

class TissueDetectorAll(*pre_filter=<wsipipe.preprocess.tissue_detection.filters.NullBlur object>*, *morph_transform=<wsipipe.preprocess.tissue_detection.morphology_transforms.NullTransform object>*)

Bases: *TissueDetector*

Parameters

- **pre_filter** (*Union[PreFilter, List[PreFilter]]*) –
- **morph_transform** (*Union[MorphologyTransform, List[MorphologyTransform]]*) –

class TissueDetectorGreyScale(*pre_filter=<wsipipe.preprocess.tissue_detection.filters.NullBlur object>*, *morph_transform=<wsipipe.preprocess.tissue_detection.morphology_transforms.NullTransform object>*, *grey_level=0.8*)

Bases: *TissueDetector*

Parameters
grey_level (*float*) –

class TissueDetectorOTSU(*pre_filter=<wsipipe.preprocess.tissue_detection.filters.NullBlur object>*, *morph_transform=<wsipipe.preprocess.tissue_detection.morphology_transforms.NullTransform object>*)

Bases: *TissueDetector*

Parameters

- **pre_filter** (*Union[PreFilter, List[PreFilter]]*) –
- **morph_transform** (*Union[MorphologyTransform, List[MorphologyTransform]]*) –

filters module

Filters to apply to images as part of tissue detection

class GaussianBlur(*sigma*)

Bases: *PreFilter*

Applies a Gaussian filter with sigma value

Parameters
sigma (*int*) –

class MedianBlur(*filter_size*)

Bases: *PreFilter*

Applies a median filter of size filter_size

Parameters
filter_size (*int*) –

class NullBlur

Bases: *PreFilter*

Null filter does nothing

class PreFilter

Bases: object

Generic class of filter

morphology_transforms module

Transforms can be applied to binary or labelled images, for example to fill holes

class FillHolesTransform(*level_in*, *hole_size_to_fill=250*, *level_zero_size=0.25*)

Bases: *MorphologyTransform*

Fills holes in an image, using segmentation Segments smaller than hole_size_to_fill in area are filled. Size of a pixel at the image level is 2**level_in * level zero size Hole_size_to_fill (an area) is converted to number of pixels by dividing by the size of rpixel at image level.

Input image is a binary image Image is segmented using scikit image regionprops. If the area of the region is less than the specified hole size and the mean intensity of the region is less than 0.1 (out of 1) then the region is filled by converting to True/1/white

Args: level_in: level of input image hole_size_to_fill: dark areas smaller in size than this will be filled level_zero_size: size of a pixel at level zero

Parameters

- **level_in** (*int*) –
- **hole_size_to_fill** (*float*) –
- **level_zero_size** (*float*) –

class MaxPoolTransform(*level_in*, *level_out*)

Bases: *MorphologyTransform*

Applies max pool Takes a big input image and returns a smaller output image. Every pixel in the output image represents 2**(level_out - level_in) pixels in input image. The pixel value for the output image is the maximum of the pixels in that region of the input image.

Args: level_in: Initial level of image level_out: Output level of image (must be a smaller image level_out > level_in)

Parameters

- **level_in** (*int*) –
- **level_out** (*int*) –

class MorphologyTransform

Bases: object

class NullTransform

Bases: *MorphologyTransform*

class SimpleClosingTransform

Bases: *MorphologyTransform*

class SimpleOpeningTransform

Bases: *MorphologyTransform*

class SizedClosingTransform(*level_in*, *expand_size=50*, *level_zero_size=0.25*)

Bases: *MorphologyTransform*

Parameters

- **level_in** (*int*) –
- **expand_size** (*float*) –
- **level_zero_size** (*float*) –

visualise module

2.1.4 wsipipe.utils package

Utility functions that are used throughout the package.

convert module

Functionality for converting between formats.

invert(*d*)

Parameters
d (*Dict*) –

Return type
Dict

np_to_pil(*arr*)

Convert a Numpy array into a PIL image

Parameters
arr (*numpy.ndarray*) – a Numpy array

Returns
the PIL image

Return type
Image

pil_to_np(*image*)

Convert a PIL image into a Numpy array

Parameters
image (*Image*) – the PIL image

Returns
a Numpy array

Return type
numpy.ndarray

remove_item_from_dict(*dict_in*, *key_to_remove*)

remove one key value pair from a dictionary by specifying the key to remove :param dict_in: dictionary to remove an item from :param key_to_remove: the key of the key value pair to be removed

Returns
the dictionary without the specified item

Parameters

- **dict_in** (*dict*) –
- **key_to_remove** (*str*) –

Return type
dict

to_frame_with_locations(*array*, *value_name='value'*)

Create a data frame with row and column locations for every value in the 2D array :param array: a Numpy array :param value_name: a string with the column name for the array values to be output in

Returns
a pandas data frame of row, column, value where each value is the value of np array at row, column

Parameters

- **array** (*numpy.ndarray*) –
- **value_name** (*str*) –

Return type
pandas.DataFrame

filters module

pool2d(*A*, *kernel_size*, *stride*, *padding*, *pool_mode='max'*)

2D Pooling Taken from https://stackoverflow.com/questions/54962004/implement-max-mean-poolingwith-stride-with-numpy :param A: input 2D array :param kernel_size: int, the size of the window :param stride: int, the stride of the window :param padding: int, implicit zero paddings on both sides of the input :param pool_mode: string, 'max' or 'avg'

geometry module

class Address(*row*, *col*)

Bases: tuple

a row and column point

Parameters

- **row** (*int*) –
- **col** (*int*) –

property col

Alias for field number 1

property row

Alias for field number 0

class Point(*x*, *y*)

Bases: tuple

an x y point in integers

Parameters

- **x** (*int*) –
- **y** (*int*) –

property x

Alias for field number 0

property y

Alias for field number 1

class PointF(*x*, *y*)

Bases: tuple

an x y point in floating numbers

Parameters

- **x** (*float*) –
- **y** (*float*) –

property x

Alias for field number 0

property y

Alias for field number 1

class Shape(*num_rows*, *num_cols*)

Bases: tuple

chape given by rows and columns

Parameters

- **num_rows** (*int*) –
- **num_cols** (*int*) –

as_size()

property num_cols

Alias for field number 1

property num_rows

Alias for field number 0

class Size(*width*, *height*)

Bases: tuple

size given by width and height

Parameters

- **width** (*int*) –
- **height** (*int*) –

as_shape()

property height

Alias for field number 1

property width

Alias for field number 0

CHAPTER

THREE

OTHER INFORMATION

3.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/StAndrewsMedTech/wsipipe/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

wsipipe could always use more documentation, whether as part of the official wsipipe docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/StAndrewsMedTech/wsipipe/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.1.2 Get Started!

Ready to contribute? Here's how to set up *wsipipe* for local development.

1. Fork the *wsipipe* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/wsipipe.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv wsipipe
$ cd wsipipe/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 wsipipe tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/StAndrewsMedTech/wsipipe/pull_requests and make sure that the tests pass for all supported Python versions.

3.1.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_wsipipe
```

3.1.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

3.2 Credits

3.2.1 Development Lead

- Christina Fell <cmf21@st-andrews.ac.uk>
- David Morrison <dm236@st-andrews.ac.uk>

3.2.2 Contributors

None yet. Why not be the first?

3.3 History

3.3.1 0.1.0 (2022-09-08)

- First release on PyPI.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W

INDEX

X

Y