dicom_csv

NeuroML

Jun 12, 2023

CONTENTS

This is a collection of utils for gathering, aggregation and handling metadata from DICOM files.

CHAPTER

ONE

CONTENTS

1.1 Crawling

Contains functions for gathering metadata from individual DICOM files or entire directories.

dicom_csv.crawler.**get_file_meta**(*path: Union[Path, str]*, *force: bool = True*, *read_pixel_array: bool = False*, *unpack_volumetric: bool = False*, *extract_private: bool = False*) → Iterable[dict]

Get a dict containing the metadata from the DICOM file located at `path`.

Parameters
PathLike (*path* -) – full path to file

:param : full path to file :param force - bool: pydicom.filereader.dcmread force parameter, default is False :param : pydicom.filereader.dcmread force parameter, default is False :param read_pixel_array - bool: if True, crawler will add information about DICOM pixel_array, False significantly increases crawling time,

default is True.

:param
[if True, crawler will add information about DICOM pixel_array, False significantly increases crawling time,] default is True.

Notes

The following keys are added:

NoError: whether an exception was raised during reading the file.
HasPixelArray: (if NoError is True) whether the file contains a pixel array.
PixelArrayShape: (if HasPixelArray is True) the shape of the pixel array.

For some formats the following packages might be required:

```
>>> conda install -c glueviz gdcm # Python 3.5 and 3.6
>>> conda install -c conda-forge gdcm # Python 3.7
```

dicom_csv.crawler.**join_tree**(*top: Union[Path, str]*, *ignore_extensions: Sequence[str] = ()*, *relative: bool = True*, *verbose: int = 0*, *read_pixel_array: bool = False*, *force: bool = True*, *unpack_volumetric: bool = True*, *extract_private: bool = False*, *total: bool = False*) → DataFrame

Returns a dataframe containing metadata for each file in all the subfolders of `top`.

Parameters

- **top** (*PathLike*) – path to crawled folder
- **ignore_extensions** (*Sequence*) – list of extensions to skip during crawling
- **relative** (*bool*) – whether the PathToFolder attribute should be relative to top default is True.
- **verbose** (*int*) –

  the verbosity level:

  0 - no progressbar
  1 - progressbar with iterations count
  2 - progressbar with filenames
- **total** (*bool*) – whether to show the total number of files in the progressbar. This is adds a bit of overhead, because each file will be visited a second time (without being opened).

References

See the *Working with DICOM files* tutorial for more details.

Notes

The following columns are added:

NoError: whether an exception was raised during reading the file.
HasPixelArray:(if NoError is True) whether the file contains a pixel array(added if read_pixel_array is True).
PixelArrayShape: (if HasPixelArray is True) the shape of the pixel array (added if read_pixel_array is True).
PathToFolder
FileName

For some formats the following packages might be required:

```
>>> conda install -c glueviz gdcm # Python 3.5 and 3.6
>>> conda install -c conda-forge gdcm # Python 3.7
```

1.2 Aggregation

Tools for grouping DICOM metadata into images.

dicom_csv.aggregation.**aggregate_images**(*metadata: DataFrame*, *by: Union[str, Sequence[str]]*, *process_series: Optional[Callable] = None*) → DataFrame

Groups DICOM metadata into images (series).

Parameters

- **metadata** – a dataframe with metadata returned by *join_tree*.

- **by** – a list of column names by which the grouping will be performed. Default columns are: PatientID, SeriesInstanceUID, StudyInstanceUID, PathToFolder, PixelArrayShape, SequenceName.
- **process_series** – a function that processes an aggregated series before it will be joined into a single entry

References

See the *Working with DICOM files* tutorial for more details.

Notes

The following columns are added:

SlicesCount: the number of files/slices in the image.
FileNames: a list of slash ("/") separated file names.
InstanceNumbers: (if InstanceNumber is in columns) a list of comma separated InstanceNumber values.

The following columns are removed:
FileName (replaced by FileNames), InstanceNumber (replaced by InstanceNumbers), any other columns that differ from file to file.

dicom_csv.aggregation.**normalize_identifiers**(*metadata: DataFrame*) → DataFrame

Converts PatientID to str and fills nan values in SequenceName.

Notes

The input dataframe will be mutated.

dicom_csv.aggregation.**select**(*dataframe: DataFrame*, *query: str*, ***where: str*) → DataFrame

1.3 Loading

dicom_csv.misc.**get_image**(*instance: Dataset*, *to_color_space: Optional[str] = None*)

dicom_csv.misc.**stack_images**(*series: Sequence[Dataset]*, *axis: int = -1*, *to_color_space: Optional[str] = None*)

1.4 Spatial operations

dicom_csv.spatial.**get_orientation_matrix**(*series: Sequence[Dataset]*) → ndarray

Returns a 3 x 3 orthogonal transition matrix from the image-based basis to the patient-based basis. Rows are coordinates of image-based basis vectors in the patient-based basis. Columns are coordinates of patient-based basis vectors in the image-based basis vectors.

See https://dicom.innolitics.com/ciods/rt-dose/image-plane/00200037 for details.

dicom_csv.spatial.**get_slice_plane**(*instance: Dataset*) → *Plane*

dicom_csv.spatial.**get_slices_plane**(*series: Sequence[Dataset]*) → *Plane*

class dicom_csv.spatial.**Plane**(*value*)

Bases: Enum

An enumeration.

Sagittal = 0

Coronal = 1

Axial = 2

dicom_csv.spatial.**order_series**(*series: Sequence[Dataset]*, *decreasing: bool = True*) → Sequence[Dataset]

dicom_csv.spatial.**get_slice_locations**(*series: Sequence[Dataset]*) → ndarray

Computes slices location from ImagePositionPatient. NOTE: the order of slice locations can be both increasing or decreasing for ordered series (see order_series).

dicom_csv.spatial.**locations_to_spacing**(*locations: Sequence[float]*, *max_delta: float = 0.1*, *errors: bool = True*) → float

dicom_csv.spatial.**get_slice_spacing**(*series: Sequence[Dataset]*, *max_delta: float = 0.1*, *errors: bool = True*) → float

Returns constant distance between slices of a series. If the series doesn't have constant spacing - raises ValueError if `errors` is True, returns `np.nan` otherwise.

dicom_csv.spatial.**get_pixel_spacing**(*series: Sequence[Dataset]*) → Tuple[float, float]

Returns pixel spacing (two numbers) in mm.

dicom_csv.spatial.**get_voxel_spacing**(*series: Sequence[Dataset]*) → Tuple[float, float, float]

Returns voxel spacing: pixel spacing and distance between slices' centers.

dicom_csv.spatial.**get_image_position_patient**(*series: Sequence[Dataset]*) → ndarray

Returns ImagePositionPatient stacked into array.

dicom_csv.spatial.**drop_duplicated_slices**(*series: Sequence[Dataset]*, *tolerance_hu=1*) → Sequence[Dataset]

dicom_csv.spatial.**orientation_matrix_to_slices_plane**(*om: ndarray*) → *Plane*

dicom_csv.spatial.**get_slice_orientation**(**args*, ***kwds*)

get_slice_orientation is deprecated!

dicom_csv.spatial.**get_slices_orientation**(*series: Sequence[Dataset]*) → *SlicesOrientation*

get_slices_orientation is deprecated!

class dicom_csv.spatial.**SlicesOrientation**(*transpose: bool*, *flip_axes: tuple*)

Bases: tuple

Defines how slices should be transformed in order to be canonically oriented: First transpose slices if `transpose == True`. Then flip slices along `flip_axes` (they already account for transposition).

property transpose

Alias for field number 0

property flip_axes

Alias for field number 1

dicom_csv.spatial.**orientation_matrix_to_slices_orientation**(**args*, ***kwds*)

orientation_matrix_to_slices_orientation is deprecated!

dicom_csv.spatial.**get_axes_permutation**(**args*, ***kwds*)

get_axes_permutation is deprecated!

dicom_csv.spatial.**get_flipped_axes**(**args*, ***kwargs*)

<lambda> is deprecated!

dicom_csv.spatial.**get_image_plane**(*series: Sequence[Dataset]*) → *Plane*

get_image_plane is deprecated!

dicom_csv.spatial.**restore_orientation_matrix**(*metadata: Union[Series, DataFrame]*)

Fills nan values (if possible) in `metadata`'s ImageOrientationPatient* rows.

Required columns: ImageOrientationPatient[0-5]

Notes

The input dataframe will be mutated.

1.5 Console scripts

This library contains a console script around *join_tree*, which is added to your namespace after installation:

```
dicom-csv folder/with/dicoms path/to/metadata.csv

# pass --help for more details:
dicom-csv --help
```

1.6 Working with DICOM files

Before we start analysing our files, let's install some additional libraries, which add support for various medical imaging formats:

```
conda install -c glueviz gdcm # Python 3.5 and 3.6
conda install -c conda-forge gdcm # Python 3.7
```

We'll be working with a subset of the `CT Lymph Nodes` dataset which can be downloaded here.

```
path = '~/dicom_data/'
```

1.6.1 Crawling

`join_tree` is the main function that collects the DICOM files' metadata:

```
from dicom_csv import join_tree

df = join_tree(path, relative=True, verbose=False)
```

It recursively visits the subfolders of `path`, also it adds some additional attributes: `NoError`, `HasPixelArray`, `PathToFolder`, `FileName`:

```
len(df), df.NoError.sum(), df.HasPixelArray.sum()
```

```
(2588, 2587, 2587)
```

Thre resulting dataframe has 2588 files' metadata in it, and only one file was openned with errors, let's check which one:

```
df.loc[~df.NoError, ['FileName', 'PathToFolder']]
```

There is a file `readme.txt` in the root of the folders tree, which is obvisously not a DICOM file.

Note that `PathToFolder` is relative to `path`, this is because we passed `relative=True` to `join_tree`.

```
# leave only dicoms that contain images (Pixel Arrays)
dicoms = df[df.NoError & df.HasPixelArray]

dicoms.FileName[1], dicoms.PathToFolder[1]
```

```
('000466.dcm',
 'ABD_LYMPH_001/09-14-2014-ABDLYMPH001-abdominallymphnodes-30274/abdominallymphnodes-
↪26828')
```

1.6.2 Aggregation

Next, we can join the dicom files into series, which are often easier to operate with:

```
from dicom_csv import aggregate_images

images = aggregate_images(dicoms)
len(images)
```

```
4
```

`aggregate_images` also adds some attributes: `SlicesCount`, `FileNames`, `InstanceNumbers`, check its docstring for more information.

For example `FileNames` contains all the files that are part of a particular series:

```
images.FileNames[0][:50] + '...'
```

```
'000466.dcm/000312.dcm/000150.dcm/000357.dcm/000311...'
```

As you can see, they are not ordered by default, but you can change this behaviour by passing the `process_series` argument which receives a subset of the dataframe, containing files from the same series:

```
images = aggregate_images(dicoms, process_series=lambda series: series.sort_values(
↪'FileName'))

images.FileNames[0][:50] + '...'
```

```
'000000.dcm/000001.dcm/000002.dcm/000003.dcm/000004...'
```

1.6.3 Loading

You can load a particular series' images stacked into a numpy array using the following function:

```
img = load_series(images.loc[0], path)
```

it expects a row from the aggregated dataframe and, optionally, the `path` argument, if the paths are relative.

The image's orientation as well as the slices' order can be determined automatically, if you pass `orientation=True`:

```
img = load_series(images.loc[0], path, orientation=True)
```

```
print(img.shape, images.PixelArrayShape[0], images.SlicesCount[0])
```

```
(512, 512, 661) 512,512 661
```

1.7 Loading contours stored in DICOM format (RTstructure).

Segmentation mask is stored in DICOMs in as a set of plain contours. These contours are nothing but a 2D curves defined by set of coordinates in 3D space. Specifically these coordinates are in physical space not in voxel space. Every single RTStructure might contain contours for multiple masks (e.g. `Brain`, `Left_Eye`, `GTV` etc.).

`dicom-csv` contains several functions to collect contours stored in RTstructures and move them into image's voxel space.

1.7.1 Crawling images folder

`join_tree` is the main function that collects the DICOM files' metadata:

```
from dicom_csv import join_tree

df = join_tree(path, relative=False, verbose=False)
```

1.7.2 Select rows related to RTStructures

`collect_rtstruct` is the function which selects rows related to RTstructures and copies additional metadata from corresponding images (RTstructure itself does not contain information about spatial orientation, it only contains coordinates and link to corresponding DICOM image):

```
from dicom_csv.rtstructs.csv import collect_rtstructs

df_rtstructs = collect_rtstructs(df)
```

1.7.3 Extract contours coordinates from RTStructure

`contours_dict` extract all coordinates from a single row of `df_rtstructs`:

```
from dicom_csv.rtstruct.contour import read_rtstruct

contours_dict = read_rtstruct(df_rtstructs.iloc[0])
```

It outputs python dictionary with keys - names of the contours and values - `n x 3` nd.arrays of coordinates in physical space.

1.7.4 Move contours to voxel space

Finally, `contours_image_dict` moves these coordinates into voxel space (basis change transformation, essentially rotation, translation and stretching):

```
from dicom_csv.rtstruct.contour import contours_image_dict

contours_image_dict = contours_to_image(df_rtstructs.iloc[0], contours_dict)
```

Resulting dictionary contains all contours stored in corresponding RTStructure.

1.7.5 Contours to masks

TODO

PYTHON MODULE INDEX

d

INDEX

T