dicom-wsi

Release 1.0.0

Mar 06, 2023

Contents:

CHAPTER 1

dicom_wsi

Package for converting whole slide image files to DICOM.

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

CHAPTER 2

Usage

First, you need to install dicom_wsi and its dependencies. See this link for details.

To use dicom-wsi:

```
python cli.py -w <WSI File path> -o <OutputDirectory> -p <output file prefix> -y yaml/
↪base.yaml
```

That's it! Most of the time you wan't need to change anything. But if you do, please see the example yaml file.

2.1 Features

- Validate DICOM elements using pydicom
- Output format DICOM formatted files (vetted with dciodvfy)

2.2 TODO

- Find out how to determine what *FileMetaInformationGroupLength* should be

2.3 Credits

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

CHAPTER 3

Installation

Important: The libvips and OpenSlide packages are required, but not available on PyPi. You will need to make sure your environment has these packages available.

Important: Windows is *VERY* finicky with OpenSlide, and it's not always straightforward. The recommended way for windows users is through Docker.

3.1 With Conda (Preferred)

The preferred method uses miniconda because there are several necessary modules that cannot be installed with pip. To install miniconda:

```
$ sudo apt-get update
$ wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh -O␣
↪miniconda.sh;
$ bash miniconda.sh -b -p $HOME/miniconda
$ source "$HOME/miniconda/etc/profile.d/conda.sh"
$ hash -r
$ conda config --set always_yes yes --set changeps1 no
$ conda update -q conda
```

Once conda is installed, then you can install *dicom_wsi*:

```
$ conda config --add channels bioconda
$ conda config --add channels conda-forge
$ conda create -q -n test-environment python pyvips openjpeg libtiff
$ conda activate test-environment
$ pip install -U -r requirements_dev.txt
```

If using iSyntax files, you also need libtiff http://download.osgeo.org/libtiff/tiff-4.1.0.zip

3.2 Stable release

To install dicom-wsi, run this command in your terminal (assuming you already have the non-pip installed libraries):

```
$ pip install dicom_wsi
```

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

3.3 From sources

The sources for dicom-wsi can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/Steven-N-Hart/dicom_wsi
```

Or download the tarball:

```
$ curl -OJL https://github.com/Steven-N-Hart/dicom_wsi/tarball/master
```

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

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

3.4 With Docker

You can also build a container using Docker:

```
$ docker build -t stevennhart/dicom_wsi .
```

3.5 Development with PyCharm

If you are going to do some development work with PyCharm, you will need to copy the binary files into your venv.

CHAPTER 4

TL;DR

To use dicom-wsi in a project, you can run in one of two ways. You can run the command line program,

```
python cli.py -w <WSI File path> -o <OutputDirectory> -p <output file prefix> -y yaml/
↪base.yaml
```

Or you can run it directly from python

```
import dicom_wsi
dicom_wsi.dicom_wsi.create_dicom(cfg, pools=n_pools)
```

The *cfg* is the dictionary of required entities, and n_pools defines the number of threads to use.

That's it!

CHAPTER 5

Getting Started

Before you can run *dicom_wsi*, you first need to gather some input data necessary for creating a valid DICOM file. There are to ways to do this: 1.) Create a YAML file or 2.) Build a dictionary yourself.

5.1 Create a YAML file

The *YAML* file contains two sections: *General* and *BaseAttributes*. The *General* section contains a fixed number of required fields that are used by *dicom_wsi*, whereas the *BaseAttributes* are restricted key words from the DICOM standard. These are cross-referenced with the pydicom dictionary.

5.1.1 *General* section

Definitions

Term	Definition
WSIFile	Path to whole slide image file
OutFilePrefix	What prefix to use when saving the DICOM output file
NumberOfLevels	How many levels should be extracted (in powers of 2)
OrgUIDRoot	Your organizations UID root prefix
FrameSize	How many pixels should be used for DICOM frames
MaxFrames	Number of frames allowed before writing to a new file

5.1.2 *BaseAttributes* section

Most of the terms in this section are defined in the DICOM standard. Many will not need to be changed, but some always will. Below, I highlight those terms that will likely need to be manually set. **Definitions**

Term	Definition
PatientName	LastName^FirstName
PatientBirthDate	Date format (i.e. 20000101)
PatientSex	M: Male, F: Female, O: Other
PatientID	Unique identifier for the patient
ReferringPhysicianName	LastName^FirstName
StudyDate	Date format
StudyID	Human readable study name

A full example can be found in *yaml/base.yaml*.

5.2 Without a YAML file

While a *YAML* file is recommended, you don't actually need one. You could choose to make the dictionary yourself. The dictionary has two nested components, *General* and *BaseAttributes*, each of which has the elements defined in *yaml/base.yaml*.

5.3 Using in existing code

To use dicom-wsi in a project:

```
from yaml import load, BaseLoader
import dicom_wsi
dwsi = dicom_wsi.dicom_wsi
get_wsi = dicom_wsi.parse_wsi.get_wsi

# Define your YAML file
my_yaml = '/path/to/yaml'
# Load your YAML file
cfg = load(open(my_yaml), Loader=BaseLoader)
# Read the WSI, updating the config with information contained in the slide
cfg, wsi = get_wsi(cfg)
# Create DICOM files
dwsi.create_dicom(cfg)
```

5.4 Sample RUN

This Step will download the sample svs file python ./tests/__init__.py

This is sample execution python cli.py -y ./tests/testfiles/base.yaml

CHAPTER 6

Examples

Here is an end to end example

Clone the git repo

```
$ git clone https://github.com/Steven-N-Hart/dicom_wsi.git
    $ cd dicom_wsi
    $ mkdir example
    $ cd example
```

Install the required packages as described here

Downloading the svs file

```
$ wget http://openslide.cs.cmu.edu/download/openslide-testdata/Aperio/CMU-1-JP2K-
↪33005.svs
```

getting the annotations file

```
$ cp ../tests/CMU-1-JP2K-33005.xml .
```

Getting the input yaml file to generate the dicom file.Modifying values for params 'WSI-File','OutFilePrefix','Annotations'.

Annotations are optional, if you want to skip annotations, then remove 'Annotations' param in the base.yaml file Below command will replace the paths for params 'WSIFile','OutFilePrefix','Annotations' to current directory

```
$ cat ../dicom_wsi/yaml/base.yaml |sed -e 's/tests\///g'|sed -e 's/.\///g' > base.yaml
```

Running the dicom_wsi tool & generating the dicom files

```
$ python ../dicom_wsi/cli.py -y base.yaml
```

Following dicom files will be generated (Multiple dicom files for multiple levels).

```
$ ls output.*.dcm
```

output.0-10.dcm
output.0-2.dcm
output.0-6.dcm
output.1-1.dcm
output.3-0.dcm
output.0-11.dcm
output.0-3.dcm
output.0-7.dcm
output.1-2.dcm
output.4-0.dcm
output.0-12.dcm
output.0-4.dcm
output.0-8.dcm
output.1-3.dcm
output.5-0.dcm
output.0-1.dcm
output.0-5.dcm
output.0-9.dcm
output.2-0.dcm
output.6-0.dcm

Optional: Validating the generated dicom files.

Download this tool dciodvfy to validate the generated dicom files

6.1 Other functions

Extracting Annotations from Dicom file to a python dictionary(Here i'm running it on only one level Dicom file)

```
$ python  ../dicom_wsi/mods/extract_annotations.py -D output.2-0.dcm
```

Extracting images from Dicom file

```
$ python  ../dicom_wsi/mods/extract_image_patches.py -D output.2-0.dcm -d output_
↪images
```

CHAPTER 7

Working with iSyntax files

I you are like me and are stuck with iSyntax files, then they first need to be converted to TIFF. The following script will parse the isyntax file to make a config file and a BigTiff file.

First, change directories so you are in the isyntax directory of this repo , and download an example iSyntax file from the Phillips Website.

```
$ cd isyntax
$ curl -o ex1.isyntax  https://zenodo.org/record/5037046/files/testslide.isyntax?
↪download=1
```

Next, get the Phillips SDK. You need to create a log in first.

You will need to follow the installation instructions for your specific operating system. Once you have it installed, open a python terminal and run:

```
import pixelengine
```

If you get *ModuleNotFoundError: No module named 'pixelengine'* then you do not have this installed properly. Ask Phillips tech support for help.

> Note you need to ensure that you have the Phillips SDK installed and available. It is not possible for this toolkit

Once you have the Phillips SDK installed, you can run the conversion script.

```
$ python isyntax_to_tiff.py --input 1.isyntax --tif BIGTIFF --sparse 0  --startlevel 0
```

This will create a file called *ex1_BIG_sparse.tiff*.

Now you can create the configuration file and proceed as normal.

> Note: Since the *isyntax_to_tiff.py* is maintained by Phillips, its name or usage might change. Please consult the Phillips documentation.

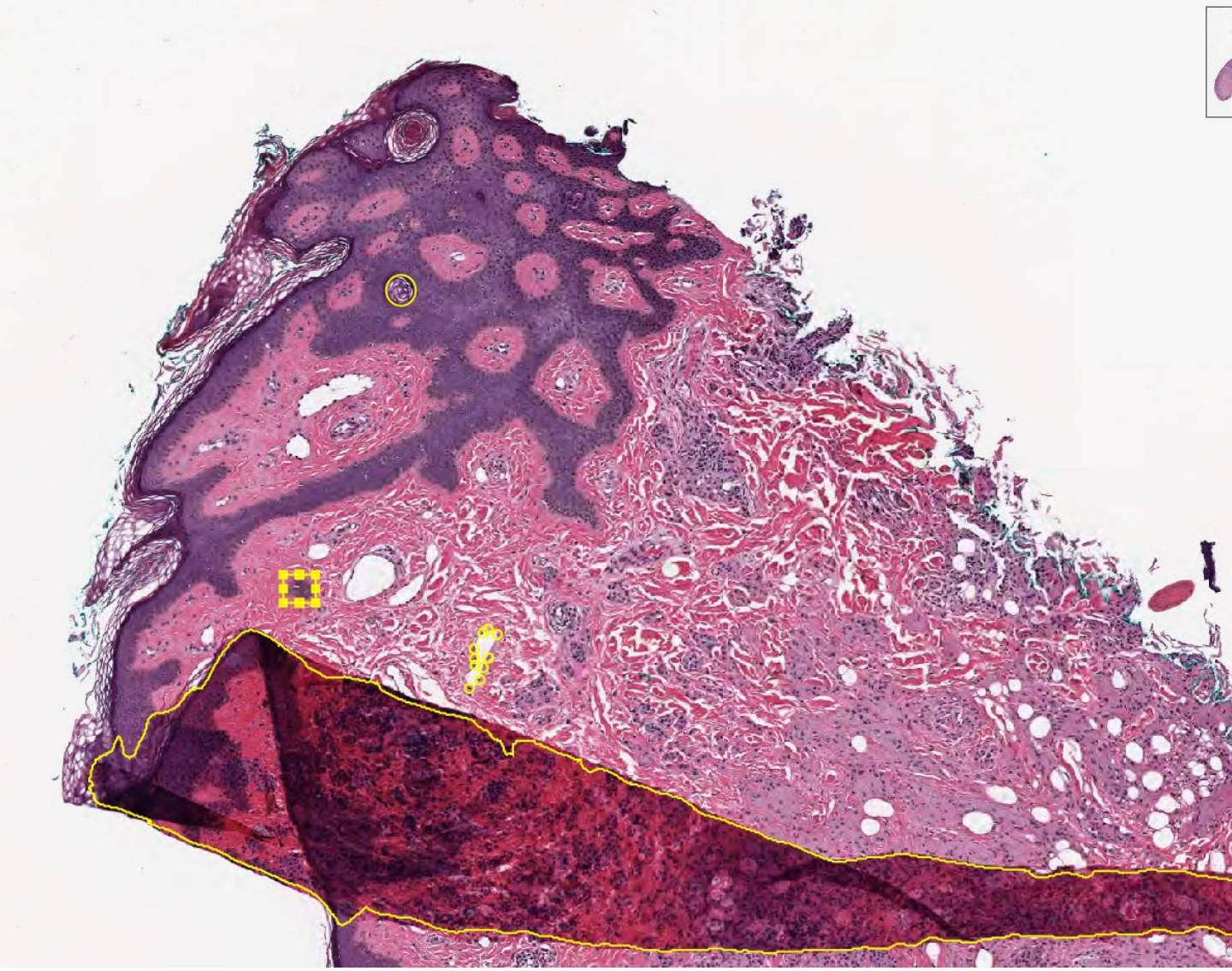
CHAPTER 8

Annotations

Given an XML file of annotations, extract the data into the appropriate DICOM element.

If you have Annotations in the XML file structure listed below, and you want to include them in your DICOM file, then all you need to do is to add the Annotations key and the associated file path to the *General* attribute in the YAML file.

General: Annotations: '/path/to/xmlFile'



8.1 Getting some sample XML data

We have prepared some different annotations on the Aperio example CMU-1-JP2K-33005.svs. These examples were drawn using QuPath and extracted with this script. The annotations have no physiological relevance, only to show how the different data types can be stored inside DICOM. Each of the yellow markups in the image above describe one of the following data types:

- Points
- Rectangle
- Area
- Ellipse

All of the data types should be encoded in an XML tree that looks like the following:

```
<Annotations>
  <Annotation>
    <Regions>
      <Region Text="null" GeoShape="Points">
        ...
      </Region>
    </Regions>
  </Annotation>
</Annotations>
```

The important characteristics here are *Text="null"* and *GeoShape="Points"*. These define a human readable label(Text), and a data type (GeoShape). Each will be discussed below.

8.1.1 Points

The simplest annotation type is the point. It defines as specific x and y coordinate (pixel). Each entry in this group represents an independent data point. If there are points that have different meanings (e.g. mitosis vs lymphocyte), then they should be grouped in a different section with a different *Text* value.

```
<Region Id="1" Type="0" Text="null" GeoShape="Points" Zoom="0.042148" Selected="0"␣
↪ImageLocation="" ImageFocus="0" Length="74565.8" Area="213363186.2" LengthMicrons=
↪"18798.0" AreaMicrons="13560170.4" NegativeROA="0" InputRegionId="0" Analyze="1"␣
↪DisplayId="1">
  <Attributes/>
  <Vertices>
    <Vertex X="37279.312500" Y="4662.189453"/>
    <Vertex X="37319.550781" Y="4588.894043"/>
    <Vertex X="..." Y="..."/>
   </Vertices>
</Region>
```

8.1.2 Rectangle

The next simplest annotation is a rectangle, or bounding box. These annotations define an area that contains an object of interest. They require four different points to describe the boundaries of the x and y corners.

```
<Region Id="2" Type="2" Text="Necrosis" GeoShape="Rectangle" Zoom="0.042148" Selected=
↪"0" ImageLocation="" ImageFocus="0" Length="74565.8" Area="213363186.2"␣
↪LengthMicrons="18798.0" AreaMicrons="13560170.4" NegativeROA="0" InputRegionId="0"␣
↪Analyze="1" DisplayId="1">
  <Attributes/>
  <Vertices>
    <Vertex X="36406.388563" Y="4324.243648"/>
    <Vertex X="36554.076020" Y="4324.243648"/>
    <Vertex X="36554.076020" Y="4452.625555"/>
    <Vertex X="36406.388563" Y="4452.625555"/>
  </Vertices>
</Region>
```

8.1.3 Area

An area annotations structured identically to the bounding box, except that there can be any number of x,y coordinate pairs. This is the annotation typically used for image segmentation.

```
<Region Id="3" Type="1" Text="Fold" GeoShape="Area" Zoom="0.042148" Selected="0"␣
↪ImageLocation="" ImageFocus="0" Length="74565.8" Area="213363186.2" LengthMicrons=
↪"18798.0" AreaMicrons="13560170.4" NegativeROA="0" InputRegionId="0" Analyze="1"␣
↪DisplayId="1">
  <Attributes/>
  <Vertices>
    <Vertex X="36382.175781" Y="4644.585938"/>
    <Vertex X="36389.238281" Y="4651.647949"/>
    ...
    <Vertex X="36262.121094" Y="4573.966309"/>
    <Vertex X="36255.058594" Y="4573.966309"/>
  </Vertices>
</Region>
```

8.1.4 Ellipse

Ellipses are just circular annotations. They have the same structure as Rectangles, but rather than being connected by straight lines in an image viewer, they will instead be connected with curved lines.

```
<Region Id="4" Type="0" Text="null" GeoShape="Ellipse" Zoom="0.042148" Selected="0"␣
↪ImageLocation="" ImageFocus="0" Length="74565.8" Area="213363186.2" LengthMicrons=
↪"18798.0" AreaMicrons="13560170.4" NegativeROA="0" InputRegionId="0" Analyze="1"␣
↪DisplayId="1">
  <Attributes/>
  <Vertices>
    <Vertex X="36943.806573" Y="2957.558623"/>
    <Vertex X="37011.368077" Y="3027.752393"/>
    <Vertex X="36943.806573" Y="3097.946164"/>
    <Vertex X="36876.245069" Y="3027.752393"/>
  </Vertices>
</Region>
```

8.2 Inserting into the DICOM file

This process assumes you have a pydicom object called *ds*. Let's go ahead and build out the base for our annotations.

```
from pydicom.sequence import Sequence
from pydicom.dataset import Dataset

ds = ... # Stuff to create DICOM file
dsDisplayedArea = Dataset()
dsDisplayedArea.PresentationSizeMode = 'TRUE SIZE'
ds.DisplayedAreaSelectionSequence = Sequence([dsDisplayedArea])
ds.GraphicAnnotationSequence = Sequence([])
ds.GraphicAnnotationSequence[0].ReferencedImageSequence = Sequence([])
ds.GraphicAnnotationSequence[0].ReferencedImageSequence[0].GraphicObjectSequence =␣
↪Sequence([])
```

Determine what type of annotation element is needed:

1. Rectangle

```
# Graphics on the first referenced image
GraphicObjectSequence  = Dataset()
GraphicObjectSequence.BoundingBoxTopLeftHandCorner = [36406.388563, 4452.625555]
GraphicObjectSequence.BoundingBoxBottomRightHandCorner = [36554.076020, 4324.243648]
↪# bottom right coordinates of bounding box [max_x, min_y]
GraphicObjectSequence.BoundingBoxAnnotationUnits = 'PIXEL'  # unit of coordinates
GraphicObjectSequence.BoundingBoxHorizontalJustification = 'LEFT'
GraphicObjectSequence.UnformattedTextValue = 'Necrosis'  # Text="Necrosis"
GraphicObjectSequence.GraphicGroupID = '2'  # Id="2"
gos = Sequence([GraphicObjectSequence])
ds.GraphicAnnotationSequence[0].ReferencedImageSequence[0].GraphicObjectSequence.
↪append(gos)
del GraphicObjectSequence
del gos
```

2. Points

```
GraphicObjectSequence  = Dataset()
GraphicObjectSequence.GraphicType = "POINT"
GraphicObjectSequence.NumberofGraphicPoints = 4  # how many points where saved in␣
↪this domain, validate data is complete
GraphicObjectSequence.GraphicData = [37279.312500, 4662.189453, 37319.550781, 4588.
↪894043, ..., ...]   # x,y coordinates of points [x0, y0, x1, y1 ....]
GraphicObjectSequence.GraphicAnnotationUnits = 'PIXEL'  # unit of coordinates
GraphicObjectSequence.GraphicGroupID = '1'  # Id="1" Type="0" Text="null"
gos = Sequence([GraphicObjectSequence ])
ds.GraphicAnnotationSequence[0].ReferencedImageSequence[0].GraphicObjectSequence.
↪append(gos)
del GraphicObjectSequence
del gos
```

3. Area

```
GraphicObjectSequence  = Dataset()
GraphicObjectSequence.GraphicType = "POLYLINE"  # add polyline
GraphicObjectSequence.NumberofGraphicPoints = 4 # how many points where saved in this␣
↪domain
GraphicObjectSequence.GraphicData = [36382.175781, 4644.585938, 36389.238281, 4651.
↪647949, ..., ...]
GraphicObjectSequence.GraphicAnnotationUnits = 'PIXEL'  # unit of coordinates
GraphicObjectSequence.GraphicGroupID = 3  # Annotation Label ID: 2
gos = Sequence([GraphicObjectSequence ])
ds.GraphicAnnotationSequence[0].ReferencedImageSequence[0].GraphicObjectSequence.
↪append(gos)
del GraphicObjectSequence
del gos
```

4. Ellipse

```
GraphicObjectSequence  = Dataset()
GraphicObjectSequence.GraphicType = "ELLIPSE"
GraphicObjectSequence.NumberofGraphicPoints = 4 # how many points where saved in this␣
↪domain
GraphicObjectSequence.GraphicData = [36943.806573, 2957.558623, 37011.368077, 3027.
↪752393, 36943.806573, 3097.946164, 36876.245069, 3027.752393]
GraphicObjectSequence.GraphicAnnotationUnits = 'PIXEL'  # unit of coordinates
GraphicObjectSequence.GraphicGroupID = 3  # Annotation Label ID: 2
```

(continues on next page)

(continued from previous page)

```
gos = Sequence([GraphicObjectSequence ])
ds.GraphicAnnotationSequence[0].ReferencedImageSequence[0].GraphicObjectSequence.
↪append(gos)
del GraphicObjectSequence
del gos
```

CHAPTER 9

Extracting Annotations & Images from DICOM Files

9.1 Extract Annotations from DICOM

The following script will parse the DICOM file to extract annotations and returns the annotations in a python dictionary object.

```
$ python  dicom_wsi/extract_annotations.py -D <Path to DICOM file>
```

or from inside a python script:

```
from dicom_wsi.extract_annotations import extract_ann_dicom
d = extract_ann_dicom('tests/output.6-0.dcm')    # Change this to your dicom file
```

9.2 Extract Images from Dicom

The following script will parse the DICOM file to extract images and write them to a specified directory.

```
$ python  ./dicom_wsi/extract_image_patches.py -D <Path to Dicom file> -d <Path to␣
↪output directory>
```

CHAPTER 10

Code

CHAPTER 11

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:

11.1 Types of Contributions

11.1.1 Report Bugs

Report bugs at https://github.com/Steven-N-Hart/dicom_wsi/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.

11.1.2 Fix Bugs

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

11.1.3 Implement Features

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

11.1.4 Write Documentation

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

11.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Steven-N-Hart/dicom_wsi/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 :)

11.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/dicom_wsi.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 dicom_wsi
$ cd dicom_wsi/
$ 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 dicom_wsi 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.

11.3 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 2.7, 3.5, 3.6 and 3.7, and for PyPy. Check https://travis-ci.org/Steven-N-Hart/dicom_wsi/pull_requests and make sure that the tests pass for all supported Python versions.

11.4 Tips

To run a subset of tests:

```
$ pytest tests.test_dicom_wsi
```

11.5 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.

CHAPTER 12

Credits

12.1 Development Lead

- Steven N. Hart, Ph.D. https://github.com/Steven-N-Hart

12.2 Contributors

- Jin Jiang, Ph.D. https://github.com/smujiang

CHAPTER 13

History

13.1 0.1.0 (2021-1-22)

- First release on PyPI.

CHAPTER 14

Indices and tables

- genindex
- modindex
- search