brainlit

Release 0.0.0

unknown

May 03, 2023

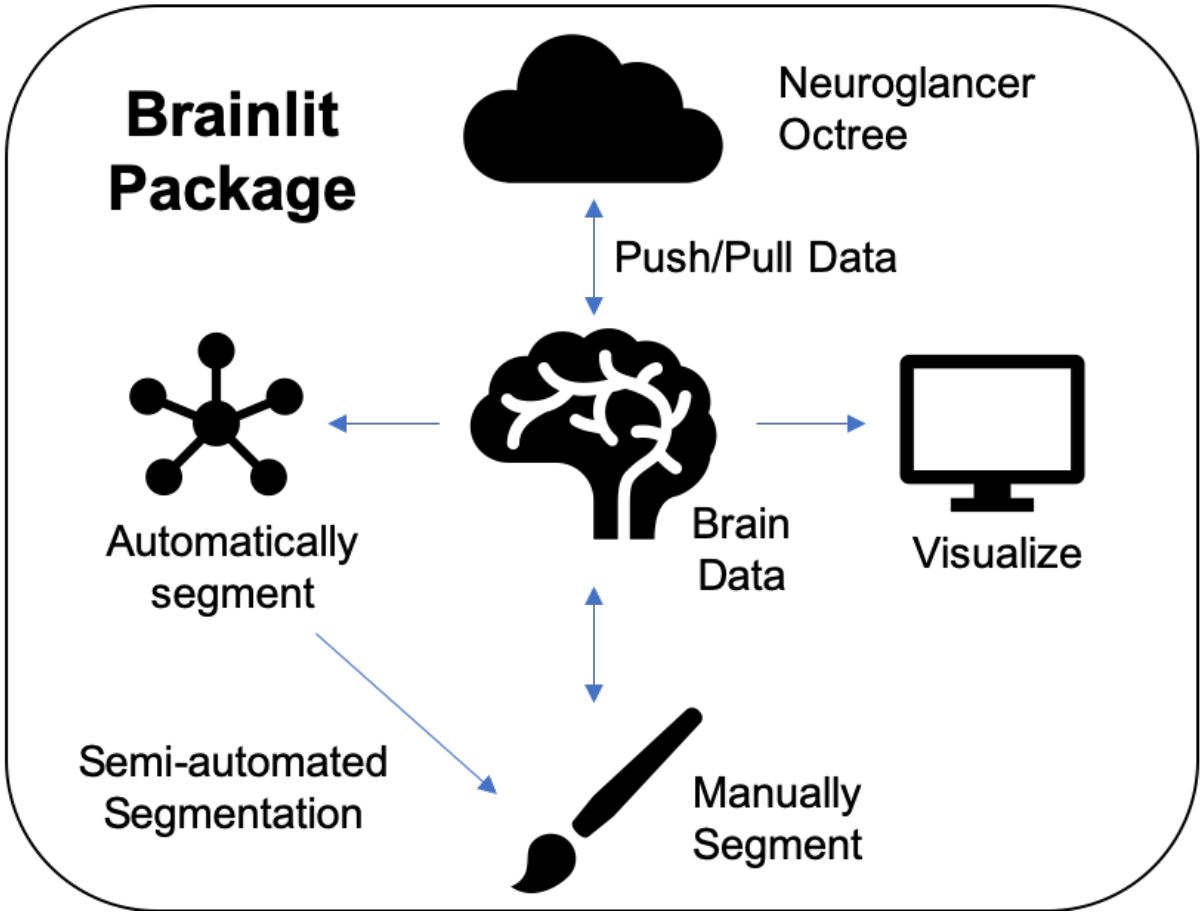
CONTENTS

CHAPTER

ONE

BRAINLIT

This repository is a container of methods that Neurodata usees to expose their open-source code while it is in the process of being merged with larger scientific libraries such as scipy, scikit-image, or scikit-learn. Additionally, methods for computational neuroscience on brains too specific for a general scientific library can be found here, such as image registration software tuned specifically for large brain volumes.



1.1 Motivation

The repository originated as the project of a team in Joshua Vogelstein's class **Neurodata** at Johns Hopkins University. This project was focused on data science towards the mouselight data. It became apparent that the tools developed for the class would be useful for other groups doing data science on large data volumes. The repository can now be considered a "holding bay" for code developed by Neurodata for collaborators and researchers to use.

1.2 Installation

1.2.1 Environment

(optional, any python >= 3.8 environment will suffice)

- get conda
- create a virtual environment: `conda create --name brainlit python=3.8`
- activate the environment: `conda activate brainlit`

1.2.2 Install from pypi

- install brainlit: `pip install brainlit`

1.2.3 Install from source

- clone the repo: `git clone https://github.com/neurodata/brainlit.git`
- cd into the repo: `cd brainlit`
- install brainlit: `pip install -e .`

1.2.4 Common Installation Issues

Mac OS

Obstacles Encountered During downloading_brains Tutorial (macOS)

(contact jduva4@jhu.edu or akodba1@jhu.edu for related questions)

1. **Issues with using a jupyter notebook**
    - fixes
2. If using `virtualenv` to create the environment rather than `conda`, make sure that you have Python 3 installed outside of Anaconda (call `python --version`) because many systems will not. Make sure that `pip` references Python 3 (the `pip --version` command should show `3.xx` in the path), otherwise `pip` installs could be updating Python 2 exclusively.
4. May run into a schema-related error when importing napari in Step 1: “This is specifically a suppressible warning because if you’re using a schema other than the ones in SUPPORTED_VERSIONS, it’s possible that not all functionality will be supported properly. If you don’t want to see these messages, add a warningfilter to your code.” (Source: https://github.com/cwacek/python-jsonschema-objects/issues/184)

5. Not exclusive to macOS but make sure aws .json file has no dollar signs in the strings and is being edited/saved within the terminal using a program like Nano or Vim. Do not use external editors like Sublime.
6. AWS Credendials Issues * See below
7. **Section (2) of downloading_brains notebook, Create a Neuroglancer instance and download the volume: make sure variab**

    - **For Example:**
        - Wrong: *img, bbox, vox = ngl_sess.pull_voxel(2, v_id, radius, radius, radius)*
        - Right: *img, bbox, vox = ngl_sess.pull_voxel(2, v_id, radius)*

8. **Section (4) of downloading_brains notebook, View the volume: the iPyNb kernel may consistently die when running, not a**

    - In terminal, type *pip install opencv-contrib-python-headless*
    - Or try including `%gui qt` just above the `import napari` line.

9. When installing `brainlit` on Mac OS BigSur, make sure you are using `python==3.9.0` and not `python==3.9.1`. This is a known *issue <https://github.com/napari/napari/issues/1393#issuecomment-745615931>*. Please report any other Mac OS BigSur compatibility issues.

Windows

Importing Brotli and Curses

When installing from source `pip install -e .` on Windows 10 with Python 3.9.0, sometimes brainlit would appear to install but upon importing, there was an issue with `import _brotli`.

It was fixed according to this thread where the Visual C++ Redistributable was updated by downloading the x64 version from here.

Also, an import error would occur for `_curses`. This was fixed by `pip install windows-curses` like from this thread.

Napari Display Problem

This document reports an issue that is encountered when running the tutorial `downloading_brains.ipynb`.

The document includes two sections: 1. a brief description of Issue#127 2. a detailed code history

1. Brief Description of Issue#127:

If your drivers/operating system are out of the date: - Windows 7 - python3.7.9

You may get the following error message:

> RuntimeError: Using glBindFramebuffer with no OpenGL context.

In the napari window, the images can be seen loaded but cannot be displayed at the screen as shown in the screenshot below: Napari screenshot

2. Detailed Code History

Input 1:

```
from brainlit.utils.session import NeuroglancerSession
from brainlit.utils.swc import graph_to_paths
import napari

dir = "s3://mouse-light-viz/precomputed_volumes/brain1"
dir_segments = "s3://mouse-light-viz/precomputed_volumes/brain1_segments"
mip = 0
v_id = 0
radius = 75

# get image and center point
ngl_sess = NeuroglancerSession(mip = mip, url = dir, url_segments=dir_segments)
img, bbox, vox = ngl_sess.pull_voxel(2, v_id, radius)
print(f"\n\nDownloaded volume is of shape {img.shape}, with total intensity
↪{sum(sum(sum(img)))}.")
```

Output 1:

```
Downloading: 100%|| 1/1 [00:00<00:00, 13.70it/s]
Downloading: 46it [00:38,  1.19it/s]

Downloaded volume is of shape (151, 151, 151), with total intensity 4946609.
```

Input 2:

```
G_sub = ngl_sess.get_segments(2, bbox)
paths = graph_to_paths(G_sub)
print(f"Selected volume contains {G_sub.number_of_nodes()} nodes and {len(paths)}␣
↪paths")
```

Output 2:

```
Downloading: 100%|| 1/1 [00:00<00:00,  3.47it/s]
Selected volume contains 6 nodes and 2 paths
```

Input 3:

```
with napari.gui_qt():
    viewer = napari.Viewer(ndisplay=3)
    viewer.add_image(img)
    viewer.add_shapes(data=paths, shape_type='path', edge_width=0.1, edge_color='blue
↪', opacity=0.1)
    viewer.add_points(vox, size=1, opacity=0.5)
```

Output 3:

```
ERROR:root:Unhandled exception:
Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\app\backends\_
↪qt.py", line 825, in paintGL
    self._vispy_canvas.events.draw(region=None)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 455, in __call__
```

(continues on next page)

(continued from previous page)

```
    self._invoke_callback(cb, event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 475, in _invoke_callback
    self, cb_event=(cb, event))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 471, in _invoke_callback
    cb(event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\context.py
↪", line 175, in flush_commands
    self.shared.parser.parse([('CURRENT', 0, fbo)])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 819, in parse
    self._parse(command)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 743, in _parse
    self._gl_initialize()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 851, in _gl_initialize
    if this_version < '2.1':
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\distutils\version.py", line 52, in _
↪_lt__
    c = self._cmp(other)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\distutils\version.py", line 335, in␣
↪_cmp
    if self.version == other.version:
AttributeError: 'LooseVersion' object has no attribute 'version'

ERROR:root:Unhandled exception:
Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 53, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native
AttributeError: 'function' object has no attribute '_native'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 72, in _get_gl_func
    func = getattr(_lib, name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 377, in __
↪getattr__
    func = self.__getitem__(name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 382, in __
↪getitem__
    func = self._FuncPtr((name_or_ordinal, self))
AttributeError: function 'glBindFramebuffer' not found

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\app\backends\_
↪qt.py", line 825, in paintGL
    self._vispy_canvas.events.draw(region=None)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 455, in __call__
    self._invoke_callback(cb, event)
```

(continues on next page)

(continued from previous page)

```
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 475, in _invoke_callback
    self, cb_event=(cb, event))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 471, in _invoke_callback
    cb(event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\context.py
↪", line 175, in flush_commands
    self.shared.parser.parse([('CURRENT', 0, fbo)])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 819, in parse
    self._parse(command)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 745, in _parse
    gl.glBindFramebuffer(gl.GL_FRAMEBUFFER, args[0])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 55, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native = _get_gl_func("glBindFramebuffer", None,␣
↪(ctypes.c_uint, ctypes.c_uint,))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 87, in _get_gl_func
    raise RuntimeError('Using %s with no OpenGL context.' % name)
RuntimeError: Using glBindFramebuffer with no OpenGL context.

WARNING: Error drawing visual <Volume at 0x21be1648>
WARNING:vispy:Error drawing visual <Volume at 0x21be1648>
ERROR:root:Unhandled exception:
Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 53, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native
AttributeError: 'function' object has no attribute '_native'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 72, in _get_gl_func
    func = getattr(_lib, name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 377, in __
↪getattr__
    func = self.__getitem__(name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 382, in __
↪getitem__
    func = self._FuncPtr((name_or_ordinal, self))
AttributeError: function 'glBindFramebuffer' not found

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\app\backends\_
↪qt.py", line 825, in paintGL
    self._vispy_canvas.events.draw(region=None)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 455, in __call__
    self._invoke_callback(cb, event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 475, in _invoke_callback
```

(continues on next page)

(continued from previous page)

```
    self, cb_event=(cb, event))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 471, in _invoke_callback
    cb(event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 217, in on_draw
    self._draw_scene()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 266, in _draw_scene
    self.draw_visual(self.scene)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 304, in draw_visual
    node.draw()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\visuals.py
↪", line 99, in draw
    self._visual_superclass.draw(self)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\visuals\visual.
↪py", line 443, in draw
    self._vshare.index_buffer)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\visuals\shaders\
↪program.py", line 101, in draw
    Program.draw(self, *args, **kwargs)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\program.py
↪", line 533, in draw
    canvas.context.flush_commands()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\context.py
↪", line 175, in flush_commands
    self.shared.parser.parse([('CURRENT', 0, fbo)])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 819, in parse
    self._parse(command)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 745, in _parse
    gl.glBindFramebuffer(gl.GL_FRAMEBUFFER, args[0])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 55, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native = _get_gl_func("glBindFramebuffer", None,␣
↪(ctypes.c_uint, ctypes.c_uint,))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 87, in _get_gl_func
    raise RuntimeError('Using %s with no OpenGL context.' % name)
RuntimeError: Using glBindFramebuffer with no OpenGL context.

WARNING: Error drawing visual <Volume at 0x21be1648>
WARNING:vispy:Error drawing visual <Volume at 0x21be1648>
ERROR:root:Unhandled exception:
Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 53, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native
AttributeError: 'function' object has no attribute '_native'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 72, in _get_gl_func
```

(continues on next page)

(continued from previous page)

```
    func = getattr(_lib, name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 377, in __
↪getattr__
    func = self.__getitem__(name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 382, in __
↪getitem__
    func = self._FuncPtr((name_or_ordinal, self))
AttributeError: function 'glBindFramebuffer' not found

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\app\backends\_
↪qt.py", line 825, in paintGL
    self._vispy_canvas.events.draw(region=None)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 455, in __call__
    self._invoke_callback(cb, event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 475, in _invoke_callback
    self, cb_event=(cb, event))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 471, in _invoke_callback
    cb(event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 217, in on_draw
    self._draw_scene()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 266, in _draw_scene
    self.draw_visual(self.scene)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 304, in draw_visual
    node.draw()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\visuals.py
↪", line 99, in draw
    self._visual_superclass.draw(self)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\visuals\visual.
↪py", line 443, in draw
    self._vshare.index_buffer)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\visuals\shaders\
↪program.py", line 101, in draw
    Program.draw(self, *args, **kwargs)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\program.py
↪", line 533, in draw
    canvas.context.flush_commands()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\context.py
↪", line 175, in flush_commands
    self.shared.parser.parse([('CURRENT', 0, fbo)])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 819, in parse
    self._parse(command)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 745, in _parse
    gl.glBindFramebuffer(gl.GL_FRAMEBUFFER, args[0])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 55, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native = _get_gl_func("glBindFramebuffer", None,␣
↪(ctypes.c_uint, ctypes.c_uint,))
```

(continues on next page)

(continued from previous page)

```
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 87, in _get_gl_func
    raise RuntimeError('Using %s with no OpenGL context.' % name)
RuntimeError: Using glBindFramebuffer with no OpenGL context.

WARNING: Error drawing visual <Volume at 0x21be1648>
WARNING:vispy:Error drawing visual <Volume at 0x21be1648>
ERROR:root:Unhandled exception:
Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 53, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native
AttributeError: 'function' object has no attribute '_native'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 72, in _get_gl_func
    func = getattr(_lib, name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 377, in __
↪getattr__
    func = self.__getitem__(name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 382, in __
↪getitem__
    func = self._FuncPtr((name_or_ordinal, self))
AttributeError: function 'glBindFramebuffer' not found

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\app\backends\_
↪qt.py", line 825, in paintGL
    self._vispy_canvas.events.draw(region=None)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 455, in __call__
    self._invoke_callback(cb, event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 475, in _invoke_callback
    self, cb_event=(cb, event))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 471, in _invoke_callback
    cb(event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 217, in on_draw
    self._draw_scene()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 266, in _draw_scene
    self.draw_visual(self.scene)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 304, in draw_visual
    node.draw()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\visuals.py
↪", line 99, in draw
    self._visual_superclass.draw(self)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\visuals\visual.
↪py", line 443, in draw
```

(continues on next page)

(continued from previous page)

```
    self._vshare.index_buffer)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\visuals\shaders\
↪program.py", line 101, in draw
    Program.draw(self, *args, **kwargs)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\program.py
↪", line 533, in draw
    canvas.context.flush_commands()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\context.py
↪", line 175, in flush_commands
    self.shared.parser.parse([('CURRENT', 0, fbo)])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 819, in parse
    self._parse(command)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 745, in _parse
    gl.glBindFramebuffer(gl.GL_FRAMEBUFFER, args[0])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 55, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native = _get_gl_func("glBindFramebuffer", None,␣
↪(ctypes.c_uint, ctypes.c_uint,))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 87, in _get_gl_func
    raise RuntimeError('Using %s with no OpenGL context.' % name)
RuntimeError: Using glBindFramebuffer with no OpenGL context.

WARNING: Error drawing visual <Volume at 0x21be1648>
WARNING:vispy:Error drawing visual <Volume at 0x21be1648>
ERROR:root:Unhandled exception:
Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 53, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native
AttributeError: 'function' object has no attribute '_native'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 72, in _get_gl_func
    func = getattr(_lib, name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 377, in __
↪getattr__
    func = self.__getitem__(name)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\ctypes\__init__.py", line 382, in __
↪getitem__
    func = self._FuncPtr((name_or_ordinal, self))
AttributeError: function 'glBindFramebuffer' not found

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\app\backends\_
↪qt.py", line 825, in paintGL
    self._vispy_canvas.events.draw(region=None)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 455, in __call__
    self._invoke_callback(cb, event)
```

(continues on next page)

(continued from previous page)

```
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 475, in _invoke_callback
    self, cb_event=(cb, event))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\util\event.py",␣
↪line 471, in _invoke_callback
    cb(event)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 217, in on_draw
    self._draw_scene()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 266, in _draw_scene
    self.draw_visual(self.scene)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\canvas.py
↪", line 304, in draw_visual
    node.draw()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\scene\visuals.py
↪", line 99, in draw
    self._visual_superclass.draw(self)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\visuals\visual.
↪py", line 443, in draw
    self._vshare.index_buffer)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\visuals\shaders\
↪program.py", line 101, in draw
    Program.draw(self, *args, **kwargs)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\program.py
↪", line 533, in draw
    canvas.context.flush_commands()
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\context.py
↪", line 175, in flush_commands
    self.shared.parser.parse([('CURRENT', 0, fbo)])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 819, in parse
    self._parse(command)
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\glir.py",␣
↪line 745, in _parse
    gl.glBindFramebuffer(gl.GL_FRAMEBUFFER, args[0])
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\_gl2.py
↪", line 55, in glBindFramebuffer
    nativefunc = glBindFramebuffer._native = _get_gl_func("glBindFramebuffer", None,␣
↪(ctypes.c_uint, ctypes.c_uint,))
File "C:\ProgramData\Miniconda3\envs\brainlit\lib\site-packages\vispy\gloo\gl\gl2.py",
↪ line 87, in _get_gl_func
    raise RuntimeError('Using %s with no OpenGL context.' % name)
RuntimeError: Using glBindFramebuffer with no OpenGL context.
```

WSL 2

WSL2 Installation Instructions

For Windows 10 users that prefer Linux functionality without the speed sacrifice of a Virtual Machine, Brainlit can be installed and run on WSL2. WSL2 is a fully functional Linux kernel that can run ELF64 binaries on a Windows Host. - OS Specifications: Version 1903, Build 18362 or higher - Installation Instructions - Any Linux distribution can be installed. Ubuntu16.04.3 was used for this tutorial.

Install python required libraries and build tools.

Run the below commands to configure the WSL2 environment. See here for more information.

```
$ sudo apt update && sudo apt install -y build-essential git libexpat1-dev libssl-dev␣
↪zlib1g-dev
$ libncurses5-dev libbz2-dev liblzma-dev
$ libsqlite3-dev libffi-dev tcl-dev linux-headers-generic libgdbm-dev
$ libreadline-dev tk tk-dev
```

Install a python version management tool, and create/activate a virtual environment

- Pyenv WSL2 Install (easiest for WSL2)
- Anaconda WSL2 Install

Install brainlit

- See installation section of README.md

Create and save AWS Secrets file

- See AWS Secrets file section below

Configure jupyter notebook

Install jupyter notebook: `$ python -m pip install jupyter notebook` and add the following line to your `~/.bashrc` script:

```
export DISPLAY=`grep -oP "(?<=nameserver ).+" /etc/resolv.conf`:0.0
```

To launch jupyter notebook, you need to type `$ jupyter notebook --allow-root`, not just `$ jupyter notebook` Then copy and paste one of the URLs outputted into your web browser. If your browser is unable to connect, try unblocking the default jupyter port via this command: ``$ sudo ufw allow 8888 ``

Configure X11 Port Forwarding

- Install VcXsrv Windows X Server on your Windows host machine
- Let VcXsrv through your Public & Private windows firewall. (Control Panel -> System and Security -> Windows Defender Firewall -> Allowed Apps -> Change Settings)
- Run XLaunch on your Windows Host Machine with default settings AND select the "Disable Access Control" option
- To confim X11 Port Forwarding is configured, run `xclock` on the subsystem. This should launch on your windows machine.

Exceptions

- The Napari viewer cannot be fully launched (only launches a black screen), because OpenGL versions>1.5 are not currently supported by WSL2. This should be resolved in upcoming WSL2 updates.

AWS Credentials Issues

warning SECURITY DISCLAIMER :warning:

Do **NOT** push any official AWS credentials to any repository. These posts are a good reference to get a sense of what pushing AWS credentials implies:

1. *I Published My AWS Secret Key to GitHub* by Danny Guo here
2. *Exposing your AWS access keys on Github can be extremely costly. A personal experience.* by Guru here
3. *Dev put AWS keys on Github. Then BAD THINGS happened* by Darren Pauli here

Brainlit can access data volumes stored in AWS S3 through the CloudVolume package. As specified in the docs, AWS credentials have to be stored in a file called `aws-secret.json` inside the `~.cloudvolume/secrets/` folder.

Prerequisites to successfully troubleshoot errors related to AWS credentials:

- The data volume is hosted on S3 (i.e. the link looks like `s3://your-bucket-name/some-path/some-folder`).
- Familiarity with IAM Roles and how to create them.
- An `AWS_ACCESS_KEY_ID` and an `AWS_SECRET_ACCESS_KEY` with adequate permissions, provided by an AWS account administrator. Brainlit does not require the IAM user associated with the credentials to have access to the AWS console (i.e. it can be a service account).

Here is a collection of known issues, along with their troubleshoot guide:

Missing `AWS_ACCESS_KEY_ID`

Error message:

```
python
~/opt/miniconda3/envs/brainlit/lib/python3.8/site-packages/cloudvolume/
↪connectionpools.py in _create_connection(self)
    99       return boto3.client(
    100          's3',
--> 101          aws_access_key_id=self.credentials['AWS_ACCESS_KEY_ID'],
    102          aws_secret_access_key=self.credentials['AWS_SECRET_ACCESS_KEY'],
    103          region_name='us-east-1',

KeyError: 'AWS_ACCESS_KEY_ID'
```

This error is thrown when the *credentials* object has an empty `AWS_ACCESS_KEY_ID` entry. This probably indicates that ``aws-secret.json` is not stored in the right folder and it cannot be found by CloudVolume. Make sure your credential file is named correctly and stored in `~.cloudvolume/secrets/`. If you are a Windows user, the output of this Python snippet is the expansion of ~ for your system:

```
python
import os
HOME = os.path.expanduser('~')
print(HOME)
```

example output:

```
bash
Python 3.8.3 (v3.8.3:6f8c8320e9)
>>> import os
>>> HOME = os.path.expanduser('~')
>>> print(HOME)
C:\Users\user
```

Empty `AKID` (Access Key ID)

Error message:

```
python
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
↪botocore/client.py in _make_api_call(self, operation_name, api_params)
    654             error_code = parsed_response.get("Error", {}).get("Code")
    655             error_class = self.exceptions.from_code(error_code)
--> 656             raise error_class(parsed_response, operation_name)
    657         else:
    658             return parsed_response
ClientError: An error occurred (AuthorizationHeaderMalformed) when calling the␣
↪GetObject operation: The authorization header is malformed; a non-empty Access Key␣
↪(AKID) must be provided in the credential.
```

This error is thrown when your `aws-secret.json` file is stored and loaded correctly, and it looks like this:

```
json
{
"AWS_ACCESS_KEY_ID": "",
"AWS_SECRET_ACCESS_KEY": ""
}
```

Even though the bucket itself may be public, boto3 requires some non-empty AWS credentials to instantiante the S3 API client.

Access denied

```
python
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
↪botocore/client.py in _make_api_call(self, operation_name, api_params)
    654             error_code = parsed_response.get("Error", {}).get("Code")
    655             error_class = self.exceptions.from_code(error_code)
--> 656             raise error_class(parsed_response, operation_name)
    657         else:
    658             return parsed_response
ClientError: An error occurred (AccessDenied) when calling the GetObject operation:␣
↪Access Denied
```

This error is thrown when:

1. The AWS credentials are stored and loaded correctly but are not allowed to access the data volume. A check with an AWS account administrator is required.
2. There is a typo in your credentials. The content of `aws-secret.json` should look like this:

```
json
{
"AWS_ACCESS_KEY_ID": "$YOUR_AWS_ACCESS_KEY_ID",
"AWS_SECRET_ACCESS_KEY": "$AWS_SECRET_ACCESS_KEY"
}
```

where the `$` are placeholder characters and should be replaced along with the rest of the string with the official AWS credentials.

1.3 How to use Brainlit

1.3.1 Data setup

The `source` data directory should have an octree data structure

```
 data/
├── default.0.tif
├── transform.txt
├── 1/
│   ├── 1/, ..., 8/
│   └── default.0.tif
├── 2/ ... 8/
└── consensus-swcs (optional)
    ├── G-001.swc
    ├── G-002.swc
    └── default.0.tif
```

If your team wants to interact with cloud data, each member will need account credentials specified in `~/.cloudvolume/secrets/x-secret.json`, where `x` is one of `[aws, gc, azure]` which contains your id and secret key for your cloud platform. We provide a template for `aws` in the repo for convenience.

1.3.2 Create a session

Each user will start their scripts with approximately the same lines:

```
from brainlit.utils.ngl import NeuroglancerSession

session = NeuroglancerSession(url='file:///abc123xyz')
```

From here, any number of tools can be run such as the visualization or annotation tools. Viz demo.

1.4 Features

1.4.1 Registration

The registration subpackage is a facsimile of ARDENT, a pip-installable (pip install ardent) package for nonlinear image registration wrapped in an object-oriented framework for ease of use. This is an implementation of the LDDMM algorithm with modifications, written by Devin Crowley and based on "Diffeomorphic registration with intensity transformation and missing data: Application to 3D digital pathology of Alzheimer's disease." This paper extends on an older LDDMM paper, "Computing large deformation metric mappings via geodesic flows of diffeomorphisms."

This is the more recent paper:

Tward, Daniel, et al. "Diffeomorphic registration with intensity transformation and missing data: Application to 3D digital pathology of Alzheimer's disease." Frontiers in neuroscience 14 (2020).

https://doi.org/10.3389/fnins.2020.00052

This is the original LDDMM paper:

Beg, M. Faisal, et al. "Computing large deformation metric mappings via geodesic flows of diffeomorphisms." International journal of computer vision 61.2 (2005): 139-157.

https://doi.org/10.1023/B:VISI.0000043755.93987.aa

A tutorial is available in docs/notebooks/registration_demo.ipynb.

1.5 Core

The core brain-lit package can be described by the diagram at the top of the readme:

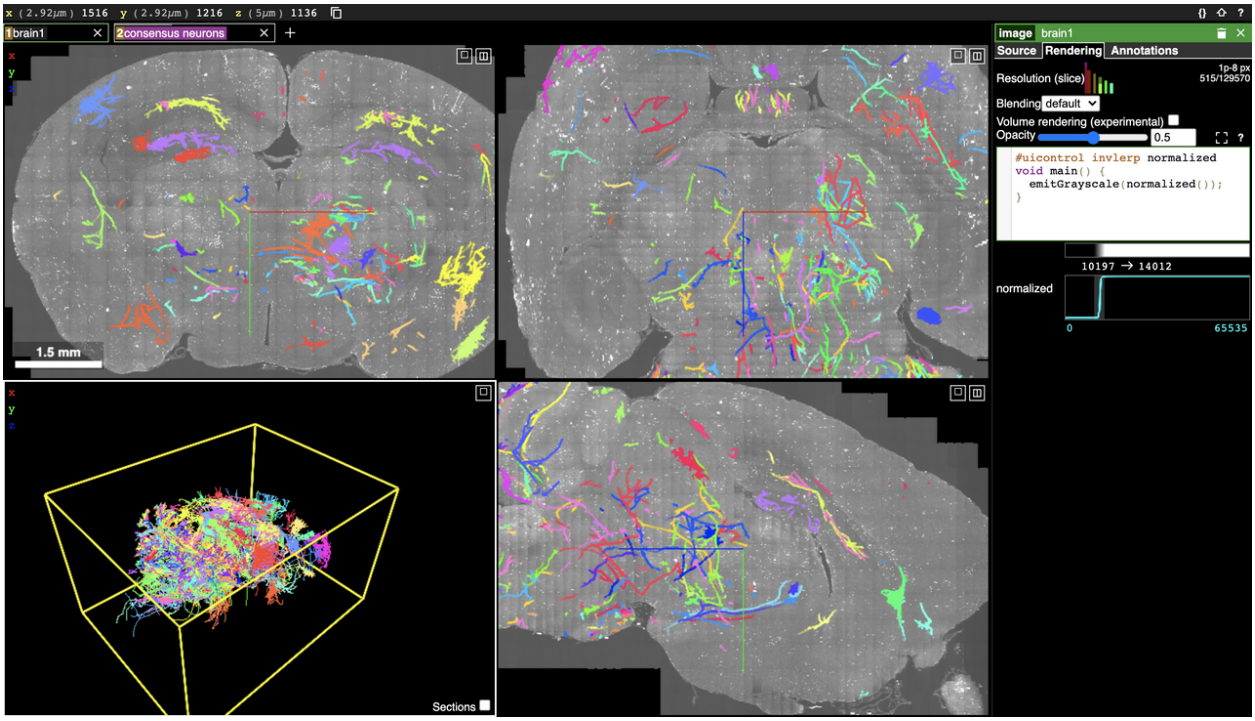
1.5.1 Push and Pull Data

Brainlit uses Seung Lab's CloudVolume package to push and pull data through the cloud or a local machine in an efficient and parallelized fashion. Uploading demo showcases how to upload both brain volumes and neuron traces. Likewise, downloading demo shows how to download data.

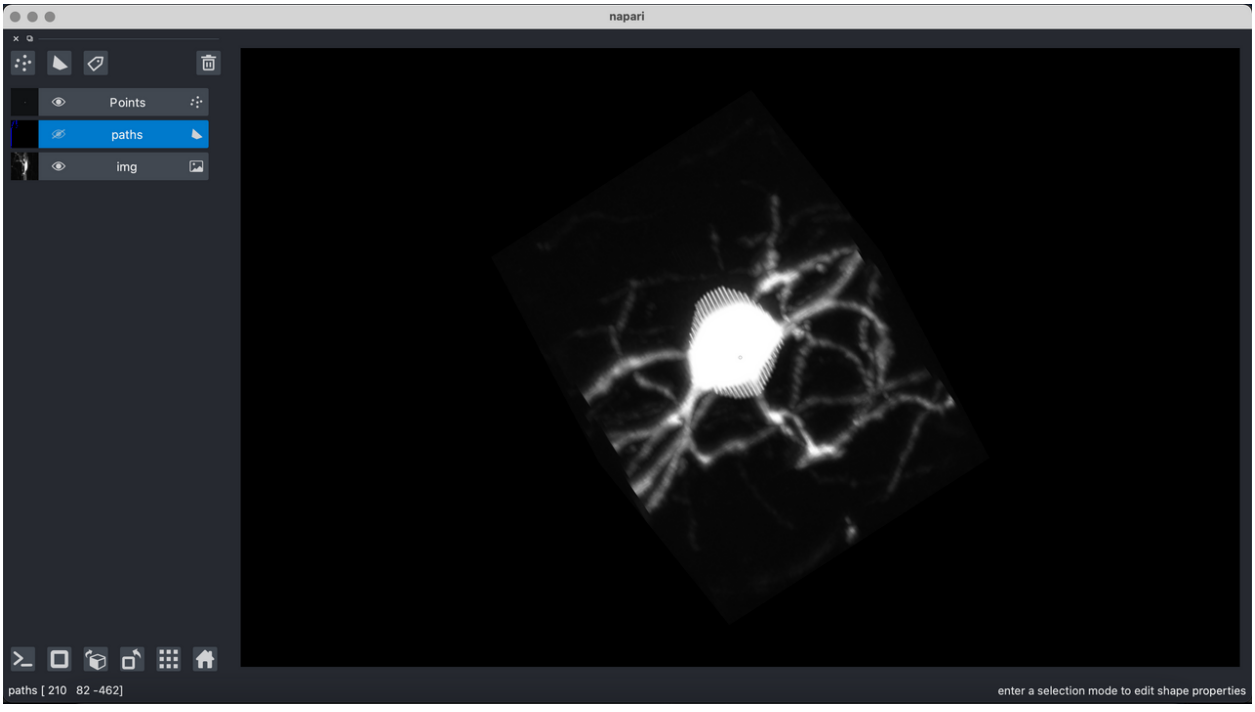
We note the CloudVolume's only requirement is to have an account on S3, as the brain data is publicly available.

1.5.2 Visualize

Brainlit supports many methods to visualize large data. Visualizing the entire data can be done via Google's Neuroglancer directly in your browser. For example, this link will visualize a slice of one of the brains contained in the MouseLight dataset provided by HHMI Janelia, as shown in the screenshot below



Brainlit also has tools to visualize chunks of data as 2d slices or as a 3d model. The visualization demo will open the following napari view of a volume of brain



1.5.3 Manually Segment

Brainlit includes a lightweight manual segmentation pipeline. This allows collaborators of a projec to pull data from the cloud, create annotations, and push their annotations back up as a separate channel. Auto demo.

1.5.4 Automatically and Semi-automatically Segment

Similar to the above pipeline, segmentations can be automatically or semi-automatically generated and pushed to a separate channel for viewing. Semi-auto demo.

1.6 Tests

Running tests can easily be done by moving to the root directory of the brainlit package ant typing `pytest tests` or `python -m pytest tests`. Running a specific test, such as `test_upload.py` can be done simply by `ptest tests/test_upload.py`.

1.7 Contributing

We welcome all contributors, and encourage them to follow our contribution guidelines found in CONTRIBUTING.md. Issues with the "good first issue" tag are meant for contributors that are either new to open source coding, or new to the package. Additionally, users are encouraged to use issues not only to discuss code-related problems, but for more general discussions about the package.

1.8 Credits

Brainlit is a product of the neurodata lab. It is actively maintained by Thomas Athey (@tathey1) and Bijan Varjavand (@bvarjavand), and is regularly used and contributed to by students in the Neuro Data Design course. We strive to follow the same code of conduct that applies to the Microsoft open source community.

CHAPTER

TWO

DOCUMENTATION

2.1 Tutorial

2.1.1 Utils

Tutorials showcasing how to use the utils folder.

[1]:
```
# from brainlit.utils import upload, session
# from pathlib import Path
# import napari
# from napari.utils import nbscreenshot
# import os
# %gui qt
```

```
/Users/thomasathey/Documents/mimlab/mouselight/env/lib/python3.8/site-packages/python_
↪jsonschema_objects/__init__.py:50: UserWarning: Schema version http://json-schema.
↪org/draft-04/schema not recognized. Some keywords and features may not be supported.
  warnings.warn(
```

Uploading Brain Images from data in the Octree format.

This notebook demonstrates uploading the 2 lowest-resolution brain volumes and a `.swc` segment file. The upload destination could easily be set to a url of a cloud data server such as s3.

1) Define variables.

- `source` and `source_segments` are the root directories of the octree-formatted data and swc files.
- `p` is the prefix string. `file://` indicates a filepath, while `s3://` or `gc://` indicate URLs.
- `dest` and `dest_segments` are the destinations for the uploads (in this case, filepaths).
- `num_res` denotes the number of resolutions to upload.

The below paths lead to sample data in the NDD Repo. Alter the below path definitions to point to your own local file locations.

[2]:
```
# source = (Path().resolve().parents[2] / "data" / "data_octree").as_posix()
# dest = (Path().resolve().parents[2] / "data" / "upload").as_uri()
# dest_segments = (Path().resolve().parents[2] / "data" / "upload_segments").as_uri()
# dest_annotation = (Path().resolve().parents[2] / "data" / "upload_annotation").as_
↪uri()
# num_res = 2
```

2) Upload the image data (.tif)

If the upload fails with the error: `timed out on a chunk on layer index 0. moving on to the next step of pipeline`, re-run the `upload_volumes` function but with the `continue_upload` parameter, which takes `layer index` (the layer index said in the error message) and `image index` (the last succesful image that uploaded).

For example, if the output failed after image 19, then run `upload.upload_volumes(source, dest, num_res, continue_upload = (0, 19))`. Repeat this till all of the upload is complete.

[3]:
```
# upload.upload_volumes(source, dest, num_res)
```

```
Creating precomputed volume at layer index 0: 100%|| 1/1 [00:04<00:00,  4.27s/it]
Creating precomputed volume at layer index 1:   0%|          | 0/8 [00:00<?, ?it/s]
Finished layer index 0, took 4.266659259796143 seconds
Creating precomputed volume at layer index 1: 100%|| 8/8 [00:09<00:00,  1.19s/it]
Finished layer index 1, took 9.484457015991211 seconds
```

3) Upload the segmentation data (.swc)

If uploading a `.swc` file associated with a brain volume, then use upload.py. Otherwise if uploading `swc` files with different name formats, use upload_benchmarking.py

[4]:
```
# upload.upload_segments(source, dest_segments, num_res)
```

```
converting swcs to neuroglancer format...: 100%|| 1/1 [00:00<00:00, 43.32it/s]
Uploading: 100%|| 1/1 [00:00<00:00, 194.74it/s]
```

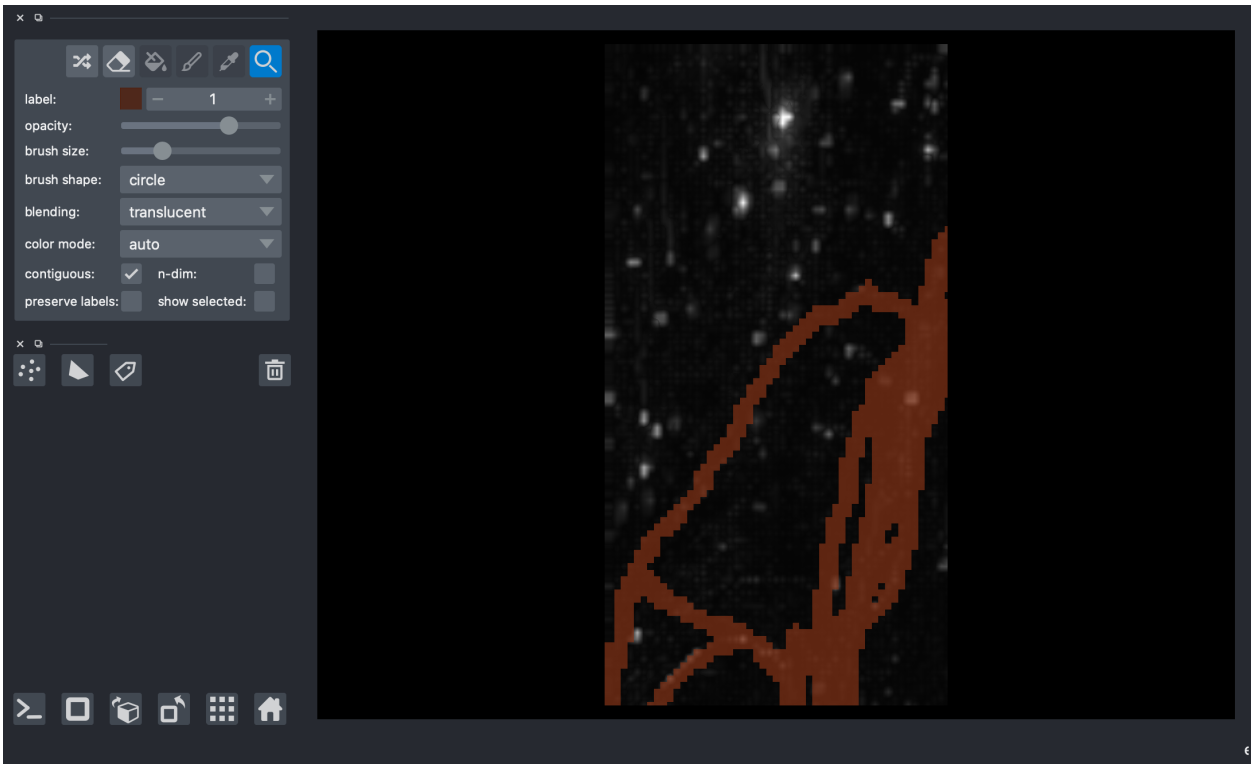
Download the data with NeuroglancerSession and generate labels.

[5]:
```
# %%capture
# sess = session.NeuroglancerSession(url=dest, url_segments=dest_segments, mip=0)  #␣
↪create session object object
# img, bounds, vertices = sess.pull_vertex_list(2, range(250, 350), expand=True)  #␣
↪get image containing some data
# labels = sess.create_tubes(2, bounds, radius=1)  # generate labels via tube␣
↪segmentation
```

4) Visualize the data with napari

[6]:
```
# viewer = napari.Viewer(ndisplay=3)
# viewer.add_image(img)
# viewer.add_labels(labels)
# nbscreenshot(viewer)
```

[6]:



[ ]:

[11]:
```
from brainlit.utils.session import NeuroglancerSession
import napari
from napari.utils import nbscreenshot

%gui qt
```

Downloading Brain data tutorial

We have prepared 2 brain volumes, as well as axon segment labels, at the below s3 urls (see uploading_brains.ipynb). The method demonstrated below pulls a region of the volume around an annotated axon point set by the user.

1) Define Variables

- mip ranges from higher resolution (0) to lower resolution (1).
- v_id are vertex ids ranging from the soma (0) to the end of the axon (1649).
- radius is the radius to pull around the selected point, in voxels.

[7]:
```
"""
dir = "s3://open-neurodata/brainlit/brain1"
dir_segments = "s3://open-neurodata/brainlit/brain1_segments"
dir_2 = "s3://open-neurodata/brainlit/brain2"
dir_2_segments = "s3://open-neurodata/brainlit/brain2_segments"
mip = 0
```

(continues on next page)

(continued from previous page)

```
v_id = 0
radius = 75
"""
```

2) Create a NeuroglancerSession instance and download the volume.

[8]:
```
"""
# get image and center point
ngl_sess = NeuroglancerSession(mip = mip, url = dir, url_segments=dir_segments)
img, bbox, vox = ngl_sess.pull_voxel(2, v_id, radius)
print(f"\n\nDownloaded volume is of shape {img.shape}, with total intensity
↪{sum(sum(sum(img)))}.")
"""
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  8.25it/s]
Downloading: 46it [00:01, 23.83it/s]

Downloaded volume is of shape (151, 151, 151), with total intensity 4946609.
```

3) Generate a graph from the segment data within the volume, and convert it to paths.

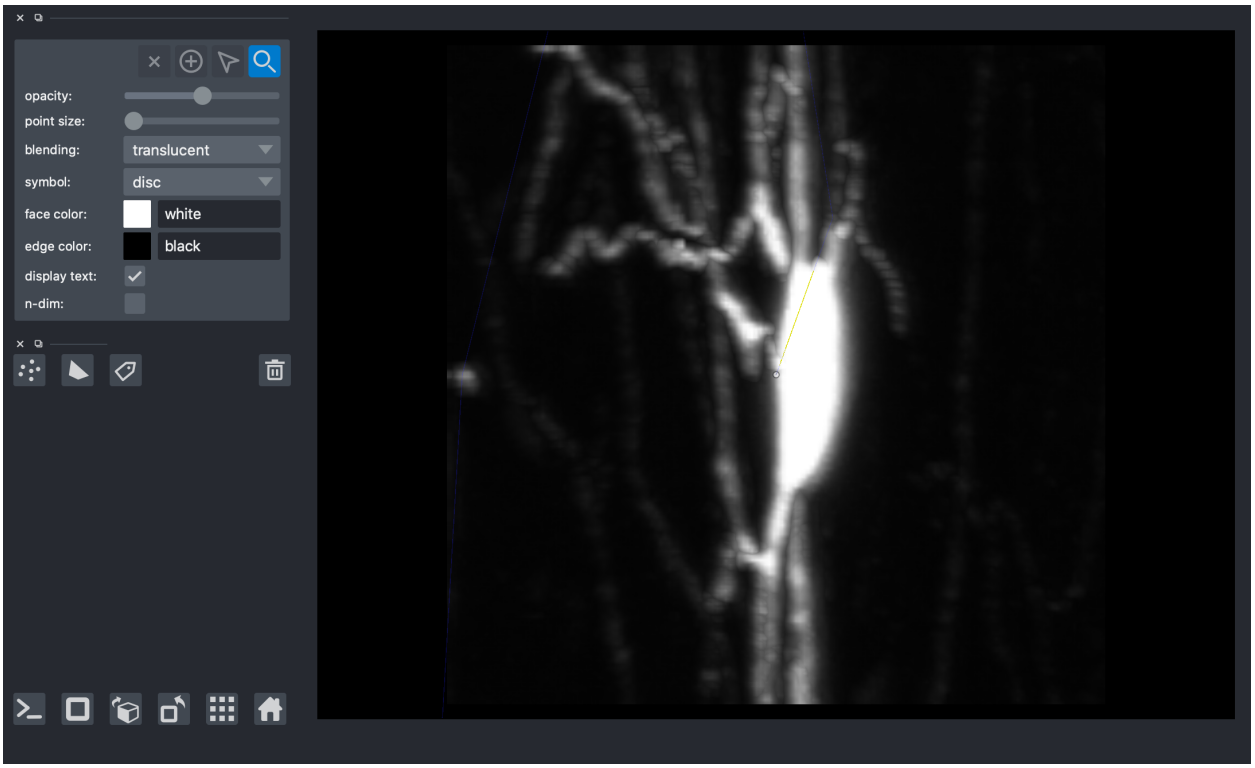
[9]:
```
"""
G_paths = ngl_sess.get_segments(2, bbox)
G_sub = G_paths[0]
paths = G_paths[1]

print(f"Selected volume contains {G_sub.number_of_nodes()} nodes and {len(paths)}␣
↪paths")
"""
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  8.73it/s]
Downloading: 100%|| 1/1 [00:00<00:00,  8.56it/s]
Selected volume contains 6 nodes and 2 paths
```

4) View the volume with paths overlaid via napari.¶

[12]:
```
"""
viewer = napari.Viewer(ndisplay=3)
viewer.add_image(img)
viewer.add_shapes(data=paths, shape_type='path', edge_width=0.1, edge_color='blue',␣
↪opacity=0.1)
viewer.add_points(vox, size=1, opacity=0.5)
nbscreenshot(viewer)
"""
```

[12]:



[ ]:

Download benchmarking data from S3 with Neuroglancer

This notebook explains how to:

(1) Read benchmarking data from S3 via Neuroglancer

(2) download raw benchmarking data to your local computer

Quick notes on the benchmarking data:

In octree format, data is labled in folders, labeled test_1 through test_25 and validation_1 through validation_25.

If when downloading, you get a reshape error, try first uploading segments and then re-uploading the volumes.

Known issues with a few of the files:

- test_9,test_10 - didnt seem to have good swc alignment
- test_24 - issues with the image
- validation_11 - seems to be a shift between swcs and the image

[1]:
```
import napari
from napari.utils import nbscreenshot
```

Define locations

[2]:
```
from brainlit.utils import session
from brainlit.utils.Neuron_trace import NeuronTrace

# #Can change to test_"1-25", validation_"1-25"
dest = "s3://open-neurodata/brainlit/benchmarking_data/validation_7"
dest_segments = "s3://open-neurodata/brainlit/benchmarking_data/validation_7"
```

```
c:\users\shrey\anaconda3\envs\ndd-windows\lib\site-packages\python_jsonschema_objects\
↪__init__.py:53: UserWarning: Schema version http://json-schema.org/draft-04/schema␣
↪not recognized. Some keywords and features may not be supported.
  self.schema["$schema"]
```

Create Neuroglancer session & download benchmarking volume

[3]:
```
%%capture
sess = session.NeuroglancerSession(
    url=dest, url_segments=dest_segments, mip=0
)  # create session object
img, bounds, vertices = sess.pull_vertex_list(
    1, [1], 0, expand=True
)  # get full benchmarking image
```

Download a specific .swc

[4]:
```
seg_id = 1  # Can change

G_paths = sess.get_segments(seg_id, bounds, rounding=False)
G = G_paths[0]
paths = G_paths[1]
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  6.65it/s]
Downloading: 100%|| 1/1 [00:00<00:00,  6.07it/s]
```

Visualize with napari

[5]:
```
# viewer = napari.Viewer(ndisplay=3)
# viewer.add_image(img)
# viewer.add_shapes(data=paths, shape_type='path', edge_width=1.0, edge_color='blue',␣
↪opacity=0.8)
```

[6]:
```
# nbscreenshot(viewer, canvas_only = True)
```

Download raw benchmarking data

This will download the benchmarking data in .tif and .swc format to a local destination

[7]:
```
import boto3
from botocore import UNSIGNED
from botocore.client import Config
import os
from pathlib import Path
import numpy as np
from skimage import io
from tqdm import tqdm
```

[8]:
```
cwd = Path(os.path.abspath(""))
data_dir = os.path.join(cwd, "data")
print(f"Downloading segments to {data_dir}")
if not os.path.exists(data_dir):
    os.makedirs(data_dir)

im_dir = os.path.join(data_dir, "sample-tif-location")
if not os.path.exists(im_dir):
    os.makedirs(im_dir)

swc_dir = os.path.join(data_dir, "sample-swc-location")
if not os.path.exists(swc_dir):
    os.makedirs(swc_dir)
```

```
Downloading segments to C:\Users\shrey\Documents\NDD\brainlit\docs\notebooks\utils\
↪data
```

On mac/linux, we use os.path.join to construct the s3 path. However on windows you should set prefix to "brainlit/benchmarking_data/tif-files"

[9]:
```
s3 = boto3.resource("s3", config=Config(signature_version=UNSIGNED))
bucket = s3.Bucket("open-neurodata")
prefix = "brainlit/benchmarking_data/tif-files"  # use this for windows
# prefix = os.path.join("brainlit", "benchmarking_data", "tif-files") #use this for␣
↪mac/linux
im_count = 0
for _ in bucket.objects.filter(Prefix=prefix):
    im_count += 1
for i, im_obj in enumerate(tqdm(bucket.objects.filter(Prefix=prefix))):
    if im_obj.key[-4:] == ".tif":
        im_name = os.path.basename(im_obj.key)
        im_path = os.path.join(im_dir, im_name)
        bucket.download_file(im_obj.key, im_path)
```

```
52it [00:57,  1.12s/it]
```

The below code can visualize a specified .tif file.

[10]:
```
file_name = "test_10-gfp.tif"  # Can change to any image (test 1-25, validation 1-25)

im_file = Path(im_dir) / file_name
im = io.imread(im_file, plugin="tifffile")

# viewer = napari.Viewer(ndisplay=3)
# viewer.add_image(im)
```

[11]:
```
# nbscreenshot(viewer, canvas_only = True)
```

Again, on windows you need to make the variable called prefix a string

[12]:
```
s3 = boto3.resource("s3", config=Config(signature_version=UNSIGNED))
bucket = s3.Bucket("open-neurodata")
prefix = "brainlit/benchmarking_data/Manual-GT"  # use this for windows
# prefix = os.path.join("brainlit", "benchmarking_data", "Manual-GT") #use this for␣
↪mac/linux
swc_count = 0
for _ in bucket.objects.filter(Prefix=prefix):
    swc_count += 1
for i, swc_obj in enumerate(tqdm(bucket.objects.filter(Prefix=prefix))):
    if swc_obj.key[-4:] == ".swc":
        idx = swc_obj.key.find("Manual-GT")
        swc_name = swc_obj.key[idx:]
        swc_path = os.path.join(swc_dir, swc_name)
        dir = os.path.dirname(swc_path)
        if not os.path.exists(dir):
            os.makedirs(dir)
        bucket.download_file(swc_obj.key, swc_path)
```

```
601it [01:30,  6.67it/s]
```

[13]:
```
from brainlit.utils.benchmarking_params import (
    brain_offsets,
    vol_offsets,
    scales,
    type_to_date,
)
from brainlit.utils.Neuron_trace import NeuronTrace
from pathlib import Path
import numpy as np
from skimage import io
```

[14]:
```
im_dir = Path(im_dir)
swc_base_path = Path(swc_dir) / "Manual-GT"

gfp_files = list(im_dir.glob("**/*-gfp.tif"))
```

[15]:
```
for im_num, im_path in enumerate(gfp_files):
    print(f"Image {im_num+1}/{len(gfp_files)}")
    print(im_path)

    f = im_path.parts[-1][:-8].split("_")
    image = f[0]
    date = type_to_date[image]
    num = int(f[1])

    scale = scales[date]
    brain_offset = brain_offsets[date]
    vol_offset = vol_offsets[date][num]
    im_offset = np.add(brain_offset, vol_offset)

    lower = int(np.floor((num - 1) / 5) * 5 + 1)
    upper = int(np.floor((num - 1) / 5) * 5 + 5)
    dir1 = date + "_" + image + "_" + str(lower) + "-" + str(upper)
```

(continues on next page)

(continued from previous page)

```
    dir2 = date + "_" + image + "_" + str(num)
    swc_path = swc_base_path / dir1 / dir2
    swc_files = list(swc_path.glob("**/*.swc"))
    im = io.imread(im_path, plugin="tifffile")
    print(f"Image shape: {im.shape}")

    paths_total = []
    for swc_num, swc in enumerate(swc_files):
        if "0" in swc.parts[-1]:
            # skip the bounding box swc
            continue

        swc_trace = NeuronTrace(path=str(swc))
        paths = swc_trace.get_paths()
        swc_offset, _, _, _ = swc_trace.get_df_arguments()
        offset_diff = np.subtract(swc_offset, im_offset)

        for path_num, p in enumerate(paths):
            pvox = (p + offset_diff) / (scale) * 1000
            paths_total.append(pvox)

    break
```

```
Image 1/50
C:\Users\shrey\Documents\NDD\brainlit\docs\notebooks\utils\data\sample-tif-location\
↪test_1-gfp.tif
Image shape: (100, 330, 330)
```

[16]:
```
# viewer = napari.Viewer(ndisplay=3)
# viewer.add_image(np.swapaxes(im,0,2))
# viewer.add_shapes(data=paths_total, shape_type='path', edge_width=1.0, edge_color=
↪'blue', opacity=0.8)
```

[17]:
```
# nbscreenshot(viewer, canvas_only = True)
```

Upload benchmarking data to S3 with Neuroglancer

[ ]:
```
from brainlit.utils import upload
from pathlib import Path
```

Uploading Benchmarking Images from local data locations .

This notebook demonstrates uploading the benchmarking data and associated `.swc` segment files. The upload destination could easily be set to a url of a cloud data server such as s3.

1) Define variables.

- `source` is the root directory of the data and swc files.
  - the .tif file is in the root directory and .swc files are in a subfolder called "consensus-swcs"
- `p` is the prefix string. `file://` indicates a filepath, while `s3://` or `gc://` indicate URLs.
- `dest` and `dest_segments` are the destinations for the uploads (in this case, filepaths).

The below paths lead to sample data in my local drive. Alter the below path definitions to point to your own local file locations.

Note:

The below upload destination points to the open-neurodata S3. Uploading data will overwrite the current benchmarking data on S3.

[ ]:
```
source = (Path().resolve().parents[5] / "Downloads" / "validation_21").as_posix()
dest = "s3://open-neurodata/brainlit/benchmarking_data/validation_21"
dest_segments = "s3://open-neurodata/brainlit/benchmarking_data/validation_21"
```

2) Upload the segmentation data (.swc)

[ ]:
```
upload.upload_segments(source, dest_segments, 1, benchmarking=True)
```

3) Upload the image data (.tif)

[ ]:
```
upload.upload_volumes(source, dest, 1, benchmarking=True)
```

Appendix

- If when downloading, you get a reshape error, try uploading the segments before uploading the volumes

[ ]:
```
test_list = []
validation_list = []
for i in range(25):
    test_list.append("test_" + str(i + 1))
    validation_list.append("validation_" + str(i + 1))
```

[ ]:
```
num_res = 1

for test in test_list:
    print(test)
```

(continues on next page)

(continued from previous page)

```
    source = (Path().resolve().parents[5] / "Downloads" / test).as_posix()
    dest = "s3://open-neurodata/brainlit/benchmarking_data/" + test
    dest_segments = "s3://open-neurodata/brainlit/benchmarking_data/" + test
    upload.upload_segments(source, dest_segments, num_res, benchmarking=True)
    upload.upload_volumes(source, dest, num_res, benchmarking=True)

for val in validation_list:
    print(val)
    source = (Path().resolve().parents[5] / "Downloads" / val).as_posix()
    dest = "s3://open-neurodata/brainlit/benchmarking_data/" + val
    dest_segments = "s3://open-neurodata/brainlit/benchmarking_data/" + val
    upload.upload_segments(source, dest_segments, num_res, benchmarking=True)
    upload.upload_volumes(source, dest, num_res, benchmarking=True)
```

2.1.2 Pipelines

BrainLine: Whole-Brain Fluorescence Volume Analysis Pipeline

Leverage ilastik to perform axon detection and soma detection on brain images, and combine with CloudReg image registration for visualization and analysis.

Axon Segmentation Analysis of Whole-Brain Fluorescence Images

[ ]:
```
from brainlit.BrainLine.util import (
    json_to_points,
    download_subvolumes,
)
from brainlit.BrainLine.parse_ara import *
import xml.etree.ElementTree as ET
from brainlit.BrainLine.imports import *
from brainlit.BrainLine.apply_ilastik import (
    ApplyIlastik,
    ApplyIlastik_LargeImage,
    plot_results,
    examine_threshold,
)
from brainlit.BrainLine.analyze_results import (
    AxonDistribution,
    collect_regional_segmentation,
)

%gui qt5
```

1. Before Using this notebook:

1a. Install brainlit, and dependencies

1b. Write images to s3 using CloudReg

```
- e.g. python -m cloudreg.scripts.create_precomputed_volumes --s3_input_paths <path-
↪to-stitched-images> --s3_output_paths  <s3-address-for-output>  --voxel_size <x-
↪resolution> <y-resolution> <z-resolution> --num_procs <num-cpus> --resample_iso
↪<boolean-to-resample-isotropically>
```

1c. Make point annotations in neuroglancer to identify subvolumes for validation (and possible training)

```
- instructions: https://neurodata.io/help/neuroglancer-pt-annotations/
    ,
{
"type":"pointAnnotation",
"name": "val",
"points": []
}
```

1d. Update axon_data.py file

```
[ ]: brainlit_path = Path(os.path.abspath(""))
brainlit_path = brainlit_path.parents[3]
print(f"Path to brainlit: {brainlit_path}")
data_file = brainlit_path / "brainlit" / "BrainLine" / "data" / "axon_data.json"

with open(data_file) as f:
    data = json.load(f)
brain2paths = data["brain2paths"]
for id in brain2paths.keys():
    if "base" in brain2paths[id].keys() and "val_info" in brain2paths[id].keys():
        base = brain2paths[id]["base"]
        if "http" in base:
            print(f"Sample {id}: http in basepath, which may cause write errors")

        try:
            url = brain2paths[id]["val_info"]["url"]
            layer = brain2paths[id]["val_info"]["layer"]
            pts = json_to_points(url)[layer]
        except:
            print(f"Sample {id}: Error with val_info")

        if "train_info" in brain2paths[id].keys():
            try:
                url = brain2paths[id]["train_info"]["url"]
                layer = brain2paths[id]["train_info"]["layer"]
                pts = json_to_points(url)[layer]
            except:
                print(f"Sample {id}: Error with train_info")
```

(continues on next page)

(continued from previous page)

```
    else:
        print(f"Sample {id}: Does not conform to desired format")
```

Steps 2, 4-6 below can be done alternatively via a script with: `brainlit/BrainLine/scripts/axon_validate.py`

2. Download benchmark data

Inputs

[ ]:
```
antibody_layer = "antibody"
background_layer = "background"
endogenous_layer = "endogenous"

brain = "test"  # brain ID
axon_data_dir = (
    str(brainlit_path.parents[0]) + "/"
)  # path to directory where training/validation data should be stored
dataset_to_save = "val"  # train or val
```

Setup paths

[ ]:
```
cvol_base = brain2paths[brain]["base"]
layer_names = [antibody_layer, background_layer, endogenous_layer]

for layer in [antibody_layer, background_layer, endogenous_layer]:
    try:
        CloudVolume(cvol_base + layer)
    except:
        print(f"Sample {id}: Layer {layer} not found in {cvol_base}")

if brain not in brain2paths.keys():
    raise ValueError(f"brain {brain} not an entry in brain2paths in axon_data.py file
↪")

if f"{dataset_to_save}_info" not in brain2paths[
    brain
].keys() or dataset_to_save not in ["train", "val"]:
    raise ValueError(f"{dataset_to_save}_info not in brain2paths[{brain}].keys()")
```

Download data

[ ]:
```
download_subvolumes(
    axon_data_dir,
    brain_id=brain,
    data_file=data_file,
    layer_names=layer_names,
    dataset_to_save=dataset_to_save,
)
```

3. View downloaded data (optional)

Inputs

```
[ ]: fname = "/Users/thomasathey/Documents/mimlab/mouselight/brainlit_parent/braintest/val/
     ↪2931_4163_1602.h5"  # path to file for viewing
     scale = [1.8, 1.8, 2]  # voxel size in microns
```

```
[ ]: with h5py.File(fname, "r") as f:
         pred = f.get("image_3channel")
         image_bg = pred[0, :, :, :]
         image_fg = pred[1, :, :, :]
         image_endo = pred[2, :, :, :]

     viewer = napari.Viewer(ndisplay=3)
     viewer.add_image(image_fg, scale=scale)
     viewer.add_image(image_bg, scale=scale)
     viewer.add_image(image_endo, scale=scale)
     viewer.scale_bar.visible = True
     viewer.scale_bar.unit = "um"
```

4. Apply Ilastik to validation data

You will need to do two things: - add annotations to the downloaded data (for me, partial labels on 3 of the z-slices using ilastik) - apply axon segmentation model to the downloaded data. Results should be located in the same directory at the subvolumes, with the addition of "_Probabilities" appended to the file names: you can do this programmatically (below), or you can use the ilastik GUI (which is sometimes faster)

Note: make sure foreground/background labels are matched between the model and your annotations (for me, blue/1 =axon yellow/0=bg)

```
[ ]: project_path = f"/Users/thomasathey/Documents/mimlab/mouselight/ailey/detection_axon/
     ↪axon_segmentation.ilp"  # path to ilastik model to be used
     ilastik_path = (
         "/Applications/ilastik-1.4.0b21-OSX.app/Contents/ilastik-release/run_ilastik.sh"
     )
     brains = [brain]
```

```
[ ]: applyilastik = ApplyIlastik(
         ilastk_path=ilastik_path,
         project_path=project_path,
         brains_path=axon_data_dir,
         brains=brains,
     )
     applyilastik.process_subvols()
```

5. Check results

```
[ ]: plot_results(
        data_dir=axon_data_dir, brain_ids=[brain], positive_channel=1, object_type="axon"
    )
```

If results above are not adequate improve the model and try again

In my case, I identify more subvolumes from the sample at hand using the same process as for validation data, and add it as training data to the model and retrain.

Examine best threshold

```
[ ]: examine_threshold(
        data_dir=axon_data_dir,
        brain_id=brain,
        threshold=0.38,
        object_type="axon",
        positive_channel=1,
    )
```

6. Make annotation layers

Transformed layers

```
[ ]: atlas_vol = CloudVolume(
        "precomputed://https://open-neurodata.s3.amazonaws.com/ara_2016/sagittal_10um/
    ↪annotation_10um_2017"
    )
    for layer in [
        antibody_layer,
        background_layer,
        "axon_mask",
    ]:  # axon_mask is transformed into an image because nearest interpolation doesnt␣
    ↪work well after downsampling
        layer_path = brain2paths[brain]["base"] + layer + "_transformed"
        info = CloudVolume.create_new_info(
            num_channels=1,
            layer_type="image",
            data_type="uint16",  # Channel images might be 'uint8'
            encoding="raw",  # raw, jpeg, compressed_segmentation, fpzip, kempressed
            resolution=atlas_vol.resolution,  # Voxel scaling, units are in nanometers
            voxel_offset=atlas_vol.voxel_offset,
            chunk_size=[32, 32, 32],  # units are voxels
            volume_size=atlas_vol.volume_size,  # e.g. a cubic millimeter dataset
        )
        vol_mask = CloudVolume(layer_path, info=info)
        vol_mask.commit_info()
```

7. Apply ilastik to whole image:

This can be done alternatively via a script with: `brainlit/BrainLine/soma_detect_image`

* Inputs *

```
[ ]: threshold = 0.38  # threshold to use for ilastik
data_dir = (
    axon_data_dir + "brain_temp/"
)  # data_dir = "/data/tathey1/matt_wright/brain_temp/"  # directory to store␣
↪temporary subvolumes for segmentation

# Ilastik will run in "headless mode", and the following paths are needed to do so:
ilastik_path = "/Applications/ilastik-1.4.0b21-OSX.app/Contents/ilastik-release/run_
↪ilastik.sh"  # "/data/tathey1/matt_wright/ilastik/ilastik-1.4.0rc5-Linux/run_
↪ilastik.sh"  # path to ilastik executable
ilastik_project = "/Users/thomasathey/Documents/mimlab/mouselight/ailey/detection_
↪axon/axon_segmentation.ilp"  # "/data/tathey1/matt_wright/ilastik/model1/axon_
↪segmentation.ilp"  # path to ilastik project


max_coords = [
    3072,
    4352,
    1792,
]  # max coords or -1 if you want to process everything along that dimension
ncpu = 1  # 16  # number of cores to use for detection
chunk_size = [256, 256, 256]  # [256, 256, 300]
```

```
[ ]: layer_names = [antibody_layer, background_layer, endogenous_layer]
alli = ApplyIlastik_LargeImage(
    ilastik_path=ilastik_path,
    ilastik_project=ilastik_project,
    ncpu=ncpu,
    data_file=data_file,
)
# alli.apply_ilastik_parallel(
#     brain_id=brain,
#     layer_names=layer_names,
#     threshold=threshold,
#     data_dir=data_dir,
#     chunk_size=chunk_size,
#     max_coords=max_coords,
# )
alli.collect_axon_results(brain_id=brain, ng_layer_name="antibody")
```

8. Register volume and transform data to atlas space using CloudReg

8a. You need to find an initial affine alignment using cloudreg.scripts.registration.get_affine_matrix. For example:

A link to the ARA parcellation is:

`precomputed://https://open-neurodata.s3.amazonaws.com/ara_2016/sagittal_10um/annotation_10um_2017`

And some python commands to help with affine alignment is:

```
from cloudreg.scripts.registration import get_affine_matrix
get_affine_matrix([1,1,1], [15,0,0], "PIR", "RAI", 1.15, "precomputed://https://open-
↪neurodata.s3.amazonaws.com/ara_2016/sagittal_10um/annotation_10um_2017")
```

8b. Run registration using cloudreg.scripts.registration. For example:

```
python -m cloudreg.scripts.registration -input_s3_path precomputed://s3://smartspim-
↪precomputed-volumes/2022_11_01/8790/Ch_561 --output_s3_path precomputed://s3://
↪smartspim-precomputed-volumes/2022_11_01/8790/atlas_to_target --atlas_s3_path https:
↪//open-neurodata.s3.amazonaws.com/ara_2016/sagittal_50um/average_50um --
↪parcellation_s3_path https://open-neurodata.s3.amazonaws.com/ara_2016/sagittal_10um/
↪annotation_10um_2017 --atlas_orientation PIR -orientation RAI --rotation 0 0 0 --
↪translation 0 0 0 --fixed_scale 1.2 -log_s3_path precomputed://s3://smartspim-
↪precomputed-volumes/2022_11_01/8790/atlas_to_target --missing_data_correction True -
↪-grid_correction False --bias_correction True --regularization 5000.0 --iterations␣
↪3000 --registration_resolution 100
```

8c. Transform segmentation to atlas space using CloudReg

```
python -m cloudreg.scripts.transform_data --target_layer_source precomputed://s3://
↪smartspim-precomputed-volumes/2022_11_03/8589/axon_mask --transformed_layer_source␣
↪precomputed://s3://smartspim-precomputed-volumes/2022_11_03/8589/axon_mask_
↪transformed --affine_path /mnt/NAS/Neuroglancer\ Data/2021_11_03/8589/8589_Ch_561_
↪registration/downloop_1_A.mat  --velocity_path /mnt/NAS/Neuroglancer\ Data/2021_11_
↪03/8589/8589_Ch_561_registration/downloop_1_v.mat
```

This will write a layer to s3 with the transformed axon mask. The s3 path to this layer should be added to `axon_data.py` under the `axon_mask_transformed` key. Then the code below, or `axon_brainrender.py`, can be used to visualize the data.

Steps 9-11 below can be done alternatively via a script with: `brainlit/BrainLine/scripts/axon_analyze.py`

9. Combine registration and segmentation results

```
[ ]: collect_regional_segmentation(
        brain_id=brain, data_file=data_file, outdir=axon_data_dir, max_coords=max_coords
    )
```

10. View axon segmentation in brain space

Inputs

```
[ ]: brain_ids = ["test"]

colors = {
    "test_type": "red",
}  # colors for different genotypes
fold_on = True
```

```
[ ]: ad = AxonDistribution(
    brain_ids=brain_ids, data_file=data_file, regional_distribution_dir=axon_data_dir
)
```

```
[ ]: ad.napari_coronal_section(z=1000, subtype_colors=colors, fold_on=fold_on)
```

```
[ ]: ad.brainrender_axons(subtype_colors=colors)
```

11. Display bar charts

Inputs

```
[ ]: wholebrain_results_dir = ""  #

brains = [brain]  # list of sample IDs to be shown

regions = [
    688,  # cerebral cortex
    698,  # olfactory areas
    1089,  # hippocampal formation
    # 583, # claustrum
    477,  # striatum
    # 803, # pallidum
    351,  # bed nuclei of stria terminalis
    # 703, #cortical subplate
    1097,  # hypothalamus
    549,  # thalamus
    186,  # lateral habenula
    519,  # cerebellar nuclei
    313,  # midbrain
    1065,  # hindbrain
]  # allen atlas region IDs to be shown
# see: https://connectivity.brain-map.org/projection/experiment/480074702?
↪imageId=480075280&initImage=TWO_PHOTON&x=17028&y=11704&z=3

composite_regions = {
    "Amygdalar Nuclei": [131, 295, 319, 780]
}  # Custom composite allen regions where key is region name and value is list of␣
↪allen regions
```

```
[ ]: ad.region_barchart(regions, composite_regions=composite_regions, normalize_region=872)
```

[ ]:

Soma Detection Analysis of Whole-Brain Fluorescence Images

[ ]:
```
from brainlit.BrainLine.analyze_results import SomaDistribution
from brainlit.BrainLine.util import (
    json_to_points,
    download_subvolumes,
)
from brainlit.BrainLine.apply_ilastik import (
    ApplyIlastik,
    ApplyIlastik_LargeImage,
    plot_results,
    examine_threshold,
)
from brainlit.BrainLine.parse_ara import *
import xml.etree.ElementTree as ET
from brainlit.BrainLine.imports import *

%gui qt5
```

1. Before Using this notebook

1a. Install brainlit, and dependencies

1b. Write images to s3 using CloudReg

```
- e.g. python -m cloudreg.scripts.create_precomputed_volumes --s3_input_paths <path-
↪to-stitched-images> --s3_output_paths  <s3-address-for-output>  --voxel_size <x-
↪resolution> <y-resolution> <z-resolution> --num_procs <num-cpus> --resample_iso
↪<boolean-to-resample-isotropically>
```

1c. Make point annotations in neuroglancer to identify subvolumes for validation (and possible training)

```
- instructions: https://neurodata.io/help/neuroglancer-pt-annotations/
- For me, this is the json snippet that I add:
,
{
"type":"pointAnnotation",
"name": "soma_val",
"points": []
},
{
"type":"pointAnnotation",
"name": "nonsoma_val",
"points":[]
}
```

1d. Update soma_data.py file

```
[ ]: brainlit_path = Path(os.path.abspath(""))
brainlit_path = brainlit_path.parents[3]
print(f"Path to brainlit: {brainlit_path}")
data_file = brainlit_path / "brainlit" / "BrainLine" / "data" / "soma_data.json"

with open(data_file) as f:
    data = json.load(f)
brain2paths = data["brain2paths"]

for id in brain2paths.keys():
    if "base" in brain2paths[id].keys() and "val_info" in brain2paths[id].keys():
        base = brain2paths[id]["base"]
        if "http" in base:
            print(f"Sample {id}: http in basepath, which may cause write errors")

        try:
            url = brain2paths[id]["val_info"]["url"]
            layer = brain2paths[id]["val_info"]["somas_layer"]
            pts = json_to_points(url)[layer]
            layer = brain2paths[id]["val_info"]["nonsomas_layer"]
            pts = json_to_points(url)[layer]
        except:
            print(f"Sample {id}: Error finding validation annotations with val_info")

        if "train_info" in brain2paths[id].keys():
            try:
                url = brain2paths[id]["train_info"]["url"]
                layer = brain2paths[id]["train_info"]["somas_layer"]
                pts = json_to_points(url)[layer]
                layer = brain2paths[id]["train_info"]["nonsomas_layer"]
                pts = json_to_points(url)[layer]
            except:
                print(
                    f"Sample {id}: Error finding training annotations with train_info"
                )
    else:
        print(f"Sample {id}: Does not conform to desired format")
```

Steps 2, 4-6 below can be done via script with: `brainlit/BrainLine/scripts/soma_validate.py`

2. Download benchmark data

Inputs

```
[ ]: brain = "test"  # brain ID
soma_data_dir = (
    str(brainlit_path.parents[0]) + "/"
)  # path to directory where training/validation data should be stored
dataset_to_save = "val"  # train or val

antibody_layer = "antibody"
```

(continues on next page)

(continued from previous page)

```
background_layer = "background"
endogenous_layer = "endogenous"
```

Setup paths

[ ]:
```
cvol_base = brain2paths[brain]["base"]
layer_names = [antibody_layer, background_layer, endogenous_layer]

if brain not in brain2paths.keys():
    raise ValueError(f"brain {brain} not an entry in brain2paths in axon_data.py file
↪")

if f"{dataset_to_save}_info" not in brain2paths[
    brain
].keys() or dataset_to_save not in ["train", "val"]:
    raise ValueError(f"{dataset_to_save}_info not in brain2paths[{brain}].keys()")


for layer in [antibody_layer, background_layer, endogenous_layer]:
    try:
        CloudVolume(cvol_base + layer)
    except:
        print(f"Sample {id}: Layer {layer} not found in {cvol_base}")
```

Download data

[ ]:
```
download_subvolumes(
    soma_data_dir,
    brain_id=brain,
    data_file=data_file,
    layer_names=layer_names,
    dataset_to_save=dataset_to_save,
)
```

3. View downloaded data (optional)

Inputs

[ ]:
```
fname = "/Users/thomasathey/Documents/mimlab/mouselight/brainlit_parent/braintest/val/
↪2936_4243_1587_pos.h5"  # path to file for viewing
scale = [1.8, 1.8, 2]  # voxel size in microns
```

[ ]:
```
with h5py.File(fname, "r") as f:
    pred = f.get("image_3channel")
    image_fg = pred[0, :, :, :]
    image_bg = pred[1, :, :, :]
    image_endo = pred[2, :, :, :]

viewer = napari.Viewer(ndisplay=3)
```

(continues on next page)

(continued from previous page)

```
viewer.add_image(image_fg, scale=scale)
viewer.add_image(image_bg, scale=scale)
viewer.add_image(image_endo, scale=scale)
viewer.scale_bar.visible = True
viewer.scale_bar.unit = "um"
```

4. Apply ilastik to validation data

You can do this programmatically (below), or you can use the ilastik GUI (which is sometimes faster)

* Inputs *

```
[ ]: project_path = f"/Users/thomasathey/Documents/mimlab/mouselight/ailey/detection_soma/
     ↪matt_soma_rabies_pix_3ch.ilp"  # path to ilastik model to be used
     ilastik_path = (
         "/Applications/ilastik-1.4.0b21-OSX.app/Contents/ilastik-release/run_ilastik.sh"
     )
     brains = [brain]
```

```
[ ]: applyilastik = ApplyIlastik(
         ilastk_path=ilastik_path,
         project_path=project_path,
         brains_path=soma_data_dir,
         brains=brains,
     )
     applyilastik.process_subvols()
     # applyilastik.move_results()
```

*Inputs *

- identify files that have two somas in variable below. Since voxel coordinates are likely to be unique across samples, the file names below do not include sample IDs.

```
[ ]: doubles = []  # e.g. ["3972_1636_1575_pos_Probabilities.h5",]
```

5. Check Results

Validation

```
[ ]: plot_results(
         data_dir=soma_data_dir,
         brain_ids=[brain],
         object_type="soma",
         positive_channel=0,
         doubles=doubles,
     )
```

If results above are not adequate, improve model and try again

In my case, I identify more subvolumes from the sample at hand using the same process as for validation data, and add it as training data to the model and retrain.

Examine best threshold

```
[ ]: examine_threshold(
        data_dir=soma_data_dir,
        brain_id=brain,
        threshold=0.2,
        object_type="soma",
        positive_channel=0,
        doubles=doubles,
    )
```

6. Make Annotation layers

Transformed layers

```
[ ]: atlas_vol = CloudVolume(
        "precomputed://https://open-neurodata.s3.amazonaws.com/ara_2016/sagittal_10um/
    ↪annotation_10um_2017"
    )
    for layer in [
        antibody_layer,
        background_layer,
    ]:  # axon_mask is transformed into an image because nearest interpolation doesnt␣
    ↪work well after downsampling
        layer_path = brain2paths[brain]["base"] + layer + "_transformed"
        info = CloudVolume.create_new_info(
            num_channels=1,
            layer_type="image",
            data_type="uint16",  # Channel images might be 'uint8'
            encoding="raw",  # raw, jpeg, compressed_segmentation, fpzip, kempressed
            resolution=atlas_vol.resolution,  # Voxel scaling, units are in nanometers
            voxel_offset=atlas_vol.voxel_offset,
            chunk_size=[32, 32, 32],  # units are voxels
            volume_size=atlas_vol.volume_size,  # e.g. a cubic millimeter dataset
        )
        vol_mask = CloudVolume(layer_path, info=info)
        vol_mask.commit_info()
```

7. Apply ilastik to whole image

This can be done via a script with: `brainlit/BrainLine/soma_detect_image`

* Inputs *

```
[ ]: threshold = 0.2  # threshold to use for ilastik
data_dir = (
    soma_data_dir + "brainr_temp/"
)  # "/data/tathey1/matt_wright/brainr_temp/"  # directory to store temporary␣
↪subvolumes for segmentation
results_dir = (
    soma_data_dir + "brainr_results/"
)  # directory to store coordinates of soma detections

# Ilastik will run in "headless mode", and the following paths are needed to do so:
ilastik_path = "/Applications/ilastik-1.4.0b21-OSX.app/Contents/ilastik-release/run_
↪ilastik.sh"  # "/data/tathey1/matt_wright/ilastik/ilastik-1.4.0rc5-Linux/run_
↪ilastik.sh"  # path to ilastik executable
ilastik_project = "/Users/thomasathey/Documents/mimlab/mouselight/ailey/detection_
↪soma/matt_soma_rabies_pix_3ch.ilp"  # "/data/tathey1/matt_wright/ilastik/soma_model/
↪matt_soma_rabies_pix_3ch.ilp"  # path to ilastik project

max_coords = [3072, 4352, 1792]  # -1 if you want to process the whole dimension
ncpu = 1  # 16  # number of cores to use for detection
chunk_size = [256, 256, 256]  # [256, 256, 300]
```

```
[ ]: layer_names = [antibody_layer, background_layer, endogenous_layer]

ilastik_largeimage = ApplyIlastik_LargeImage(
    ilastik_path=ilastik_path,
    ilastik_project=ilastik_project,
    data_file=data_file,
    results_dir=results_dir,
    ncpu=1,
)
ilastik_largeimage.apply_ilastik_parallel(
    brain_id=brain,
    layer_names=layer_names,
    threshold=threshold,
    data_dir=data_dir,
    chunk_size=chunk_size,
    max_coords=max_coords,
)
```

Before this step you will need to make sure that the original data is being served to neuroglancer. For example, in this case, our data is local so we can serve it with the command:

```
python cors_webserver.py -d "<path-to-brainlit>/brainlit/brainlit/BrainLine/
data/example" -p 9010
```

which needs to be run in the neuroglancer folder (git clone from here: https://github.com/google/neuroglancer)

```
[ ]: ilastik_largeimage.collect_soma_results(brain_id="test")
```

8. Register volume and transform data to atlas space using CloudReg

8a. You need to find an initial affine alignment using cloudreg.scripts.registration.get_affine_matrix. For example:

A link to the ARA parcellation is:

`precomputed://https://open-neurodata.s3.amazonaws.com/ara_2016/sagittal_10um/annotation_10um_2017`

And some python commands to help with affine alignment is:

```
from cloudreg.scripts.registration import get_affine_matrix
get_affine_matrix([1,1,1], [15,0,0], "PIR", "RAI", 1.15, "precomputed://https://open-
↪neurodata.s3.amazonaws.com/ara_2016/sagittal_10um/annotation_10um_2017")
```

8b. Run registration using cloudreg.scripts.registration. For example:

```
python -m cloudreg.scripts.registration -input_s3_path precomputed://s3://smartspim-
↪precomputed-volumes/2023_01_20/MPRRabies/Ch_561 --output_s3_path precomputed://s3://
↪smartspim-precomputed-volumes/2023_01_20/MPRRabies/atlas_to_target --atlas_s3_path␣
↪https://open-neurodata.s3.amazonaws.com/ara_2016/sagittal_50um/average_50um --
↪parcellation_s3_path https://open-neurodata.s3.amazonaws.com/ara_2016/sagittal_10um/
↪annotation_10um_2017 --atlas_orientation PIR -orientation RPI --rotation 0 0 0 --
↪translation 0 0 0 --fixed_scale 1.07 -log_s3_path precomputed://s3://smartspim-
↪precomputed-volumes/2023_01_20/MPRRabies/atlas_to_target --missing_data_correction␣
↪True --grid_correction False --bias_correction True --regularization 5000.0 --
↪iterations 3000 --registration_resolution 100
```

8c. Transform data to atlas space using CloudReg

Soma coordinates

```
python -m cloudreg.scripts.transform_points --target_viz_link https://viz.neurodata.
↪io/?json_url=https://json.neurodata.io/v1?NGStateID=6ti276yAxXF_Rw --atlas_viz_link␣
↪https://ara.viz.neurodata.io/?json_url=https://json.neurodata.io/v1?
↪NGStateID=HvyNDGaPsd1wyg --affine_path /mnt/NAS/Neuroglancer\ Data/  --velocity_
↪path /mnt/NAS/Neuroglancer\ Data/  --transformation_direction atlas
```

or

```
python -m cloudreg.scripts.transform_points --target_viz_link https://viz.neurodata.
↪io/?json_url=https://json.neurodata.io/v1?NGStateID=05Fhxt5VBT-_1A --atlas_viz_link␣
↪https://ara.viz.neurodata.io/?json_url=https://json.neurodata.io/v1?
↪NGStateID=HvyNDGaPsd1wyg --affine_path /cis/home/tathey/MPRRabies_Ch_561_
↪registration/downloop_1_A.mat   --velocity_path /cis/home/tathey/MPRRabies_Ch_561_
↪registration/downloop_1_v.mat  --transformation_direction atlas
```

This will produce a neuroglancer link with the transformed soma coordinates, which should be added to `soma_data.py` under the `somas_atlas_url` key. Then the code below, or `soma_brainrender.py`, can be used to visualize the data.

Image

```
python -m cloudreg.scripts.transform_data --target_layer_source precomputed://s3://
↪smartspim-precomputed-volumes/2022_09_20/887/Ch_647 --transformed_layer_source␣
↪precomputed://s3://smartspim-precomputed-volumes/2022_09_20/887/Ch_647_transformed -
↪-affine_path /cis/home/tathey/887_Ch_561_registration/downloop_1_A.mat  --velocity_
↪path /cis/home/tathey/887_Ch_561_registration/downloop_1_v.mat
```

Steps 9-10 below can be done via a script with: `brainlit/BrainLine/scripts/soma_analyze.py`

9. View somas in brain space

Inputs

[ ]:
```
# colors = {
#     "tph2 vglut3": "blue",
#     "tph2 gad2": "red",
#     "gad2 vgat": "green",
# }  # colors for different genotypes
colors = {
    "test_type": "red",
}  # colors for different genotypes
symbols = ["o", "+", "^", "vbar"]
brain_ids = ["test"]
fold_on = True
```

[ ]:
```
sd = SomaDistribution(brain_ids=brain_ids, data_file=data_file)
sd.napari_coronal_section(
    z=1000, subtype_colors=colors, symbols=symbols, fold_on=fold_on
)
```

[ ]:
```
sd.brainrender_somas(subtype_colors=colors)
```

10. Display bar charts

[ ]:
```
regions = [
    688,  # cerebral cortex
    698,  # olfactory areas
    1089,  # hippocampal formation
    # 583, # claustrum
    477,  # striatum
    # 803, # pallidum
    351,  # bed nuclei of stria terminalis
    # 703, #cortical subplate
    1097,  # hypothalamus
    549,  # thalamus
    186,  # lateral habenula
    519,  # cerebellar nuclei
    313,  # midbrain
    1065,  # hindbrain
```

(continues on next page)

(continued from previous page)

```
]  # allen atlas region IDs to be shown
# see: https://connectivity.brain-map.org/projection/experiment/480074702?
↪imageId=480075280&initImage=TWO_PHOTON&x=17028&y=11704&z=3

composite_regions = {
    "Amygdalar Nuclei": [131, 295, 319, 780]
}  # Custom composite allen regions where key is region name and value is list of␣
↪allen regions

brain_ids = ["test"]
sd = SomaDistribution(brain_ids=brain_ids, data_file=data_file)
sd.region_barchart(regions, composite_regions=composite_regions, normalize_region=872)
```

[ ]:

Semi-automatic Annotation Pipeline

Demonstrate pulling data and pushing traced annotations.

Automatic and manual segmentation pipeline

[ ]:
```
import brainlit
from brainlit.utils.session import NeuroglancerSession
from brainlit.utils.Neuron_trace import NeuronTrace
from brainlit.algorithms.generate_fragments import adaptive_thresh
import napari
from napari.utils import nbscreenshot

%gui qt5
```

Find valid segments

In this cell, we set up a NeuroglancerSession object. Since segmentation ID numbers are not in order, we print out a list of valid IDs in some range `id_range`. Most segment IDs are in `range(300)`, additionally, segments `999` and `1000` are available.

[ ]:
```
"""
# Optional: Print the IDs of segments in Neuroglancer
url = "s3://open-neurodata/brainlit/brain1"
ngl_skel = NeuroglancerSession(url+"_segments", mip=1,use_https=False)
working_ids = []
id_range = 14
for seg_id in range(id_range):
    try:
        segment = ngl_skel.cv.skeleton.get(seg_id)
        working_ids.append(seg_id)
    except:
          pass
print(working_ids)
"""
```

Download SWC information

Download the information contained in a SWC file for labelled vertices of a given `seg_id` at a valid `mip` from AWS.

[ ]:
```
"""
seg_id = 13
mip = 2
s3_trace = NeuronTrace(url+"_segments", seg_id, mip)
df = s3_trace.get_df()
df['sample'].size # the number of vertex IDs [1, 2, ..., df['sample'].size]
"""
```

[ ]:
```
"""
print(df)
"""
```

Select vertices

Select a subset of the vertices in the downloaded SWC to view and segment.

[ ]:
```
"""
subneuron_df = df[0:5] # choose vertices to use for the subneuron
vertex_list = subneuron_df['sample'].array
print(vertex_list)
"""
```

Download the Volume

Download the volume containing the specified vertices.

[ ]:
```
"""
ngl = NeuroglancerSession(url, mip=mip)
buffer = 10
img, bounds, vox_in_img_list = ngl.pull_vertex_list(seg_id, vertex_list, buffer =␣
↪buffer, expand = True)
"""
```

Plot

[ ]:
```
"""
# Reference: https://github.com/NeuroDataDesign/mouselit/blob/master/bijan/mouse_test/
↪final%20notebook.ipynb
def napari_viewer(img, labels=None, shapes=None, label_name="Segmentation"):
    viewer = napari.view_image(np.squeeze(np.array(img)))
    if labels is not None:
        viewer.add_labels(labels, name=label_name)
    if shapes is not None:
        viewer.add_shapes(data=shapes, shape_type='path', edge_color='blue', name=
↪'Skeleton')
    return viewer
"""
```

Let's take a look at the downloaded volume. Napari will open in a new window.

[ ]:
```
"""
viewer = napari.Viewer(ndisplay=3)
viewer.add_image(img)
nbscreenshot(viewer)
"""
```

[ ]:
```
"""
n=napari_viewer(img)
"""
```

[ ]:
```
"""
import inspect
a = repr(n)
print(a)

b = repr(n).find(('napari.viewer.Viewer'))
print(b)
"""
```

[ ]:
```
"""
n.window.close()
"""
```

[ ]:
```
# We get a `corrected_subneuron_df` that contains `(x,y,z)` coordinates within the␣
↪downloaded volume for the vertices in the SWC.
```

[ ]:
```
"""
import inspect
a = repr(n)
print(a)

b = repr(n).find(('napari.viewer.Viewer'))
print(b)
"""
```

[ ]:
```
# We get a `corrected_subneuron_df` that contains `(x,y,z)` coordinates within the␣
↪downloaded volume for the vertices in the SWC.
```

[ ]:
```
"""
import inspect
a = repr(n)
print(a)

b = repr(n).find(('napari.viewer.Viewer'))
print(b)
"""
```

[ ]:
```
# We get a `corrected_subneuron_df` that contains `(x,y,z)` coordinates within the␣
↪downloaded volume for the vertices in the SWC.
```

[ ]:
```
"""
import inspect
```

(continues on next page)

(continued from previous page)

```
a = repr(n)
print(a)

b = repr(n).find(('napari.viewer.Viewer'))
print(b)
"""
```

[ ]:
```
# We get a `corrected_subneuron_df` that contains `(x,y,z)` coordinates within the␣
↪downloaded volume for the vertices in the SWC.
```

[ ]:
```
"""
transpose = vox_in_img_list.T
vox_in_img_list_t = transpose.tolist()

corrected_subneuron_df = s3_trace.generate_df_subset(list(vox_in_img_list_t),␣
↪subneuron_start = 0, subneuron_end = 5)
print(corrected_subneuron_df)
"""
```

Convert the SWC to a graph and print some information about the graph.

[ ]:
```
"""
G = s3_trace._df_to_graph(df_voxel=corrected_subneuron_df)
print('Number of nodes:', len(G.nodes))
print('Number of edges:', len(G.edges))
print('Sample 1 coordinates (x,y,z):', G.nodes[1])
paths = s3_trace._graph_to_paths(G)
print('Number of paths:', len(paths))
"""
```

We can display the SWC on the Volume

[ ]:
```
"""
%gui qt
napari_viewer(img, shapes=paths)
nbscreenshot(viewer)
"""
```

Automatically segment the neuron

We start by converting the seed points to a format used by the thresholding.

[ ]:
```
"""
seed = [adaptive_thresh.get_seed(sample)[1] for sample in vox_in_img_list]
print(seed)
"""
```

Next, we compute a confidence-connected threshold segmentation.

[ ]:
```
"""
labels = adaptive_thresh.confidence_connected_threshold(img, seed, num_iter=1,␣
↪multiplier=0.5)
"""
```

We can display the volume, SWC, and segmentation in Napari.

[ ]:
```
"""
%gui qt
viewer = napari_viewer(img, labels=labels, shapes=paths, label_name="Confidence-
↪Connected Threshold")
nbscreenshot(viewer)
"""
```

Steps to Manually Edit Labels

Labels can be manually edited following these steps:

1. Ensure Napari is in 2D-slice viewing, not 3D view. (The second button from the bottom left)
2. Click the image layer and adjust the contrast limits as desired.
3. Click the "Confidence-Connected Threshold Layer"
4. Click the paintbrush tool and adjust the brush size. Ensure that "label" is set to 1 to paint and 0 to erase.
5. Click and drag on the image to adjust labels. Changes are saved automatically, and CMD-Z to undo is supported.

Extract the manual labels for uploading.

[ ]:
```
# manual_labels = viewer.layers['Confidence-Connected Threshold'].data
```

Upload the segmentation to AWS.

[ ]:
```
# %%capture
# ngl_upload = NeuroglancerSession(url+"_seg", mip=mip)
# ngl_upload.push(manual_labels, bounds);
```

Confirm that the upload was successful. It was!

[ ]:
```
# downloaded_labels = ngl_upload.pull_bounds_seg(bounds)
```

[ ]:
```
# print(np.all(manual_labels == downloaded_labels))
```

Segmentation

Notebooks showing how to manually and automatically segment data.

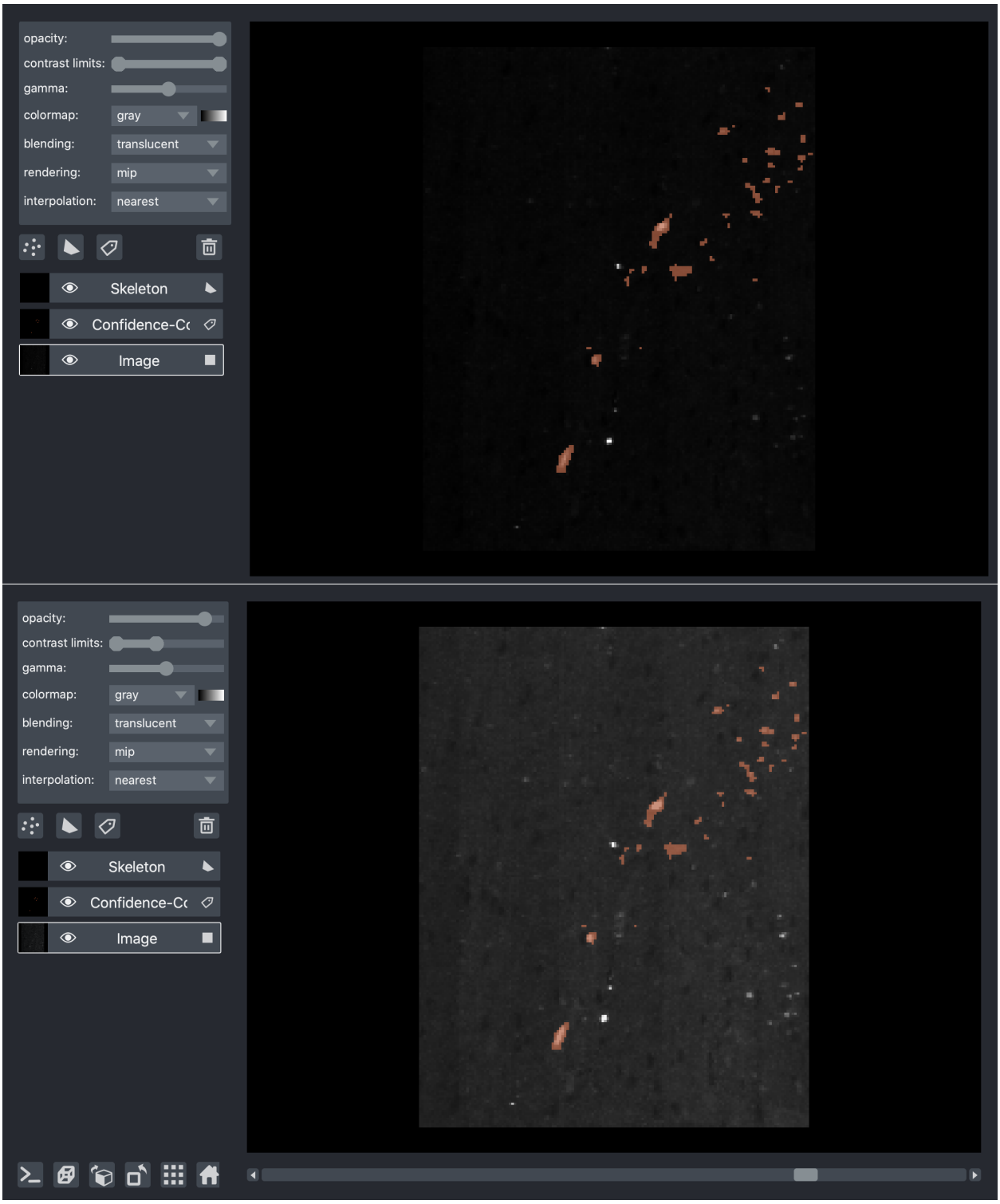
Napari Manual Segmentation Tutorial

This notebook demonstrates how to use Napari to manually edit a segmentation. To learn about generating an automatic segmentation and uploading segmentations to the cloud, refer to the segmentation pipeline demo.

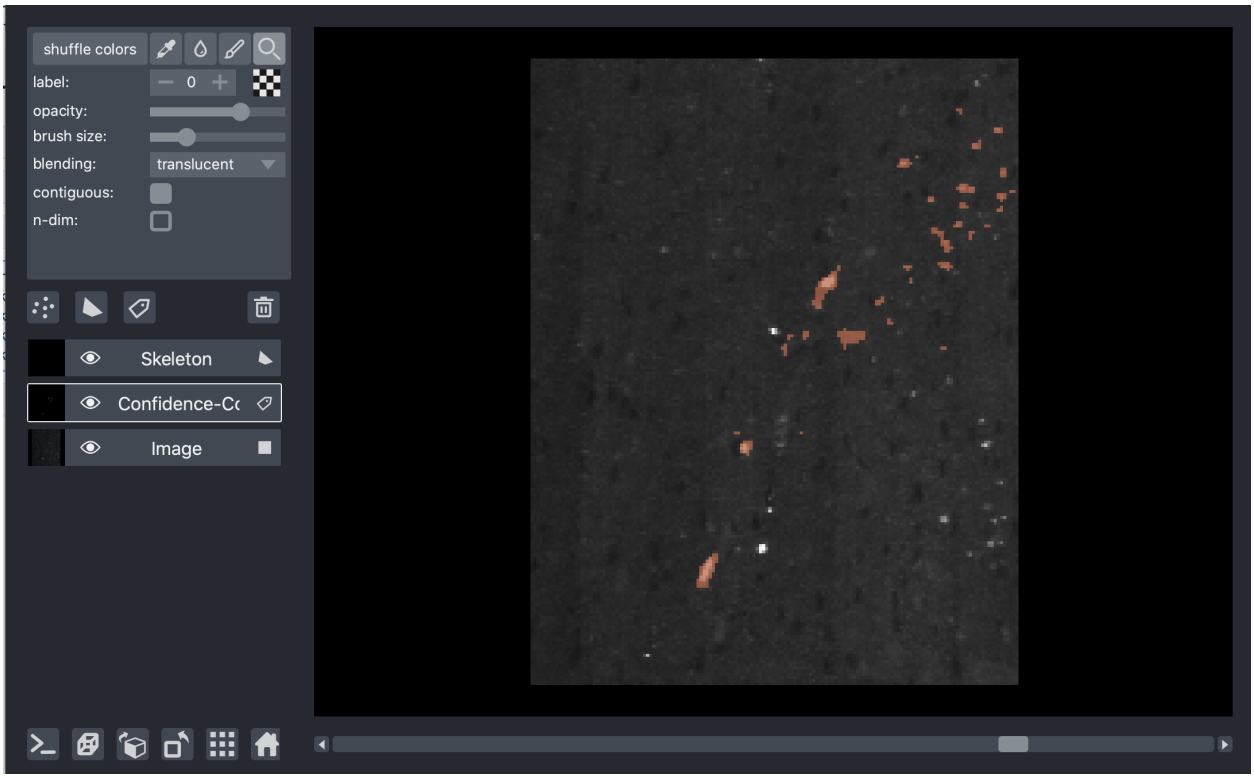
Ensure Napari is in 2D-slice viewing

1. Ensure Napari is in 2D-slice viewing, not 3D view (The second button from the bottom left).

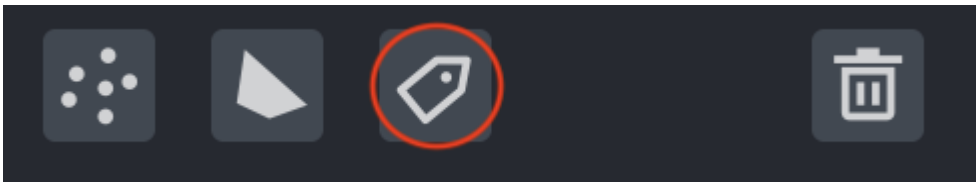
2. Click the image layer and adjust the opacity, contrast limits, and gamma as desired.



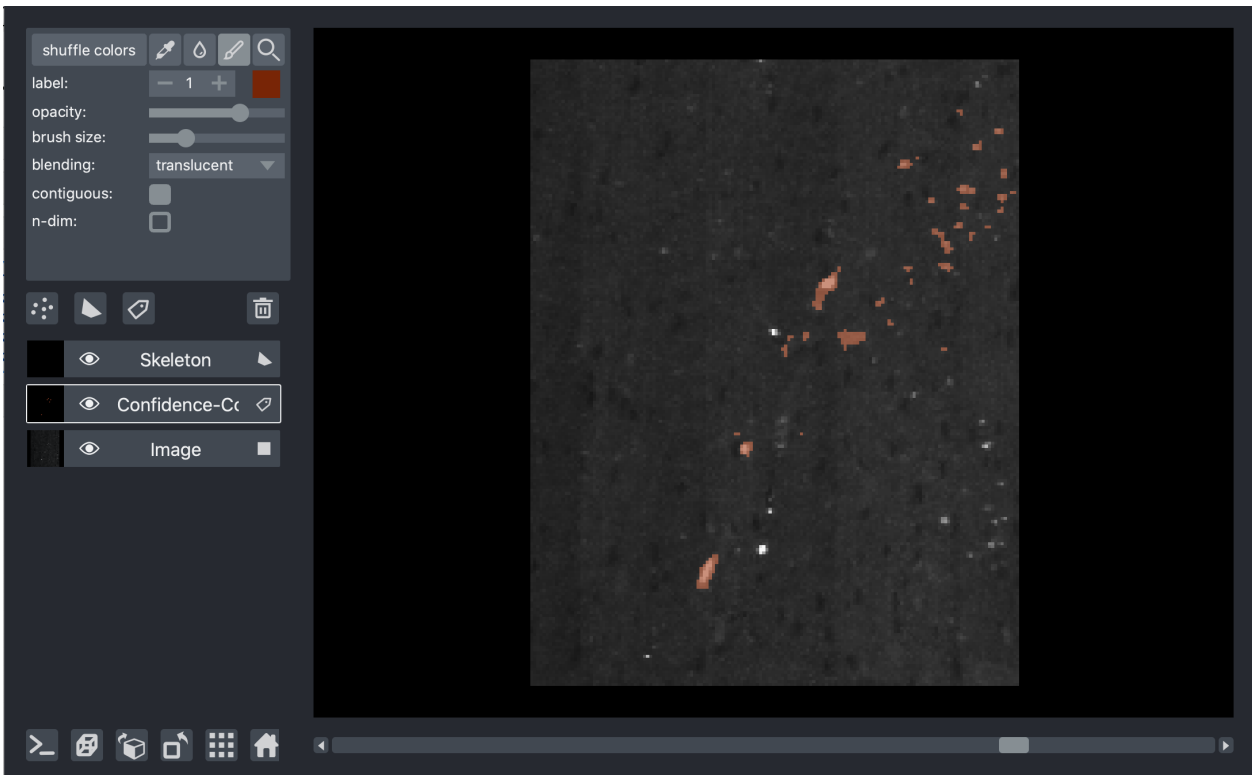
3. Click on the layer for your existing automatic segmentation mask.



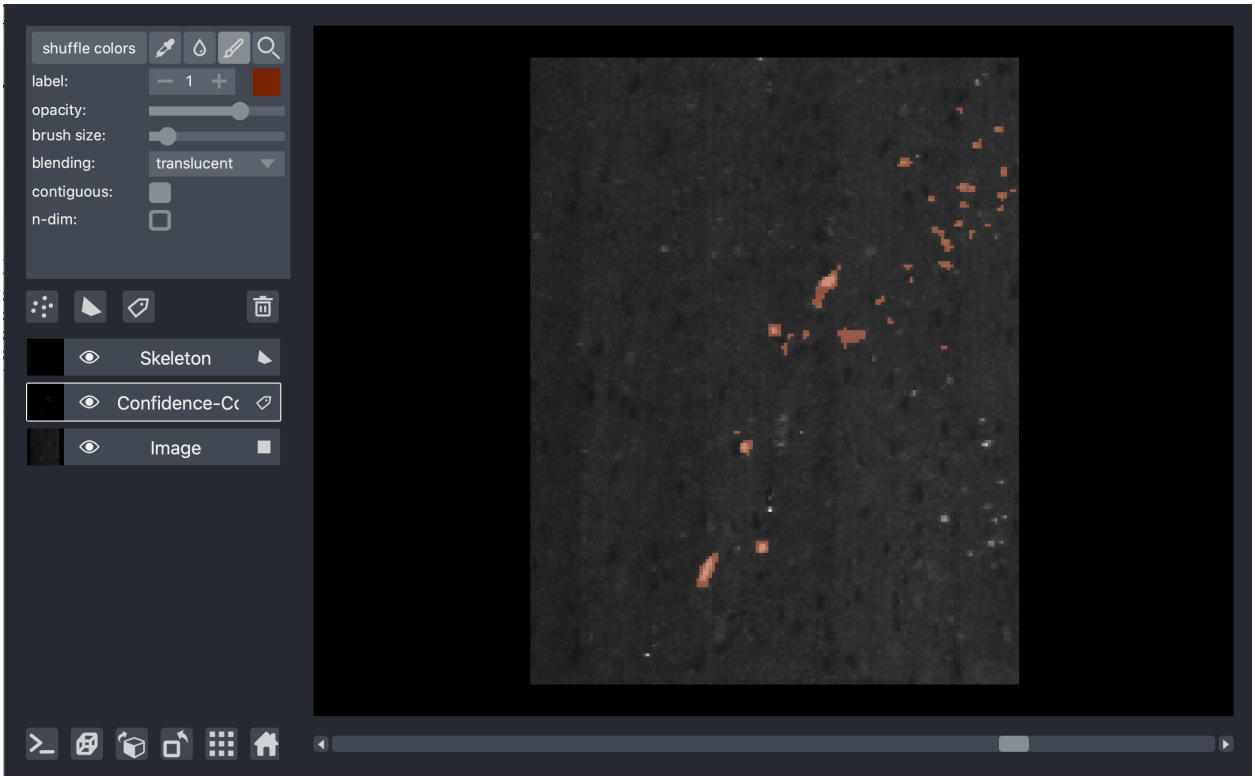
Alternatively, create a new layer for the segmentation mask.



4. Click the paintbrush tool and adjust the brush size. Ensure that "label" is set to 1 to paint and 0 to erase.



5. Click and drag on the image to adjust labels. Changes are saved automatically, and CMD-Z to undo is supported.



[ ]:

Tube segmentation

[1]:
```
from brainlit.utils import session
from brainlit.feature_extraction import *
import napari
```

```
/Users/thomasathey/Documents/mimlab/mouselight/env/lib/python3.8/site-packages/python_
↪jsonschema_objects/__init__.py:50: UserWarning: Schema version http://json-schema.
↪org/draft-04/schema not recognized. Some keywords and features may not be supported.
  warnings.warn(
```

[2]:
```
url = "s3://open-neurodata/brainlit/brain1"
sess = session.NeuroglancerSession(url=url, url_segments=url + "_segments", mip=0)
SEGLIST = [
    101,
    103,
    106,
    107,
    109,
    11,
    111,
    112,
    115,
```

(continues on next page)

(continued from previous page)

```
    11,
    12,
    120,
    124,
    126,
    127,
    129,
    13,
    132,
    133,
    136,
    137,
    14,
    140,
    141,
    142,
    143,
    144,
    145,
    146,
    147,
    149,
    150,
]
SEGLIST = SEGLIST[:1]
```

[3]:
```
# %%capture
nbr = NeighborhoodFeatures(
    url=url, radius=1, offset=[50, 50, 50], segment_url=url + "_segments"
)
nbr.fit(seg_ids=SEGLIST, num_verts=10, file_path="demo", batch_size=10)
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  9.18it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/2 [00:00<?, ?it/s]
Downloading:   0%|          | 0/2 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/2 [00:00<?, ?it/s]
Downloading:   0%|          | 0/2 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
Downloading:   0%|          | 0/1 [00:00<?, ?it/s]
```

[4]:
```
import glob, feather
```

(continues on next page)

(continued from previous page)

```
feathers = glob.glob("*.feather")


for count, feather_file in enumerate(feathers):
    if count == 0:
        data = feather.read_dataframe(feather_file)
    else:
        df = feather.read_dataframe(feather_file)
        data = pd.concat([data, df])
data.shape
```

[4]:
```
(20, 30)
```

[5]:
```
data.head()
```

[5]:
```
   Segment  Vertex  Label      0      1      2      3      4      5      6  \
0      101       0      1  28778  30111  30120  28438  29909  30092  28315
1      101       0      0  12290  12090  12222  12340  12215  12376  12185
2      101       1      1  15429  16558  17587  15353  16662  17459  15462
3      101       1      0  12166  12290  12180  12162  12129  12124  12058
4      101       2      1  13005  12535  12333  12903  12660  12402  12846

   ...     17     18     19     20     21     22     23     24     25     26
0  ...  28751  30383  30683  29420  29565  30769  29762  28865  29364  29687
1  ...  12208  12144  11845  12030  12682  12340  12336  12194  12383  12067
2  ...  18105  14926  16441  17088  15187  16712  17474  15401  16939  18366
3  ...  12297  12090  12038  12144  11959  11929  12116  12093  12298  12407
4  ...  12502  12388  12404  12471  12860  12526  12441  12951  12771  12571

[5 rows x 30 columns]
```

[6]:
```
from sklearn.preprocessing import StandardScaler

X = data.iloc[:, 3:]
X = StandardScaler().fit_transform(X)

y = data["Label"]
```

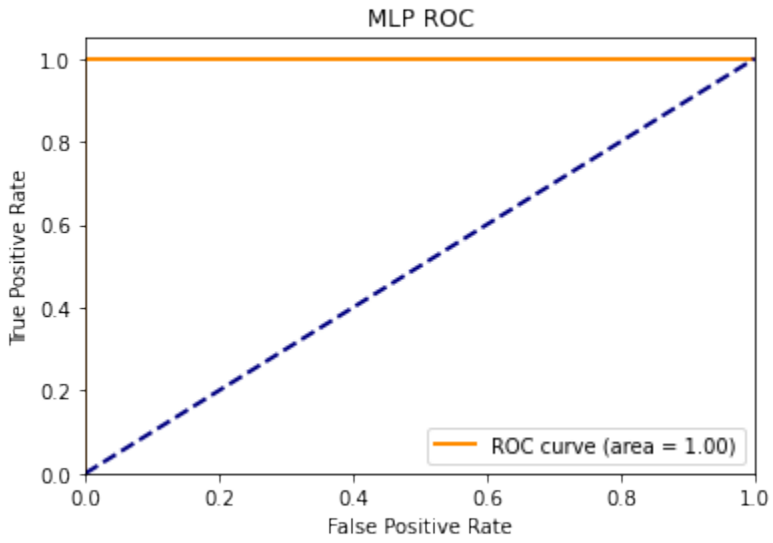
[7]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=1)
clf = MLPClassifier(
    hidden_layer_sizes=4, activation="logistic", alpha=1, max_iter=1000
).fit(X_train, y_train)
y_score = clf.predict_proba(X_test)
```

[8]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

fpr, tpr, _ = roc_curve(y_test, y_score[:, 1])
roc_auc = auc(fpr, tpr)

plt.figure()
lw = 2
plt.plot(
```

(continues on next page)

(continued from previous page)

```
    fpr, tpr, color="darkorange", lw=lw, label="ROC curve (area = %0.2f)" % roc_auc
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("MLP ROC")
plt.legend(loc="lower right")
plt.show()
```



[9]:
```
from brainlit.feature_extraction.neighborhood import subsample
```

[11]:
```
X.shape
```

[11]:
```
(20, 27)
```

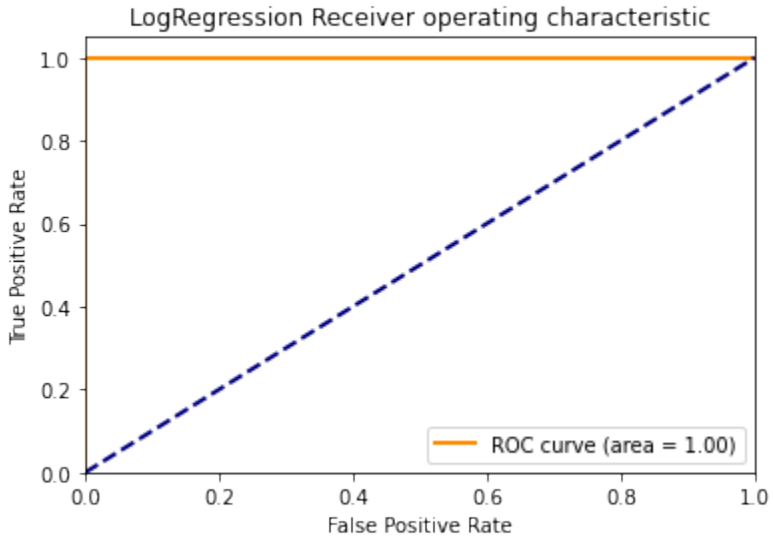
[12]:
```
from sklearn.linear_model import LogisticRegression

Xc_train, Xc_test, yc_train, yc_test = train_test_split(
    X, y, stratify=y, random_state=1
)
clf = LogisticRegression(random_state=1, max_iter=2000).fit(Xc_train, yc_train)
yc_score = clf.predict_proba(Xc_test)
```

[13]:
```
fpr_c, tpr_c, _ = roc_curve(yc_test, yc_score[:, 1])
roc_auc_c = auc(fpr_c, tpr_c)

plt.figure()
lw = 2
plt.plot(
    fpr_c,
    tpr_c,
    color="darkorange",
    lw=lw,
    label="ROC curve (area = %0.2f)" % roc_auc_c,
```

(continues on next page)

(continued from previous page)

```
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("LogRegression Receiver operating characteristic")
plt.legend(loc="lower right")
plt.show()
```



[ ]:

2.1.3 Algorithms

Adaptive Thresholding

Demonstrate region growing methods using GMM and simple ITK.

Connecting Fragments

Demonstrate fragment path connections using Viterbi algorithm on a simple grid example.

Trace Analysis

Demonstrate estimation of curvature and torsion on simple curve, and fitting splines to a neuron.

`spline_fxns` module tutorial

This tutorial showcases the usage and results of the three methods implemented in the `brainlit.algoritm.trace_analysis.spline_fxns` module:

1. `speed()`
2. `curvature()`
3. `torsion()`

Here, we will apply the module's methods to a synthetic case where

$f : u \mapsto [u^3, \sin(u), u^2], u \in [-\pi, \pi]$,

using B-Splines with order $k \in \{1, 2, 3, 4, 5\}$. The goal of the experiment is to show how changing the order of the B-Spline affects the accuracy of the methods with respects to the theoretical ground truth. We remark that `scipy.interpolate.BSpline` has a default value of $k = 3$.

First of all, it is important to remark that values of k less or equal to 2 should be avoided because they provide very poor results. Furthermore, $k = 1$ cannot be used to evaluate the curvature because B-Splines with order 1 do not have a second derivative, and $k = 2$ cannot be used to evaluate the torsion because B-Splines with order 2 do not have a third derivative. The results of this experiment suggest that it is not necessarily true that higher orders will provide more accurate results, since the accuracy varies with the value of the parameter that we are trying to estimate. For example, we will show in this experiment that a B-Spline with order 5 is better than a B-Spline with order 3 when the torsion is much greater than 0, while its performance degrades almost completely for values close to 0.

To conclude, this simple experiment wants to show the performance of the `spline_fxns` module on a synthetic, 3D curve. By changing the order of the B-Spline used to interpolate the curve, we see that the accuracy of the methods changes significantly. We do not provide a general rule to pick the best value of k, but we suggest that using $k = 3, 4$ could provide a better performance on average, avoiding singularities that can arise with $k = 5$.

0. Define and evaluate the function

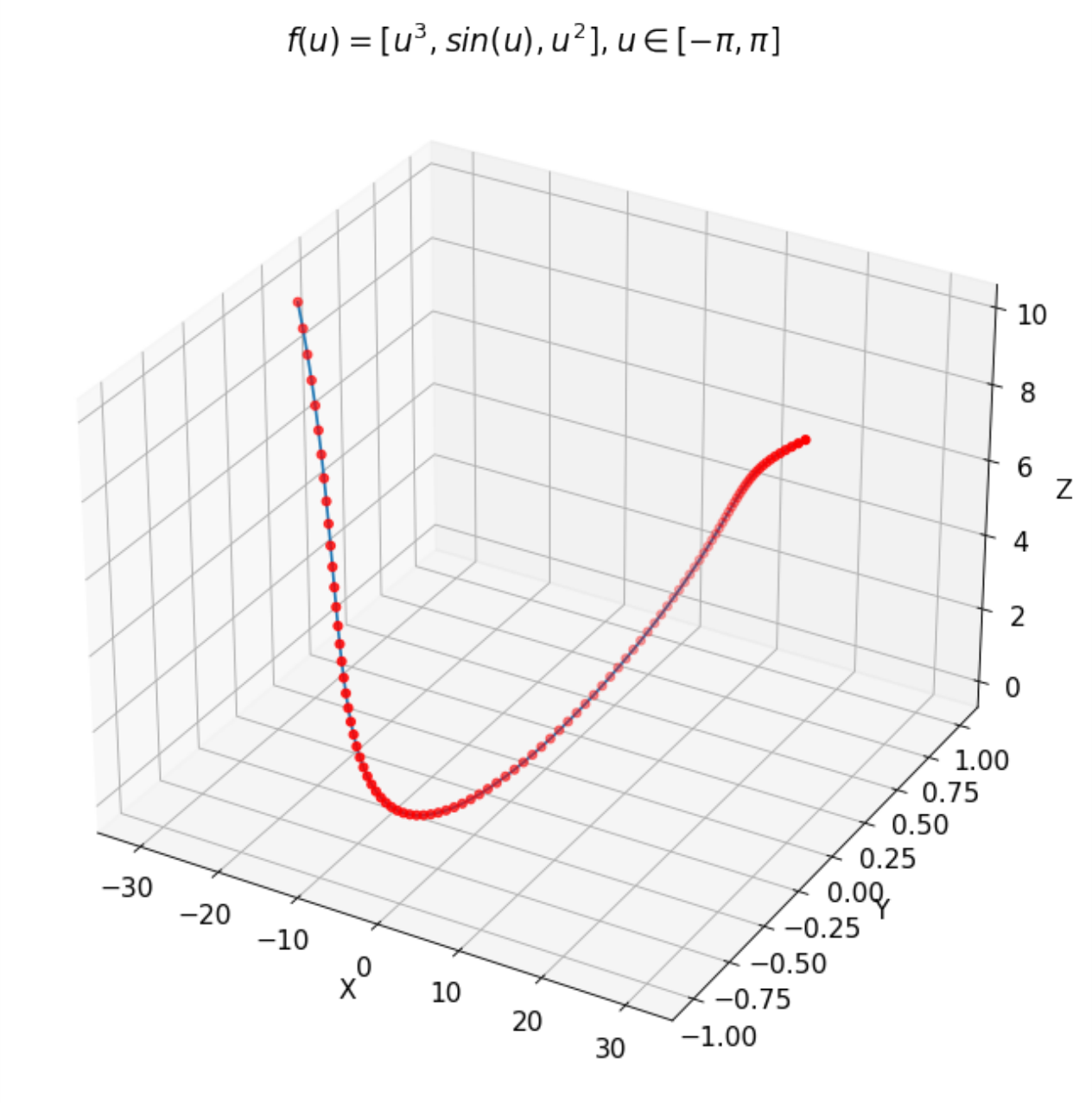
Here, we define and plot the function f - the ground truth of the experiment.

[7]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams.update({"font.size": 14})
from brainlit.algorithms.trace_analysis import spline_fxns
from scipy.interpolate import BSpline, splprep

# define the paremeter space
theta = np.linspace(-np.pi, np.pi, 100)
L = len(theta)
# define f(u)
X = theta**3
Y = np.sin(theta)
Z = theta**2
# define df(u)
dX = 3 * theta**2
dY = np.cos(theta)
dZ = 2 * theta
# define ddf(u)
ddX = 6 * theta
ddY = -np.sin(theta)
ddZ = 2 * np.ones(L)
```

(continues on next page)

(continued from previous page)

```
# define dddf(u)
dddX = 6 * np.ones(L)
dddY = -np.cos(theta)
dddZ = np.zeros(L)

# define the ground-truth arrays
C = np.array([X, Y, Z])
dC = np.array([dX, dY, dZ]).T
ddC = np.array([ddX, ddY, ddZ]).T
dddC = np.array([dddX, dddY, dddZ]).T

# plot f(u)
fig = plt.figure(figsize=(12, 10), dpi=80)
ax = fig.add_subplot(1, 1, 1, projection="3d")
ax.plot(X, Y, Z)
ax.scatter(X, Y, Z, c="r")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.set_title(r"$f(u) = [u^3, sin(u), u^2], u \in [-\pi, \pi]$")
plt.show()
```



1. Speed

The speed measures how fast a point is moving on a parametric curve.

Let $F : \mathbf{R} \to \mathbf{R}^d$ be a differentiable function, its speed is the ℓ^2-norm of $\mathbf{J}(F) = \left[\frac{\partial F_i}{\partial x}, \ldots, \frac{\partial F_d}{\partial x}\right]$.

Given $u_1, \ldots, u_N$ evaluation points of the parameter, we will compare the results of `spline_fxns.speed()` (denoted with $\hat{S}_k$) with the ground truth $S = ||\mathbf{J}(f)||_2 = \sqrt{(3u_i^2)^2 + (\cos(u_i))^2 + (2u_i)^2}$. Here, we will use the uniform norm of the relative error:

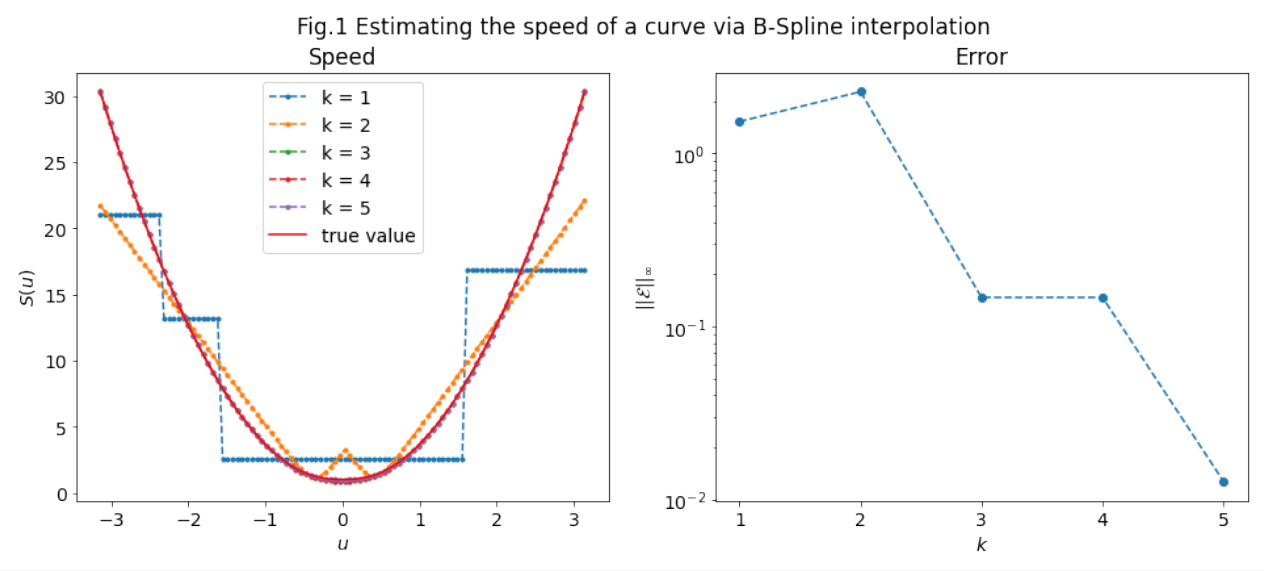
$||\mathcal{E}||_\infty = \max |\mathcal{E}|, \quad \mathcal{E} = \frac{S(u) - \hat{S}_k(u)}{S(u)},$

to evaluate the accuracy as a function of k.

$Fig.1$ shows the estimated speed and its error for $k \in \{1, 2, 3, 4, 5\}$. Specifically, we see that the default value of $k = 3$ implies a 10% error on the speed, while $k = 5$ performs better, with an error $\sim 1\%$.

[8]:
```
# prepare output figure and axes
fig = plt.figure(figsize=(16, 6))
axes = fig.subplots(1, 2)

# evaluate the theoretical expected value S(u)
expected_speed = np.linalg.norm(dC, axis=1)

# initialize vector of B-Spline orders to test
ks = [1, 2, 3, 4, 5]
# initialize vector that will contain the relative errors
uniform_err = []
for k in ks:
    tck, u = splprep(C, u=theta, k=k)
    t = tck[0]
    c = tck[1]
    k = tck[2]
    speed = spline_fxns.speed(theta, t, c, k, aux_outputs=False)
    # plot the estimated curvature
    axes[0].plot(theta, speed, "o--", label="k = %d" % k, markersize=3)
    # evaluate the uniform error
    uniform_err.append(np.amax(np.abs((expected_speed - speed) / expected_speed)))

# plot speed
ax = axes[0]
ax.plot(theta, expected_speed, c="r", label="true value")
ax.set_xlabel(r"$u$")
ax.set_ylabel(r"$S(u)$")
ax.set_title("Speed")
ax.legend()

# plot error
ax = axes[1]
ax.plot(ks, uniform_err, "o--")
ax.set_yscale("log")
ax.set_xlabel(r"$k$")
ax.set_xticks(ks)
ax.set_ylabel(r"$||\mathcal{E}||_\infty$")
ax.set_title("Error")

fig.suptitle("Fig.1 Estimating the speed of a curve via B-Spline interpolation")
```

[8]:
```
Text(0.5, 0.98, 'Fig.1 Estimating the speed of a curve via B-Spline interpolation')
```



2. Curvature

The curvature measures the failure of a curve to be a straight line.

Given $u_1, \ldots, u_N$ evaluation points of the parameter, the expected curvature vector C for the ground truth function f is

$C(u) = \|f'(u) \times f''(u)\| \; / \; \|f'(u)\|^3$.

Here, we will compare the results of `spline_fxns.curvature()` (denoted with $\hat{C}_k$) with the ground truth C. Again, we will use the uniform norm of the relative error:

$||\mathcal{E}||_\infty = \max|\mathcal{E}|, \quad \mathcal{E} = \frac{C(u) - \hat{C}_k(u)}{C(u)},$

to evaluate the accuracy as a function of k.

$Fig.2$ shows the estimated curvature and its error for $k \in \{1, 2, 3, 4, 5\}$. For $k = 1$, the curvature is identically 0 for any u because the second derivative of a B-Spline of order 1 does not exist, and we set it to 0. Specifically, we see that the default value of $k = 3$ implies a $\sim 30\%$ error on the curvature, which is much higher than the previous error found for the speed. We also see that for $k = 5$ the uniform error is $\sim 10\%$, which is almost 10 times bigger than the error on the speed for $k = 5$.

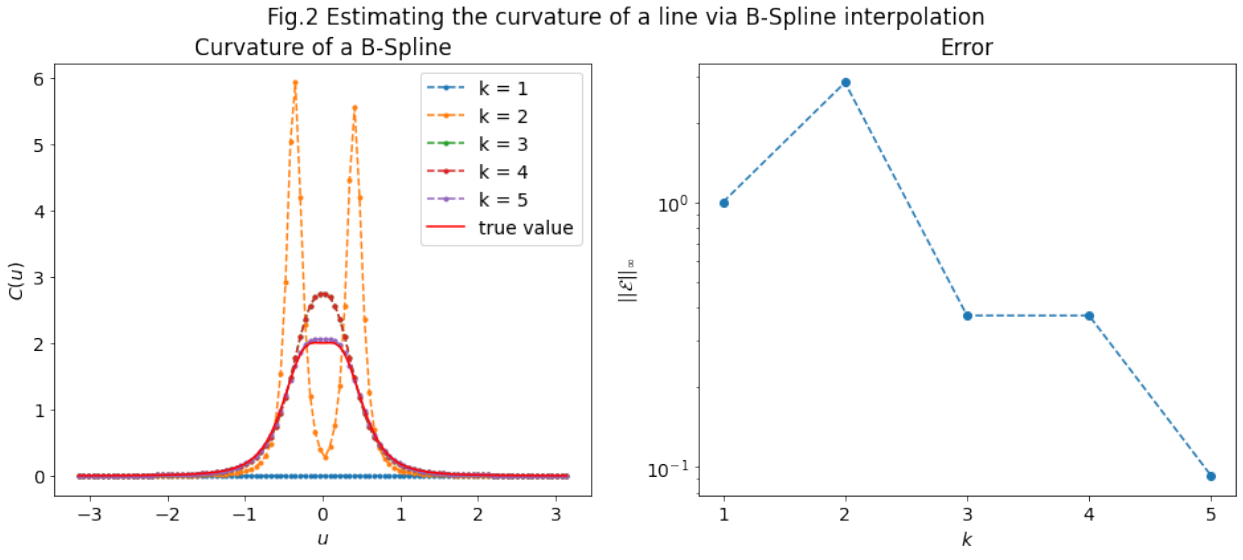
[11]:
```
# prepare output figure and axes
fig = plt.figure(figsize=(16, 6))
axes = fig.subplots(1, 2)

# evaluate the theoretical expected value C(u)
cross = np.cross(dC, ddC)
num = np.linalg.norm(cross, axis=1)
denom = np.linalg.norm(dC, axis=1) ** 3
expected_curvature = np.nan_to_num(num / denom)

# initialize vector of B-Spline orders to test
ks = [1, 2, 3, 4, 5]
# initialize vector that will contain the relative errors
uniform_err = []
for k in ks:
```

(continues on next page)

(continued from previous page)

```
    tck, u = splprep(C, u=theta, k=k)
    t = tck[0]
    c = tck[1]
    k = tck[2]
    curvature, deriv, dderiv = spline_fxns.curvature(theta, t, c, k, aux_outputs=True)
    # plot the estimated curvature
    axes[0].plot(theta, curvature, "o--", label="k = %d" % k, markersize=3)
    # evaluate the uniform error
    uniform_err.append(
        np.amax(np.abs((expected_curvature - curvature) / expected_curvature))
    )

# plot curvature
ax = axes[0]
ax.plot(theta, expected_curvature, c="r", label="true value")
ax.set_xlabel(r"$u$")
ax.set_ylabel(r"$C(u)$")
ax.set_title("Curvature of a B-Spline")
ax.legend()

# plot error
ax = axes[1]
ax.plot(ks, uniform_err, "o--")
ax.set_yscale("log")
ax.set_xlabel(r"$k$")
ax.set_xticks(ks)
ax.set_ylabel(r"$||\mathcal{E}||_\infty$")
ax.set_title("Error")

fig.suptitle("Fig.2 Estimating the curvature of a line via B-Spline interpolation")
```

[11]:
```
Text(0.5, 0.98, 'Fig.2 Estimating the curvature of a line via B-Spline interpolation')
```



3. Torsion

The torsion measures the failure of a curve to be planar.

Given $u_1, \dots, u_N$ evaluation points of the parameter, the expected torsion vector τ for the ground truth function f is

$\tau(u) = |f'(u), f''(u), f'''(u)| \ / \ \|f'(u) \times f''(u)\|^2$

Here, we will compare the results of `spline_fxns.torsion()` (denoted with $\hat{\tau_k}$) with the ground truth τ. Again, ee will use the uniform norm of the relative error:

$||\mathcal{E}||_\infty = \max |\mathcal{E}|, \quad \mathcal{E} = \frac{\tau(u) - \hat{\tau_k}(u)}{\tau(u)},$

to evaluate the accuracy as a function of k.

$Fig.3$ shows the estimated torsion and its error for $k \in \{1, 2, 3, 4, 5\}$. For $k = 1, 2$ the torsion is identically 0 for any u because the second, third derivatives of a B-Spline of order $1, 2$ respectively, cannot be evaluated, so we set them to 0. Interestingly, we see that $k = 3$ reduces the error compared to $k = 5$. This happens because the B-Spline with order 5 is worse at estimating the long tails close to 0, while it performs better with larger values of the torsion.

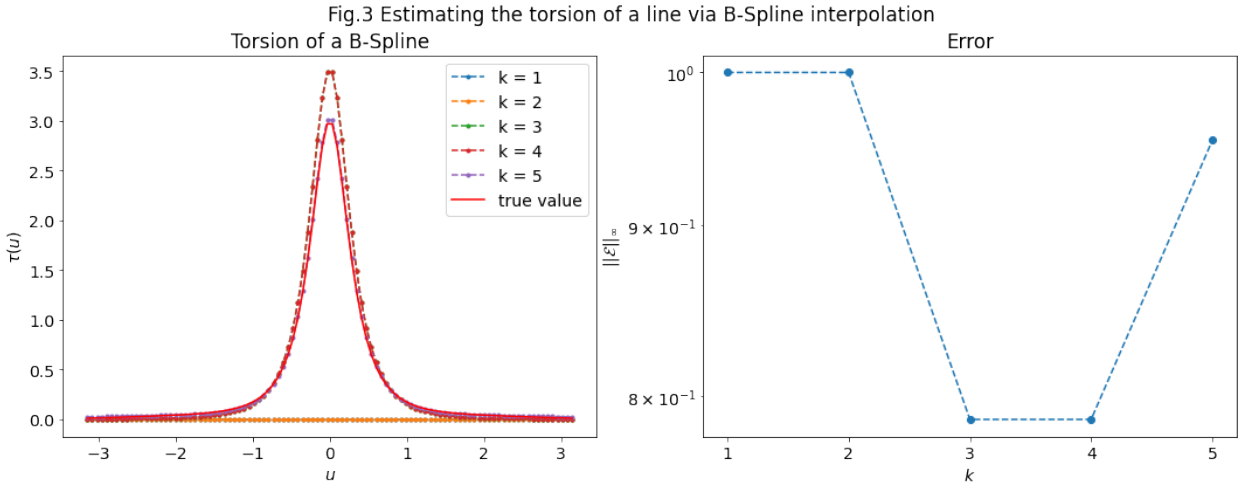
[9]:
```
# prepare output figure and axes
fig = plt.figure(figsize=(18, 6))
axes = fig.subplots(1, 2)

# evaluate the theoretical expected value \tau(u)
expected_cross = np.cross(dC, ddC)
expected_num = np.diag((expected_cross @ dddC.T))
expected_denom = np.linalg.norm(expected_cross, axis=1) ** 2
expected_torsion = np.nan_to_num(expected_num / expected_denom)

# initialize vector of B-Spline orders to test
ks = [1, 2, 3, 4, 5]
# initialize vector that will contain the relative errors
uniform_err = []
for k in [1, 2, 3, 4, 5]:
    tck, u = splprep(C, u=theta, k=k)
    t = tck[0]
    c = tck[1]
    k = tck[2]
    torsion = spline_fxns.torsion(theta, t, c, k, aux_outputs=False)
    # plot the estimated curvature
    axes[0].plot(theta, torsion, "o--", label="k = %d" % k, markersize=3)
    # evaluate the uniform error
    uniform_err.append(np.amax(np.abs((expected_torsion - torsion) / expected_
↪torsion)))

# plot torsion
ax = axes[0]
ax.plot(theta, expected_torsion, c="r", label="true value")
ax.set_xlabel(r"$u$")
ax.set_ylabel(r"$\tau(u)$")
ax.set_title("Torsion of a B-Spline")
ax.legend()

# plot error
ax = axes[1]
ax.plot(ks, uniform_err, "o--")
ax.set_yscale("log")
ax.set_xlabel(r"$k$")
```

(continues on next page)

(continued from previous page)

```
ax.set_xticks(ks)
ax.set_ylabel(r"$||\mathcal{E}||_\infty$")
ax.set_title("Error")

fig.suptitle("Fig.3 Estimating the torsion of a line via B-Spline interpolation")
```

[9]:
```
Text(0.5, 0.98, 'Fig.3 Estimating the torsion of a line via B-Spline interpolation')
```



[10]:
```
curvature, deriv, dderiv = spline_fxns.curvature(theta, t, c, k, aux_outputs=True)
print(deriv, dC)
```

```
[[ 2.96088132e+01 -8.71069066e-01 -6.28318531e+00]
 [ 2.84245815e+01 -9.01479929e-01 -6.15625227e+00]
 [ 2.72645178e+01 -9.22808496e-01 -6.02931923e+00]
 [ 2.61286221e+01 -9.35578172e-01 -5.90238620e+00]
 [ 2.50168944e+01 -9.40301452e-01 -5.77545316e+00]
 [ 2.39293347e+01 -9.37479932e-01 -5.64852012e+00]
 [ 2.28659430e+01 -9.27604299e-01 -5.52158709e+00]
 [ 2.18267193e+01 -9.11154340e-01 -5.39465405e+00]
 [ 2.08116635e+01 -8.88598935e-01 -5.26772102e+00]
 [ 1.98207758e+01 -8.60396060e-01 -5.14078798e+00]
 [ 1.88540560e+01 -8.26992789e-01 -5.01385494e+00]
 [ 1.79115043e+01 -7.88825289e-01 -4.88692191e+00]
 [ 1.69931205e+01 -7.46318825e-01 -4.75998887e+00]
 [ 1.60989048e+01 -6.99887756e-01 -4.63305583e+00]
 [ 1.52288570e+01 -6.49935539e-01 -4.50612280e+00]
 [ 1.43829772e+01 -5.96854724e-01 -4.37918976e+00]
 [ 1.35612654e+01 -5.41026959e-01 -4.25225672e+00]
 [ 1.27637216e+01 -4.82822986e-01 -4.12532369e+00]
 [ 1.19903458e+01 -4.22602646e-01 -3.99839065e+00]
 [ 1.12411380e+01 -3.60714871e-01 -3.87145761e+00]
 [ 1.05160982e+01 -2.97497694e-01 -3.74452458e+00]
 [ 9.81522642e+00 -2.33278239e-01 -3.61759154e+00]
 [ 9.13852259e+00 -1.68372729e-01 -3.49065850e+00]
 [ 8.48598677e+00 -1.03086482e-01 -3.36372547e+00]
 [ 7.85761893e+00 -3.77139112e-02 -3.23679243e+00]
 [ 7.25341909e+00  2.74614739e-02 -3.10985939e+00]
 [ 6.67338724e+00  9.21670682e-02 -2.98292636e+00]
 [ 6.11752339e+00  1.56141171e-01 -2.85599332e+00]
 [ 5.58582753e+00  2.19132984e-01 -2.72906028e+00]
```

(continues on next page)

(continued from previous page)

```
[ 5.07829966e+00  2.80902617e-01 -2.60212725e+00]
[ 4.59493979e+00  3.41221081e-01 -2.47519421e+00]
[ 4.13574791e+00  3.99870290e-01 -2.34826118e+00]
[ 3.70072403e+00  4.56643066e-01 -2.22132814e+00]
[ 3.28986813e+00  5.11343133e-01 -2.09439510e+00]
[ 2.90318024e+00  5.63785119e-01 -1.96746207e+00]
[ 2.54066033e+00  6.13794557e-01 -1.84052903e+00]
[ 2.20230842e+00  6.61207884e-01 -1.71359599e+00]
[ 1.88812450e+00  7.05872441e-01 -1.58666296e+00]
[ 1.59810858e+00  7.47646473e-01 -1.45972992e+00]
[ 1.33226065e+00  7.86399129e-01 -1.33279688e+00]
[ 1.09058071e+00  8.22010464e-01 -1.20586385e+00]
[ 8.73068770e-01  8.54371436e-01 -1.07893081e+00]
[ 6.79724821e-01  8.83383906e-01 -9.51997774e-01]
[ 5.10548866e-01  9.08960641e-01 -8.25064737e-01]
[ 3.65540904e-01  9.31025311e-01 -6.98131701e-01]
[ 2.44700936e-01  9.49512491e-01 -5.71198664e-01]
[ 1.48028961e-01  9.64367661e-01 -4.44265628e-01]
[ 7.55249801e-02  9.75547203e-01 -3.17332591e-01]
[ 2.71889928e-02  9.83018405e-01 -1.90399555e-01]
[ 3.02099920e-03  9.86759458e-01 -6.34665183e-02]
[ 3.02099920e-03  9.86759458e-01  6.34665183e-02]
[ 2.71889928e-02  9.83018405e-01  1.90399555e-01]
[ 7.55249801e-02  9.75547203e-01  3.17332591e-01]
[ 1.48028961e-01  9.64367661e-01  4.44265628e-01]
[ 2.44700936e-01  9.49512491e-01  5.71198664e-01]
[ 3.65540904e-01  9.31025311e-01  6.98131701e-01]
[ 5.10548866e-01  9.08960641e-01  8.25064737e-01]
[ 6.79724821e-01  8.83383906e-01  9.51997774e-01]
[ 8.73068770e-01  8.54371436e-01  1.07893081e+00]
[ 1.09058071e+00  8.22010464e-01  1.20586385e+00]
[ 1.33226065e+00  7.86399129e-01  1.33279688e+00]
[ 1.59810858e+00  7.47646473e-01  1.45972992e+00]
[ 1.88812450e+00  7.05872441e-01  1.58666296e+00]
[ 2.20230842e+00  6.61207884e-01  1.71359599e+00]
[ 2.54066033e+00  6.13794557e-01  1.84052903e+00]
[ 2.90318024e+00  5.63785119e-01  1.96746207e+00]
[ 3.28986813e+00  5.11343133e-01  2.09439510e+00]
[ 3.70072403e+00  4.56643066e-01  2.22132814e+00]
[ 4.13574791e+00  3.99870290e-01  2.34826118e+00]
[ 4.59493979e+00  3.41221081e-01  2.47519421e+00]
[ 5.07829966e+00  2.80902617e-01  2.60212725e+00]
[ 5.58582753e+00  2.19132984e-01  2.72906028e+00]
[ 6.11752339e+00  1.56141171e-01  2.85599332e+00]
[ 6.67338724e+00  9.21670682e-02  2.98292636e+00]
[ 7.25341909e+00  2.74614739e-02  3.10985939e+00]
[ 7.85761893e+00 -3.77139112e-02  3.23679243e+00]
[ 8.48598677e+00 -1.03086482e-01  3.36372547e+00]
[ 9.13852259e+00 -1.68372729e-01  3.49065850e+00]
[ 9.81522642e+00 -2.33278239e-01  3.61759154e+00]
[ 1.05160982e+01 -2.97497694e-01  3.74452458e+00]
[ 1.12411380e+01 -3.60714871e-01  3.87145761e+00]
[ 1.19903458e+01 -4.22602646e-01  3.99839065e+00]
[ 1.27637216e+01 -4.82822986e-01  4.12532369e+00]
[ 1.35612654e+01 -5.41026959e-01  4.25225672e+00]
[ 1.43829772e+01 -5.96854724e-01  4.37918976e+00]
[ 1.52288570e+01 -6.49935539e-01  4.50612280e+00]
```

(continues on next page)

(continued from previous page)

```
 [ 1.60989048e+01 -6.99887756e-01  4.63305583e+00]
 [ 1.69931205e+01 -7.46318825e-01  4.75998887e+00]
 [ 1.79115043e+01 -7.88825289e-01  4.88692191e+00]
 [ 1.88540560e+01 -8.26992789e-01  5.01385494e+00]
 [ 1.98207758e+01 -8.60396060e-01  5.14078798e+00]
 [ 2.08116635e+01 -8.88598935e-01  5.26772102e+00]
 [ 2.18267193e+01 -9.11154340e-01  5.39465405e+00]
 [ 2.28659430e+01 -9.27604299e-01  5.52158709e+00]
 [ 2.39293347e+01 -9.37479932e-01  5.64852012e+00]
 [ 2.50168944e+01 -9.40301452e-01  5.77545316e+00]
 [ 2.61286221e+01 -9.35578172e-01  5.90238620e+00]
 [ 2.72645178e+01 -9.22808496e-01  6.02931923e+00]
 [ 2.84245815e+01 -9.01479929e-01  6.15625227e+00]
 [ 2.96088132e+01 -8.71069066e-01  6.28318531e+00]] [[ 2.96088132e+01 -1.00000000e+00␣
↪-6.28318531e+00]
 [ 2.84245815e+01 -9.97986676e-01 -6.15625227e+00]
 [ 2.72645178e+01 -9.91954813e-01 -6.02931923e+00]
 [ 2.61286221e+01 -9.81928697e-01 -5.90238620e+00]
 [ 2.50168944e+01 -9.67948701e-01 -5.77545316e+00]
 [ 2.39293347e+01 -9.50071118e-01 -5.64852012e+00]
 [ 2.28659430e+01 -9.28367933e-01 -5.52158709e+00]
 [ 2.18267193e+01 -9.02926538e-01 -5.39465405e+00]
 [ 2.08116635e+01 -8.73849377e-01 -5.26772102e+00]
 [ 1.98207758e+01 -8.41253533e-01 -5.14078798e+00]
 [ 1.88540560e+01 -8.05270258e-01 -5.01385494e+00]
 [ 1.79115043e+01 -7.66044443e-01 -4.88692191e+00]
 [ 1.69931205e+01 -7.23734038e-01 -4.75998887e+00]
 [ 1.60989048e+01 -6.78509412e-01 -4.63305583e+00]
 [ 1.52288570e+01 -6.30552667e-01 -4.50612280e+00]
 [ 1.43829772e+01 -5.80056910e-01 -4.37918976e+00]
 [ 1.35612654e+01 -5.27225468e-01 -4.25225672e+00]
 [ 1.27637216e+01 -4.72271075e-01 -4.12532369e+00]
 [ 1.19903458e+01 -4.15415013e-01 -3.99839065e+00]
 [ 1.12411380e+01 -3.56886222e-01 -3.87145761e+00]
 [ 1.05160982e+01 -2.96920375e-01 -3.74452458e+00]
 [ 9.81522642e+00 -2.35758936e-01 -3.61759154e+00]
 [ 9.13852259e+00 -1.73648178e-01 -3.49065850e+00]
 [ 8.48598677e+00 -1.10838200e-01 -3.36372547e+00]
 [ 7.85761893e+00 -4.75819158e-02 -3.23679243e+00]
 [ 7.25341909e+00  1.58659638e-02 -3.10985939e+00]
 [ 6.67338724e+00  7.92499569e-02 -2.98292636e+00]
 [ 6.11752339e+00  1.42314838e-01 -2.85599332e+00]
 [ 5.58582753e+00  2.04806668e-01 -2.72906028e+00]
 [ 5.07829966e+00  2.66473814e-01 -2.60212725e+00]
 [ 4.59493979e+00  3.27067963e-01 -2.47519421e+00]
 [ 4.13574791e+00  3.86345126e-01 -2.34826118e+00]
 [ 3.70072403e+00  4.44066613e-01 -2.22132814e+00]
 [ 3.28986813e+00  5.00000000e-01 -2.09439510e+00]
 [ 2.90318024e+00  5.53920064e-01 -1.96746207e+00]
 [ 2.54066033e+00  6.05609687e-01 -1.84052903e+00]
 [ 2.20230842e+00  6.54860734e-01 -1.71359599e+00]
 [ 1.88812450e+00  7.01474888e-01 -1.58666296e+00]
 [ 1.59810858e+00  7.45264450e-01 -1.45972992e+00]
 [ 1.33226065e+00  7.86053095e-01 -1.33279688e+00]
 [ 1.09058071e+00  8.23676581e-01 -1.20586385e+00]
 [ 8.73068770e-01  8.57983413e-01 -1.07893081e+00]
 [ 6.79724821e-01  8.88835449e-01 -9.51997774e-01]
```

(continues on next page)

(continued from previous page)

```
[ 5.10548866e-01  9.16108457e-01 -8.25064737e-01]
[ 3.65540904e-01  9.39692621e-01 -6.98131701e-01]
[ 2.44700936e-01  9.59492974e-01 -5.71198664e-01]
[ 1.48028961e-01  9.75429787e-01 -4.44265628e-01]
[ 7.55249801e-02  9.87438889e-01 -3.17332591e-01]
[ 2.71889928e-02  9.95471923e-01 -1.90399555e-01]
[ 3.02099920e-03  9.99496542e-01 -6.34665183e-02]
[ 3.02099920e-03  9.99496542e-01  6.34665183e-02]
[ 2.71889928e-02  9.95471923e-01  1.90399555e-01]
[ 7.55249801e-02  9.87438889e-01  3.17332591e-01]
[ 1.48028961e-01  9.75429787e-01  4.44265628e-01]
[ 2.44700936e-01  9.59492974e-01  5.71198664e-01]
[ 3.65540904e-01  9.39692621e-01  6.98131701e-01]
[ 5.10548866e-01  9.16108457e-01  8.25064737e-01]
[ 6.79724821e-01  8.88835449e-01  9.51997774e-01]
[ 8.73068770e-01  8.57983413e-01  1.07893081e+00]
[ 1.09058071e+00  8.23676581e-01  1.20586385e+00]
[ 1.33226065e+00  7.86053095e-01  1.33279688e+00]
[ 1.59810858e+00  7.45264450e-01  1.45972992e+00]
[ 1.88812450e+00  7.01474888e-01  1.58666296e+00]
[ 2.20230842e+00  6.54860734e-01  1.71359599e+00]
[ 2.54066033e+00  6.05609687e-01  1.84052903e+00]
[ 2.90318024e+00  5.53920064e-01  1.96746207e+00]
[ 3.28986813e+00  5.00000000e-01  2.09439510e+00]
[ 3.70072403e+00  4.44066613e-01  2.22132814e+00]
[ 4.13574791e+00  3.86345126e-01  2.34826118e+00]
[ 4.59493979e+00  3.27067963e-01  2.47519421e+00]
[ 5.07829966e+00  2.66473814e-01  2.60212725e+00]
[ 5.58582753e+00  2.04806668e-01  2.72906028e+00]
[ 6.11752339e+00  1.42314838e-01  2.85599332e+00]
[ 6.67338724e+00  7.92499569e-02  2.98292636e+00]
[ 7.25341909e+00  1.58659638e-02  3.10985939e+00]
[ 7.85761893e+00 -4.75819158e-02  3.23679243e+00]
[ 8.48598677e+00 -1.10838200e-01  3.36372547e+00]
[ 9.13852259e+00 -1.73648178e-01  3.49065850e+00]
[ 9.81522642e+00 -2.35758936e-01  3.61759154e+00]
[ 1.05160982e+01 -2.96920375e-01  3.74452458e+00]
[ 1.12411380e+01 -3.56886222e-01  3.87145761e+00]
[ 1.19903458e+01 -4.15415013e-01  3.99839065e+00]
[ 1.27637216e+01 -4.72271075e-01  4.12532369e+00]
[ 1.35612654e+01 -5.27225468e-01  4.25225672e+00]
[ 1.43829772e+01 -5.80056910e-01  4.37918976e+00]
[ 1.52288570e+01 -6.30552667e-01  4.50612280e+00]
[ 1.60989048e+01 -6.78509412e-01  4.63305583e+00]
[ 1.69931205e+01 -7.23734038e-01  4.75998887e+00]
[ 1.79115043e+01 -7.66044443e-01  4.88692191e+00]
[ 1.88540560e+01 -8.05270258e-01  5.01385494e+00]
[ 1.98207758e+01 -8.41253533e-01  5.14078798e+00]
[ 2.08116635e+01 -8.73849377e-01  5.26772102e+00]
[ 2.18267193e+01 -9.02926538e-01  5.39465405e+00]
[ 2.28659430e+01 -9.28367933e-01  5.52158709e+00]
[ 2.39293347e+01 -9.50071118e-01  5.64852012e+00]
[ 2.50168944e+01 -9.67948701e-01  5.77545316e+00]
[ 2.61286221e+01 -9.81928697e-01  5.90238620e+00]
[ 2.72645178e+01 -9.91954813e-01  6.02931923e+00]
[ 2.84245815e+01 -9.97986676e-01  6.15625227e+00]
[ 2.96088132e+01 -1.00000000e+00  6.28318531e+00]]
```

[ ]:

[4]:
```
from pathlib import Path
from brainlit.utils.Neuron_trace import NeuronTrace
from brainlit.algorithms.trace_analysis.fit_spline import GeometricGraph
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.cm import ScalarMappable
from mpl_toolkits.mplot3d import Axes3D
import matplotlib
from scipy.interpolate import splev
```

Fitting Splines to Neuron Trace SWC Tutorial

1) Define variables

- `swc` the geometric graph
- `df, _, _, _` read the x, y, and z columns in swc file
- `neuron` define a new class inherited from `GeometricGraph` class
- `soma` define the data on the first run as the location of soma

[6]:
```
brainlit_path = Path.cwd().parent.parent.parent
swc = Path.joinpath(
    brainlit_path,
    "data",
    "data_octree",
    "consensus-swcs",
    "2018-08-01_G-002_consensus.swc",
)
nt = NeuronTrace(path=str(swc))
df = nt.get_df()
neuron = GeometricGraph(df=df)
soma = np.array([df.x[0], df.y[0], df.z[0]])
```

2) Plot the whole spline tree

- `spline_tree` use the `fit_spline_tree_invariant` to locate neuron branches

[8]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
spline_tree = neuron.fit_spline_tree_invariant()
for node in spline_tree.nodes:
    path = spline_tree.nodes[node]["path"]
    locs = np.zeros((len(path), 3))
    for p, point in enumerate(path):
        locs[p, :] = neuron.nodes[point]["loc"]
    ax.scatter(
        locs[:, 0],
        locs[:, 1],
        locs[:, 2],
        marker=".",
```

(continues on next page)

(continued from previous page)

```
        edgecolor="yellowgreen",
        linewidths=1,
        c="mediumblue",
        s=8,
    )
    ax.plot(
        locs[:, 0],
        locs[:, 1],
        locs[:, 2],
        linestyle="-",
        color="midnightblue",
        linewidth=0.8,
    )
ax.scatter(soma[0], soma[1], soma[2], c="darkorange", s=5)
ax.w_xaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
ax.w_yaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
ax.w_zaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
ax.grid(True)


plt.show()
```

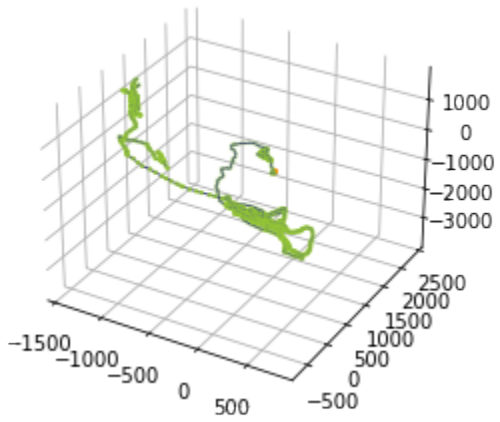


Figure 1. The green dots indicate the locations of nodes on a tree, representing a neuron, and the single orange dot locates the position of the soma. Each nodes are connected with darkblue lines to illustrate the path of the neuron.

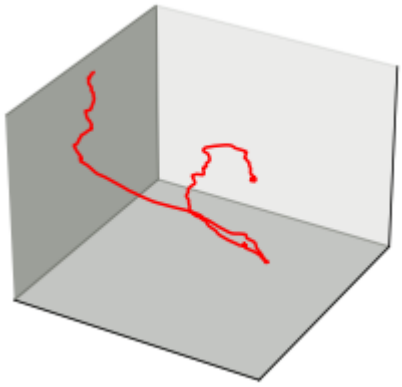
3) Plot each branch in separate plots

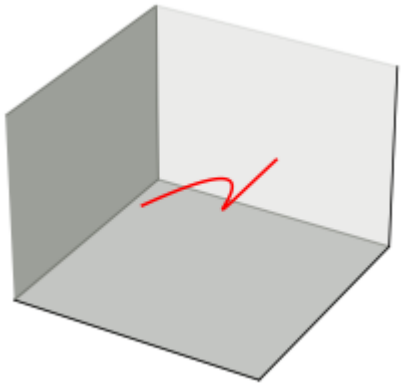
[9]:
```
for node in spline_tree.nodes:
    path = spline_tree.nodes[node]["path"]
    locs = np.zeros((len(path), 3))
    for p, point in enumerate(path):
        locs[p, :] = neuron.nodes[point]["loc"]

    spline = spline_tree.nodes[node]["spline"]
    u = spline[1]
    u = np.arange(u[0], u[-1] + 0.9, 1)
```

(continues on next page)

(continued from previous page)

```
    tck = spline[0]
    pts = splev(u, tck)

    if node < 3:
        fig = plt.figure()
        ax = fig.add_subplot(111, projection="3d")
        ax.plot(pts[0], pts[1], pts[2], "red")
        ax.w_xaxis.set_pane_color((0.23, 0.25, 0.209, 0.5))
        ax.w_yaxis.set_pane_color((0.23, 0.25, 0.209, 0.1))
        ax.w_zaxis.set_pane_color((0.23, 0.25, 0.209, 0.3))
        ax.grid(False)
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_zticks([])
        plt.axis("on")
        plt.show()
```





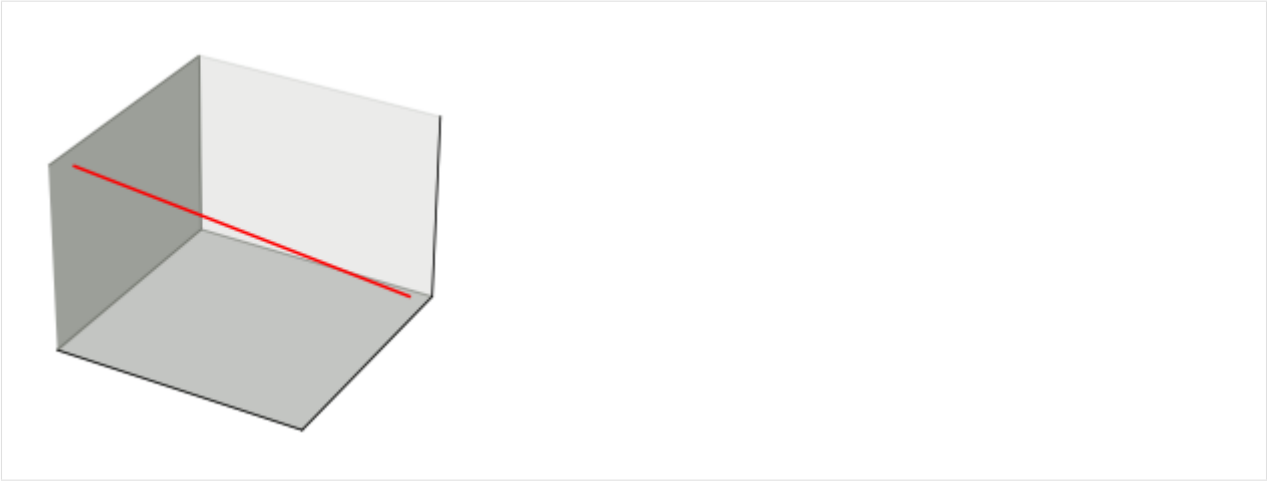


Figure 2. Examples of fitted splines to three different paths in a tree-like neuron.

BICCN PI Meeting Demo

Estimating Neuron Curvature/Torsion Demo

This notebook demonstrates fitting splines, and computing curvature/torsion from a sample neuron trace in SWC format

[17]:
```
from pathlib import Path
from brainlit.utils.Neuron_trace import NeuronTrace
from brainlit.algorithms.trace_analysis.fit_spline import GeometricGraph
from brainlit.algorithms.trace_analysis import spline_fxns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from scipy.interpolate import splev

# %matplotlib tk
```

Supplemental function for visualization later

[2]:
```
def hist_equalize(array, bins):
    ra_histogram, bins = np.histogram(array, bins, density=True)
    cdf = ra_histogram.cumsum()  # cumulative distribution function
    cdf = cdf / cdf[-1]
    # use linear interpolation of cdf to find new pixel values
    ra_equalized = np.interp(array, bins[:-1], cdf)

    return ra_equalized, cdf
```

Read SWC and fit splines

[3]:
```
brainlit_path = Path.cwd().parent.parent.parent
swc_path = Path.joinpath(
    brainlit_path,
    "data",
    "data_octree",
    "consensus-swcs",
    "2018-08-01_G-002_consensus.swc",
)

nt = NeuronTrace(path=str(swc_path))
g = nt.get_graph()
df = nt.get_df()
neuron = GeometricGraph(df=df)
spline_tree = neuron.fit_spline_tree_invariant()
soma = np.array([df.x[0], df.y[0], df.z[0]])
```

View Splines

[20]:
```
def node_height(G, node):
    predecessors = list(G.predecessors(node))
    L = len(predecessors)
    assert L == 1 or L == 0
    if L == 0:
        return 0
    else:
        return 1 + node_height(G, predecessors[0])


fig = plt.figure(figsize=(12, 10), dpi=80)
ax = Axes3D(fig)
for edge in g.edges:
    n1 = g.nodes[edge[0]]
    n2 = g.nodes[edge[1]]
    ax.plot(
        [n1["x"], n2["x"]], [n1["y"], n2["y"]], [n1["z"], n2["z"]], "b", linewidth=0.5
    )


ax.set_axis_off()
ax.set_title("Piecewise Linear")
ax.view_init(-140, 60)

fig = plt.figure(figsize=(12, 10), dpi=80)
ax = Axes3D(fig)

for j, node in enumerate(spline_tree.nodes):
    spline = spline_tree.nodes[node]
    spline_height = node_height(spline_tree, node)
    tck, u_um = spline["spline"]
    y = splev(np.arange(u_um[0], u_um[-1], 0.1), tck)

    if spline_height == 0:
        c = "b"
```

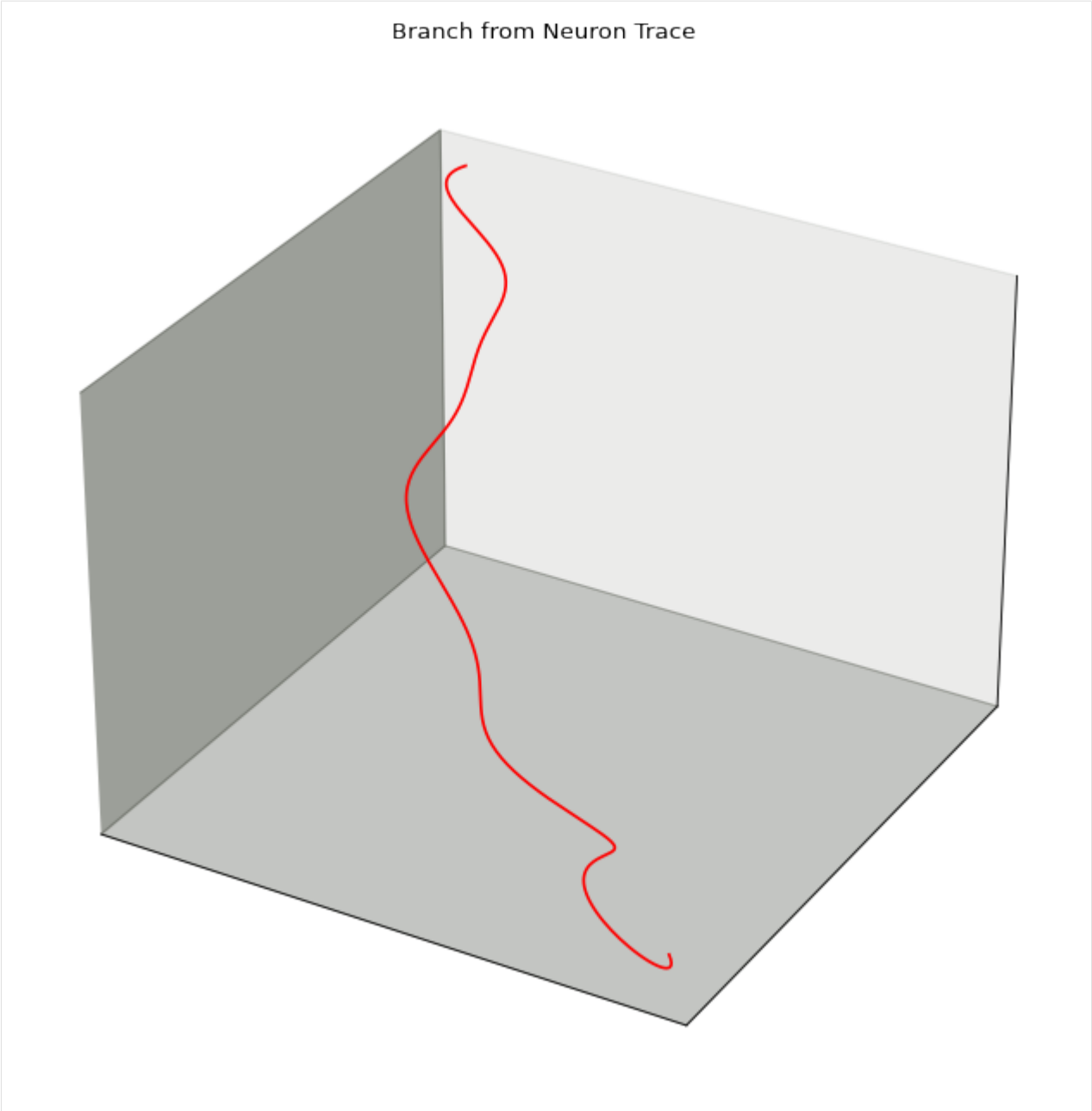
(continues on next page)

(continued from previous page)

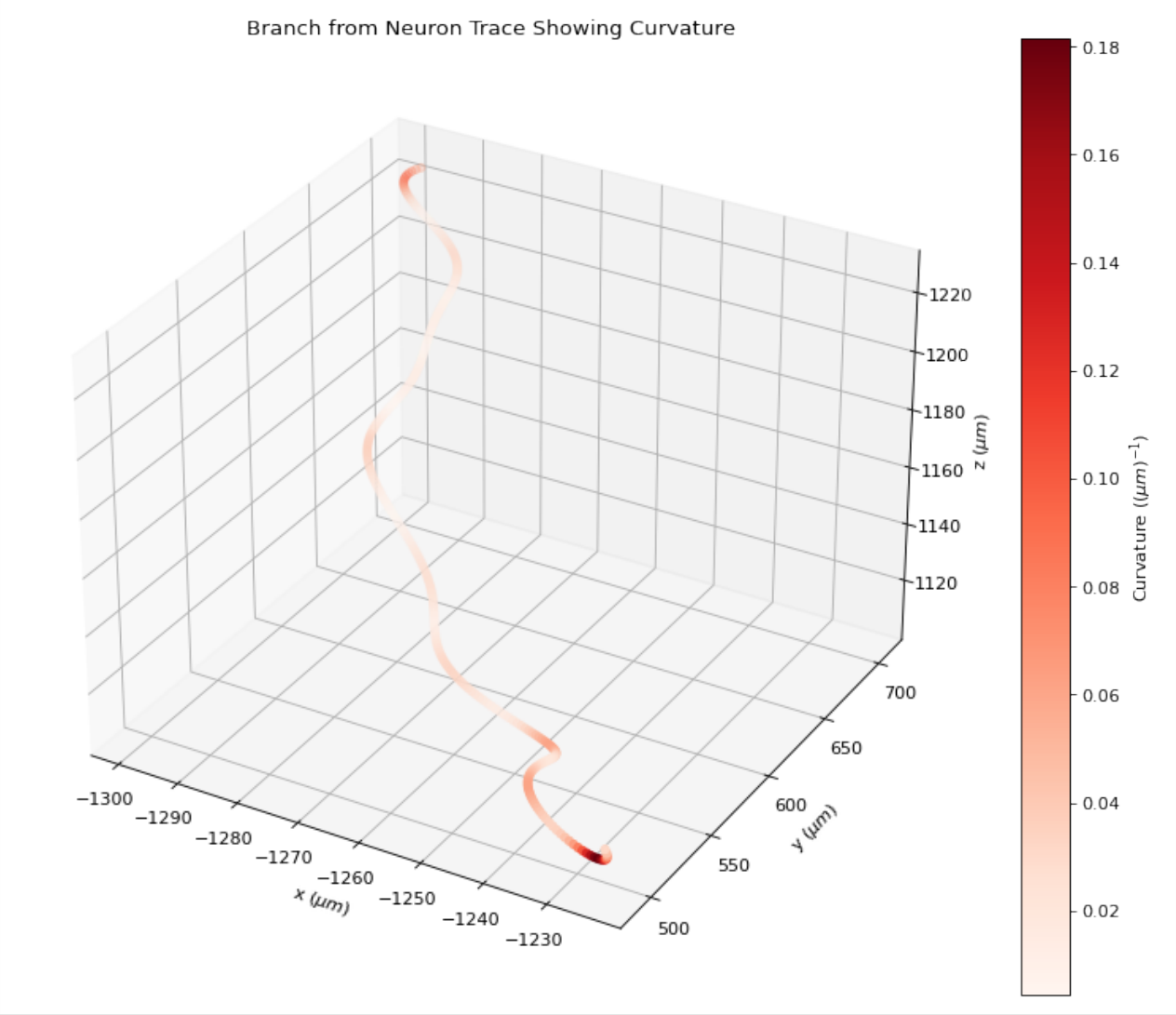
```
        ax.scatter(y[0][0], y[1][0], y[2][0], "b", s=100)
    else:
        successors = spline_tree.successors(node)
        if len(list(successors)) == 0:
            c = "g"
        else:
            c = "r"

    ax.plot(y[0], y[1], y[2], c, linewidth=0.5)

ax.set_axis_off()
ax.set_title("Splines")
ax.view_init(-140, 60)
```

View one of the branches

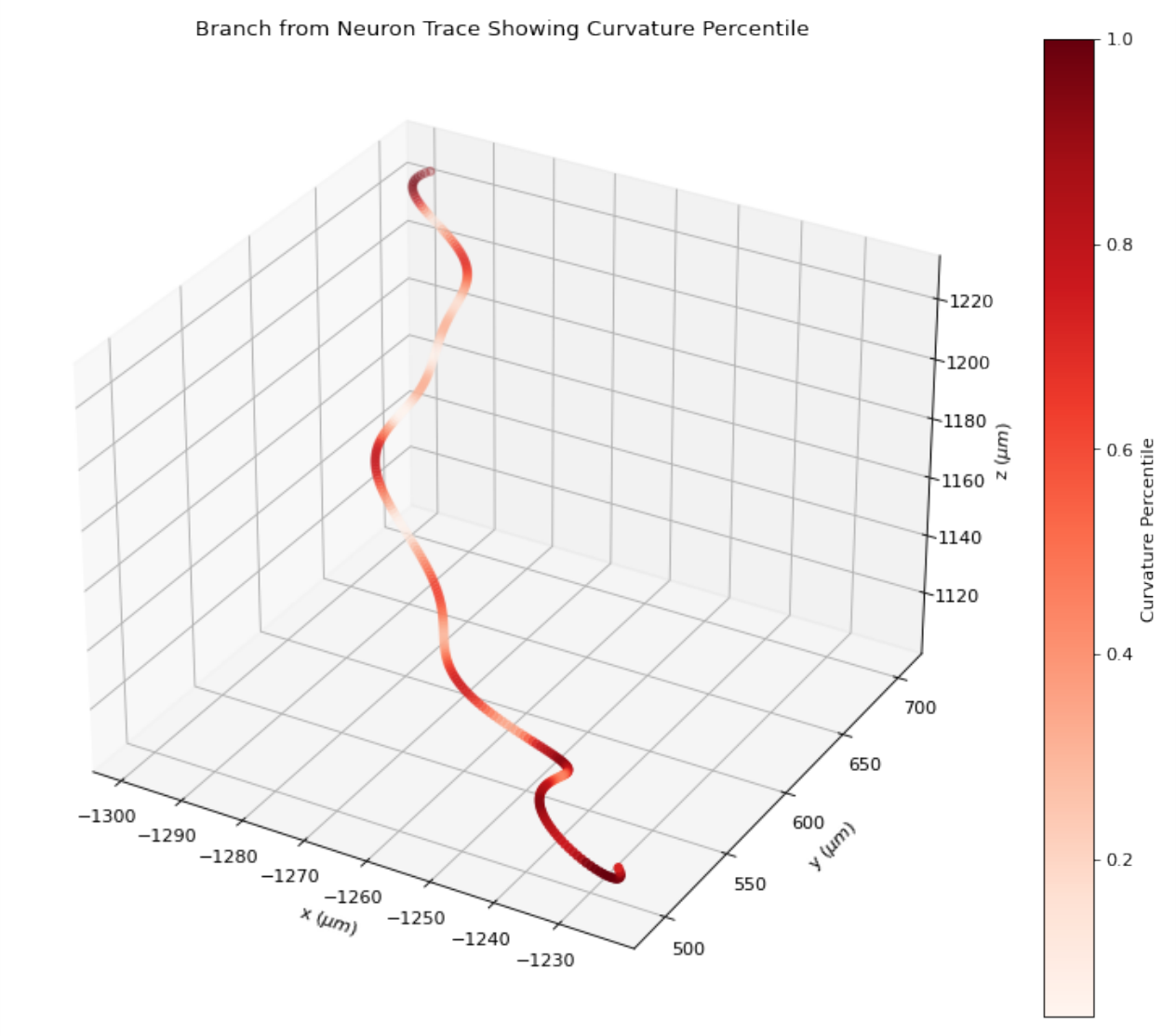
[33]:
```
node = 7

path = spline_tree.nodes[node]["path"]
locs = np.zeros((len(path), 3))
for p, point in enumerate(path):
    locs[p, :] = neuron.nodes[point]["loc"]

spline = spline_tree.nodes[node]["spline"]
u = spline[1]
u = np.arange(u[0], u[-1] + 0.9, 1)
tck = spline[0]
pts = splev(u, tck)

fig = plt.figure(figsize=(12, 10), dpi=80)
ax = fig.add_subplot(111, projection="3d")
ax.plot(pts[0], pts[1], pts[2], "red")
ax.w_xaxis.set_pane_color((0.23, 0.25, 0.209, 0.5))
ax.w_yaxis.set_pane_color((0.23, 0.25, 0.209, 0.1))
ax.w_zaxis.set_pane_color((0.23, 0.25, 0.209, 0.3))
ax.grid(False)
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])
ax.set_title("Branch from Neuron Trace")
plt.axis("on")
plt.show()
```



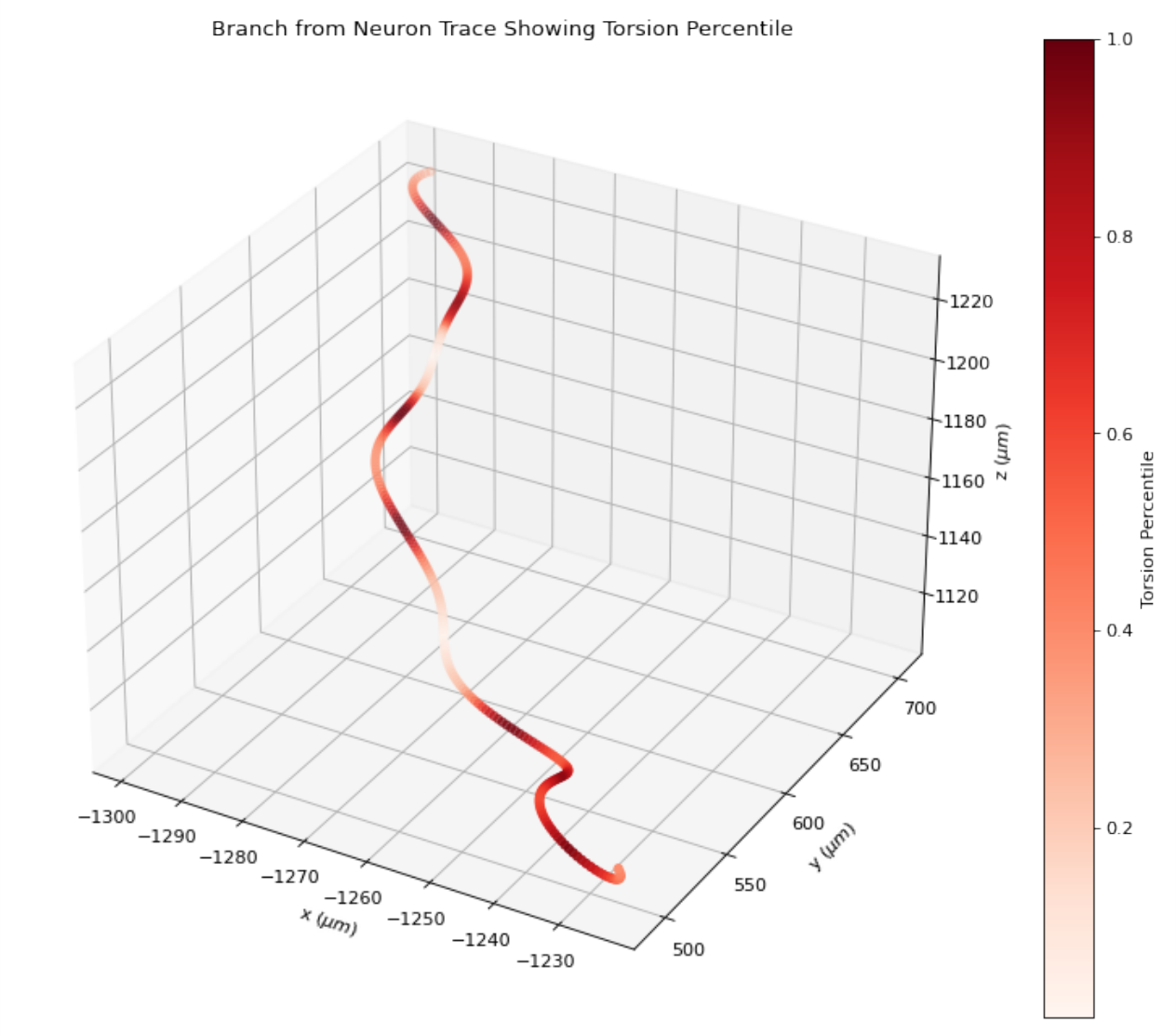
[34]:
```
curvature = spline_fxns.curvature(u, tck[0], tck[1], tck[2])
ra_eq, cdf = hist_equalize(curvature, 100)

fig = plt.figure(figsize=(12, 10), dpi=80)
ax = fig.add_subplot(111, projection="3d")
im = ax.scatter(pts[0], pts[1], pts[2], c=curvature, cmap="Reds")
ax.set_xlabel("x ($\mu m$)")
ax.set_ylabel("y ($\mu m$)")
ax.set_zlabel("z ($\mu m$)")
ax.set_title("Branch from Neuron Trace Showing Curvature")
cbar = fig.colorbar(im)
cbar.set_label("Curvature ($(\mu m)^{-1}$)")
```



[35]:
```
ra_eq, cdf = hist_equalize(curvature, 100)

fig = plt.figure(figsize=(12, 10), dpi=80)
ax = fig.add_subplot(111, projection="3d")
im = ax.scatter(pts[0], pts[1], pts[2], c=ra_eq, cmap="Reds")
ax.set_xlabel("x ($\mu m$)")
ax.set_ylabel("y ($\mu m$)")
ax.set_zlabel("z ($\mu m$)")
ax.set_title("Branch from Neuron Trace Showing Curvature Percentile")
cbar = fig.colorbar(im)
cbar.set_label("Curvature Percentile")
```



[36]:
```
torsion = spline_fxns.torsion(u, tck[0], tck[1], tck[2])
ra_eq, cdf = hist_equalize(torsion, 100)

fig = plt.figure(figsize=(12, 10), dpi=80)
ax = fig.add_subplot(111, projection="3d")
im = ax.scatter(pts[0], pts[1], pts[2], c=ra_eq, cmap="Reds")
ax.set_xlabel("x ($\mu m$)")
ax.set_ylabel("y ($\mu m$)")
ax.set_zlabel("z ($\mu m$)")
ax.set_title("Branch from Neuron Trace Showing Torsion Percentile")
cbar = fig.colorbar(im)
cbar.set_label("Torsion Percentile")
```



[ ]:

Soma Detection

Demonstrate simple soma detection algorithm on known somas in Janelia dataset, brain1.

Soma detection notebook

This notebook demonstrates how to use `brainlit.algorithms.detect_somas.find_somas` to detect bright somas in brain volumes of $100\mu m^3$.

[1]:
```
import boto3
import numpy as np
from io import BytesIO
from cloudvolume.lib import Bbox
from brainlit.utils.session import NeuroglancerSession
from brainlit.algorithms.detect_somas import find_somas
import matplotlib.pyplot as plt

brain = 1
mip = 1
```

```
/Users/jacopo/opt/miniconda3/envs/brainlit/lib/python3.8/site-packages/python_
↪jsonschema_objects/__init__.py:50: UserWarning: Schema version http://json-schema.
↪org/draft-04/schema not recognized. Some keywords and features may not be supported.
  warnings.warn(
WARNING: Could not load OpenGL library.
```

[2]:
```
s3 = boto3.resource("s3")
bucket = s3.Bucket("open-neurodata")

brain_name = f"brain{brain}"

brain_prefix = f"brainlit/{brain_name}"
segments_prefix = f"brainlit/{brain_name}_segments"
somas_prefix = f"brainlit/{brain_name}_somas"

brain_url = f"s3://open-neurodata/{brain_prefix}"
segments_url = f"s3://open-neurodata/{segments_prefix}"

volume_keys = [
    "4807349.0_3827990.0_2922565.75_4907349.0_3927990.0_3022565.75",
    "4873075.5_4753413.5_6851474.0_4973075.5_4853413.5_6951474.0",
    "4881146.0_4792378.5_7001708.5_4981146.0_4892378.5_7101708.5",
]

ngl_sess = NeuroglancerSession(
    mip=mip, url=brain_url, url_segments=segments_url, use_https=True
)
res = ngl_sess.cv_segments.scales[ngl_sess.mip]["resolution"]
```

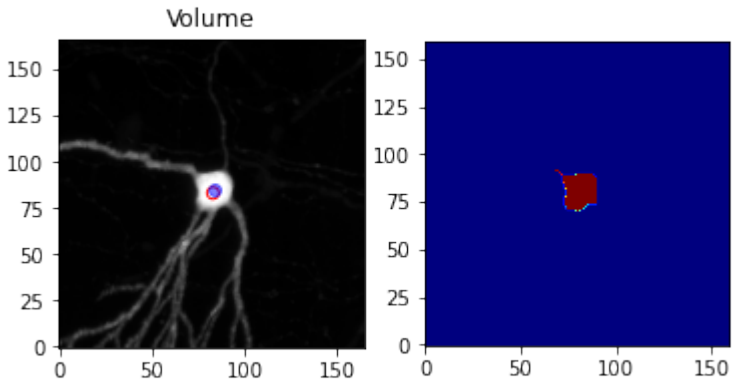
[3]:
```
for volume_key in volume_keys:
    volume_coords = np.array(os.path.basename(volume_key).split("_")).astype(float)
    volume_vox_min = np.round(np.divide(volume_coords[:3], res)).astype(int)
    volume_vox_max = np.round(np.divide(volume_coords[3:], res)).astype(int)

    bbox = Bbox(volume_vox_min, volume_vox_max)
    volume = ngl_sess.pull_bounds_img(bbox)

    somas_obj = s3.Object("open-neurodata", f"{somas_prefix}/{volume_key}").get()
    somas = np.load(BytesIO(somas_obj["Body"].read()))
    rel_soma_coords = np.array(
        [np.round(np.divide(c, res)).astype(int) - volume_vox_min for c in somas]
```

(continues on next page)

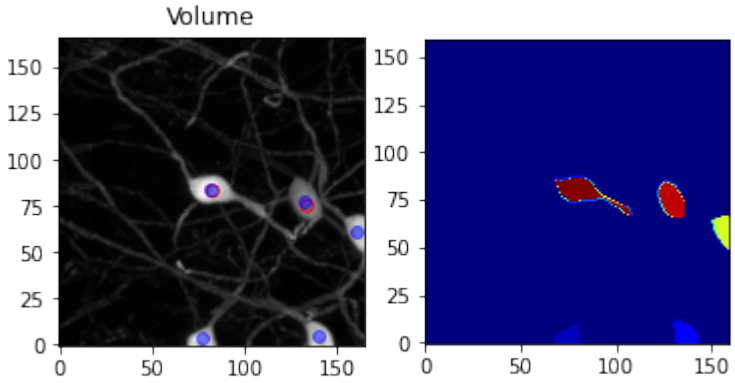
(continued from previous page)

```
    )

    label, rel_pred_centroids, out = find_somas(volume, res)

    _, axes = plt.subplots(1, 2)

    ax = axes[0]
    vol_proj = np.amax(volume, axis=2)
    ax.imshow(vol_proj, cmap="gray", origin="lower")
    ax.scatter(
        rel_soma_coords[:, 1],
        rel_soma_coords[:, 0],
        c="none",
        edgecolor="r",
        label="Ground truth",
    )
    if label == 1:
        ax.scatter(rel_pred_centroids[:, 1], rel_pred_centroids[:, 0], c="b", alpha=0.
↪5)
    ax.set_title("Volume")

    ax = axes[1]
    mask_proj = np.amax(out, axis=2)
    ax.imshow(mask_proj, cmap="jet", vmin=0, origin="lower")
    plt.show()
```

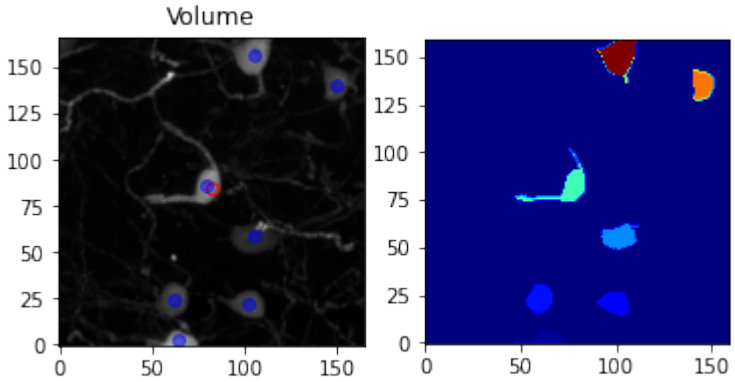
```
Downloading:   0%|          | 0/18 [00:01<?, ?it/s]
```




```
Downloading:   0%|          | 0/12 [00:04<?, ?it/s]
```



```
Downloading:   0%|          | 0/18 [00:02<?, ?it/s]
```



2.1.4 Preprocessing

Connected Component Manipulation

The Brainlit package contains some functions to manipulate connected components. This is usually done on binary images, especially labels.

[ ]:
```
import numpy as np
from brainlit.preprocessing import getLargestCC, removeSmallCCs
from skimage import data
import matplotlib.pyplot as plt

img = data.binary_blobs(512, 0.1, n_dim=2, volume_fraction=0.5, seed=10)
largest_cc = getLargestCC(img)
large_cc = removeSmallCCs(img, 10000)


plt.figure()
plt.subplot(1, 3, 1)
plt.imshow(img)
plt.title("Original Image")
plt.axis("Off")
plt.subplot(1, 3, 2)
plt.imshow(largest_cc)
plt.title("Largest CC")
plt.axis("Off")
plt.subplot(1, 3, 3)
plt.imshow(large_cc)
plt.title("Small CCs Removed")
plt.axis("Off")
plt.show()
```

[ ]:

Gabor Filters

Gabor filters are used to extract features from an image. They extract spatial frequency content in a certain direction. Brainlit's Gabor implementation can be used for nD images

[ ]:
```
import numpy as np
from brainlit.preprocessing import gabor_filter
from skimage import data
import matplotlib.pyplot as plt

img = data.brick()

frequencies = [0.1, 0.1, 0.25, 0.25]
phi = [0, np.pi / 2, 0, np.pi / 2]

plt.figure()
plt.imshow(img, cmap="gray")
plt.axis("off")
plt.title("Original Image")
plt.show()

plt.figure()
for i in range(4):
    plt.subplot(2, 2, i + 1)
    filtered = gabor_filter(img, 5, phi[i], frequencies[i], truncate=3)
    plt.imshow(filtered[0], cmap="gray")
    plt.xticks([])
    plt.yticks([])
    if i == 0:
        plt.title("Orientation=0")
        plt.ylabel("Frequency=0.1")
    elif i == 1:
        plt.title("Orientation=\u03C0/2")
    elif i == 2:
        plt.ylabel("Frequency=0.25")
plt.show()
```

[ ]:

2.1.5 Vizualization

These tutorials demonstrate tools to load and visualize data from s3 buckets or .swc files.

Loading neurons from s3

[1]:
```
import numpy as np
from skimage import io
from pathlib import Path
from brainlit.utils.session import NeuroglancerSession
from brainlit.utils.Neuron_trace import NeuronTrace
import napari
from napari.utils import nbscreenshot

%gui qt
```

Loading entire neuron from AWS

`s3_trace = NeuronTrace(s3_path,seg_id,mip)` to create a NeuronTrace object with s3 file path `swc_trace = NeuronTrace(swc_path)` to create a NeuronTrace object with swc file path 1. `s3_trace.get_df()` to output the s3 NeuronTrace object as a pd.DataFrame 2. `swc_trace.get_df()` to output the swc NeuronTrace object as a pd.DataFrame 3. `swc_trace.generate_df_subset(list_of_voxels)` creates a smaller subset of the original dataframe with coordinates in img space 4. `swc_trace.get_df_voxel()` to output a DataFrame that converts the coordinates from spatial to voxel coordinates 5. `swc_trace.get_graph()` to output the s3 NeuronTrace object as a netwrokx.DiGraph 6. `swc_trace.get_paths()` to output the s3 NeuronTrace object as a list of paths 7. `ViewerModel.add_shapes` to add the paths as a shape layer into the napari viewer 8. `swc_trace.get_sub_neuron(bounding_box)` to output NeuronTrace object as a graph cropped by a bounding box 9. `swc_trace.get_sub_neuron(bounding_box)` to output NeuronTrace object as paths cropped by a bounding box

1. `s3_trace.get_df()`

This function outputs the s3 NeuronTrace object as a pd.DataFrame. Each row is a vertex in the swc file with the following information:

`sample number`

`structure identifier`

`x coordinate`

`y coordinate`

`z coordinate`

`radius of dendrite`

`sample number of parent`

The coordinates are given in spatial units of micrometers (swc specification)

[3]:
```
"""
s3_path = "s3://open-neurodata/brainlit/brain1_segments"
seg_id = 2
mip = 1
```

(continues on next page)

(continued from previous page)

```
s3_trace = NeuronTrace(s3_path, seg_id, mip)
df = s3_trace.get_df()
df.head()
"""
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  5.13it/s]
Downloading: 100%|| 1/1 [00:00<00:00,  5.82it/s]
```

[3]:

	sample	structure	x	y	z	r	parent
0	1	0	4717.0	4464.0	3855.0	1.0	-1
1	4	192	4725.0	4439.0	3848.0	1.0	1
2	7	64	4727.0	4440.0	3849.0	1.0	4
3	8	0	4732.0	4442.0	3850.0	1.0	7
4	14	0	4749.0	4439.0	3856.0	1.0	8

2. `swc_trace.get_df()`

This function outputs the swc NeuronTrace object as a pd.DataFrame. Each row is a vertex in the swc file with the following information:

`sample number`

`structure identifier`

`x coordinate`

`y coordinate`

`z coordinate`

`radius of dendrite`

`sample number of parent`

The coordinates are given in spatial units of micrometers (swc specification)

[4]:
```
"""
swc_path = str(Path().resolve().parents[2] / "data" / "data_octree" / "consensus-swcs
↪" / '2018-08-01_G-002_consensus.swc')

swc_trace = NeuronTrace(path=swc_path)
df = swc_trace.get_df()
df.head()
"""
```

[4]:

	sample	structure	x	y	z	r	parent
0	1	0	-387.114438	1928.663327	-1846.508302	1.0	-1
1	2	0	-385.023123	1917.704355	-1833.659048	1.0	1
2	3	0	-385.023123	1892.742251	-1846.508302	1.0	2
3	4	0	-381.438012	1881.174447	-1858.369151	1.0	3
4	5	0	-388.010715	1881.783279	-1873.195213	1.0	4

3. swc_trace.generate_df_subset(list_of_voxels)

This function creates a smaller subset of the original dataframe with coordinates in img space. Each row is a vertex in the swc file with the following information:

`sample number`

`structure identifier`

`x coordinate`

`y coordinate`

`z coordinate`

`radius of dendrite`

`sample number of parent`

The coordinates are given in same spatial units as the image file when using `ngl.pull_vertex_list`

[5]:
```
"""# Choose vertices to use for the subneuron
subneuron_df = df[0:3]
vertex_list = subneuron_df['sample'].array

# Define a neuroglancer session
url = "s3://open-neurodata/brainlit/brain1"
mip = 1
ngl = NeuroglancerSession(url, mip=mip)

# Get vertices
seg_id = 2
buffer = 10
img, bounds, vox_in_img_list = ngl.pull_vertex_list(seg_id=seg_id, v_id_list=vertex_
↪list, buffer = buffer, expand = True)

df_subneuron = swc_trace.generate_df_subset(vox_in_img_list.tolist(),subneuron_
↪start=0,subneuron_end=3 )
print(df_subneuron)
"""
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  6.08it/s]
Downloading: 100%|| 1/1 [00:00<00:00,  6.95it/s]
Downloading: 100%|| 1/1 [00:00<00:00,  5.02it/s]
Downloading:   0%|          | 0/4 [00:01<?, ?it/s]   sample  structure    x    y    z␣
↪    r  parent
0       1          0  106  106  112  1.0      -1
1       2          0  121   80   61  1.0       1
2       3          0   61   55   49  1.0       2
```

4. `swc_trace.get_df_voxel()`

If we want to overlay the swc file with a corresponding image, we need to make sure that they are in the same coordinate space. Because an image in an array of voxels, it makes sense to convert the vertices from spatial units into voxel units.

Given the `spacing` (spatial units/voxel) and `origin` (spatial units) of the image, `swc_to_voxel` does the conversion by using the following equation:

$voxel = \frac{spatial - origin}{spacing}$

[6]:
```
# spacing = np.array([0.29875923,0.3044159,0.98840415])
# origin = np.array([70093.276,15071.596,29306.737])


# df_voxel = swc_trace.get_df_voxel(spacing=spacing, origin=origin)
# df_voxel.head()
```

[6]:

	sample	structure	x	y	z	r	parent
0	1	0	-235910	-43174	-31519	1.0	-1
1	2	0	-235903	-43210	-31506	1.0	1
2	3	0	-235903	-43292	-31519	1.0	2
3	4	0	-235891	-43330	-31531	1.0	3
4	5	0	-235913	-43328	-31546	1.0	4

5. `swc_trace.get_graph()`

A neuron is a graph with no cycles (tree). While napari does not support displaying graph objects, it can display multiple paths.

The DataFrame already contains all the possible edges in the neurons. Each row in the DataFrame is an edge. For example, from the above we can see that `sample 2` has `parent 1`, which represents edge `(1,2)`. `sample 1` having `parent -1` means that `sample 1` is the root of the tree.

`swc_trace.get_graph()` converts the NeuronTrace object into a networkx directional graph.

[7]:
```
# G = swc_trace.get_graph()
# print('Number of nodes:', len(G.nodes))
# print('Number of edges:', len(G.edges))
# print('\n')
# print('Sample 1 coordinates (x,y,z)')
# print(G.nodes[1]['x'],G.nodes[1]['y'],G.nodes[1]['z'])
```

```
Number of nodes: 1650
Number of edges: 1649


Sample 1 coordinates (x,y,z)
-387 1928 -1846
```

6. `swc_trace.get_paths()`

This function returns the NeuronTrace object as a list of non-overlapping paths. The union of the paths forms the graph.

The algorithm works by:

1. Find longest path in the graph (networkx.algorithms.dag.dag_longest_path)
2. Remove longest path from graph
3. Repeat steps 1 and 2 until there are no more edges left in the graph

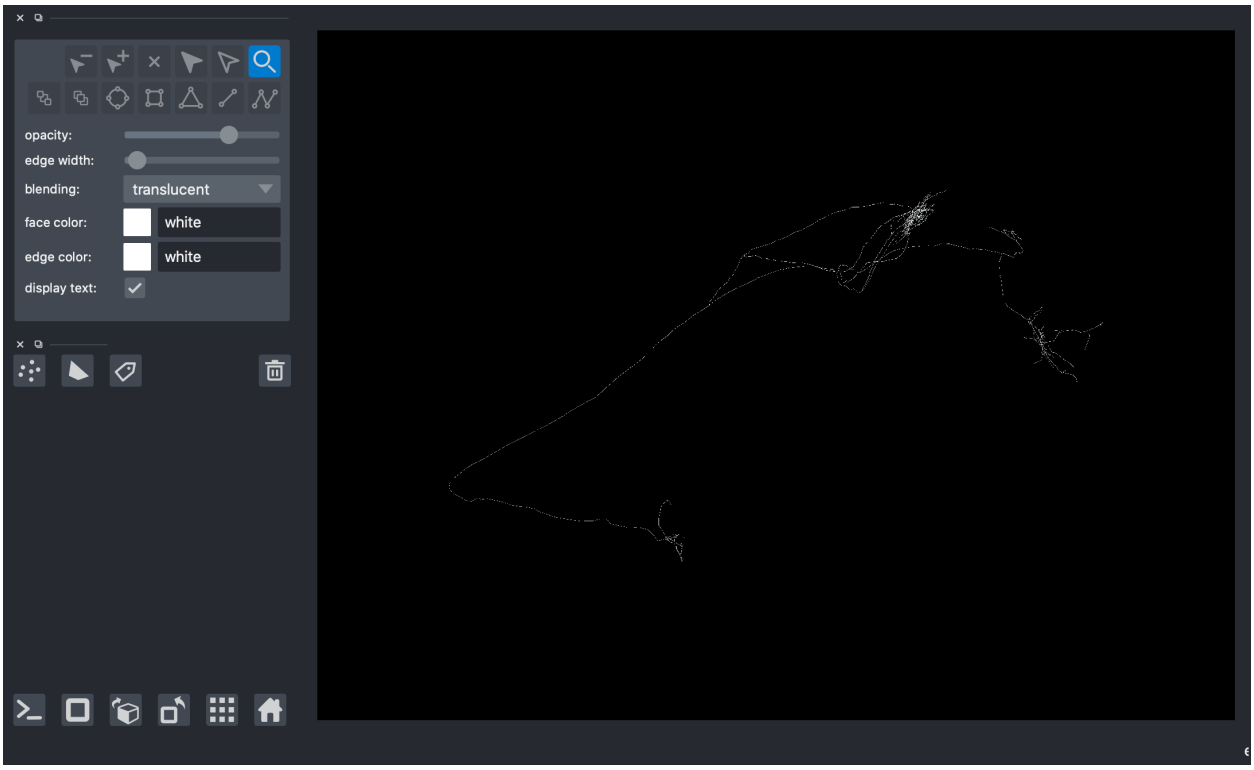
[8]:
```
# paths = swc_trace.get_paths()
# print(f"The graph was decomposed into {len(paths)} paths")
```

```
The graph was decomposed into 179 paths
```

7. `ViewerModel.add_shapes`

napari displays "layers". The most common layer is the image layer. In order to display the neuron, we use `path` from the shapes layer

[9]:
```
# viewer = napari.Viewer(ndisplay=3)
# viewer.add_shapes(data=paths, shape_type='path', edge_color='white', name='Skeleton␣
↪2')
# nbscreenshot(viewer)
```

[9]:


Loading sub-neuron

The image of the entire brain has dimensions of (33792, 25600, 13312) voxels. G-002 spans a sub-image of (7386, 9932, 5383) voxels. Both are too big to load in napari and overlay the neuron. To circumvent this, we can crop out a smaller region of the neuron, load the sub-neuron, and load the corresponding sub-image.

In order to get a sub-neuron, we need to specify the `bounding_box` that will be used to crop the neuron. `bounding_box` is a length 2 tuple. The first element is one corner of the bounding box (inclusive) and the second element is the opposite corner of the bounding box (exclusive). Both corners are in voxel units.

`add_swc` can do all of this automatically when given `bounding_box` by following these steps:

1. `read_s3` to read the swc file into a pd.DataFrame
2. `swc_to_voxel` to convert the coordinates from spatial to voxel coordinates
3. `df_to_graph` to convert the DataFrame into a netwrokx.DiGraph **3.1 ``swc.get_sub_neuron`` to crop the graph by ``bounding_box``**
4. `graph_to_paths` to convert from a graph into a list of paths
5. `ViewerModel.add_shapes` to add the paths as a shape layer into the napari viewer

8. `swc_trace.get_sub_neuron(bounding_box)`

9. `swc_trace.get_sub_neuron_paths(bounding_box)`

This function crops a graph by removing edges. It removes edges that do not intersect the bounding box.

Edges that intersect the bounding box will have at least one of its vertices be contained by the bounding box. The algorithm follows this principle by checking the neighborhood of vertices.

For each vertex v in the graph:

1. Find vertices belonging to local neighborhood of v
2. If vertex v or any of its local neighborhood vertices are in the bounding box, do nothing. Otherwise, remove vertex v and its edges from the graph

We check the neighborhood of v along with v because we want the sub-neuron to show all edges that pass through the bounding box, including edges that are only partially contained.

`swc_trace.get_sub_neuron(bounding_box)` returns a sub neuron in graph format `swc_trace.get_sub_neuron_paths(bounding_box)` returns a sub neuron in paths format

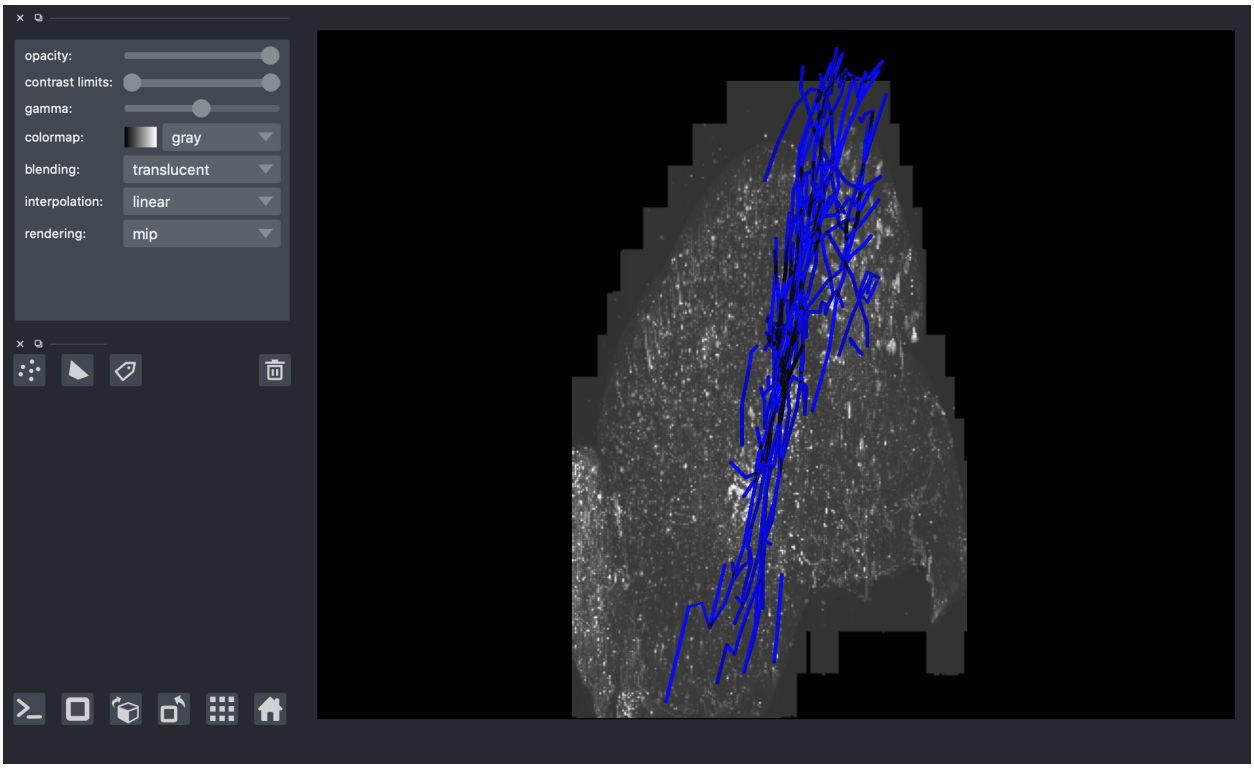
[10]:
```
# # Create an NGL session to get the bounding box
# url = "s3://open-neurodata/brainlit/brain1"
# mip = 1
# ngl = NeuroglancerSession(url, mip=mip)

# img, bbbox, vox = ngl.pull_chunk(2, 300, 1)
# bbox = bbbox.to_list()
# box = (bbox[:3], bbox[3:])
# print(box)
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  6.28it/s]
Downloading: 52it [00:02, 19.61it/s]([7392, 2300, 3120], [7788, 2600, 3276])
```

[11]:
```
# G_sub = s3_trace.get_sub_neuron(box)
# paths_sub = s3_trace.get_sub_neuron_paths(box)
# print(len(G_sub))
# viewer = napari.Viewer(ndisplay=3)
# viewer.add_shapes(data=paths_sub, shape_type='path', edge_color='blue', name='sub-
↪neuron')

# # overlay corresponding image (random image but correct should be G-002_15312-4400-
↪6448_15840-4800-6656.tif' )
# image_path = str(Path().resolve().parents[2] / "data" / "data_octree" / 'default.0.
↪tif')
# img_comp = io.imread(image_path)
# img_comp = np.swapaxes(img_comp,0,2)

# viewer.add_image(img_comp)
# nbscreenshot(viewer)
```

```
459
```

[11]:



[ ]:

Visualization of Neighborhoods Tutorial

Objective: This tutorial covers how to perform visualize neighborhoods based on two approaches.

```
1) Grabbing a bounding box region a vertex
2) Grabbing n neighbors around a vertex
```

[1]:
```
from brainlit.utils.Neuron_trace import NeuronTrace
from brainlit.utils.session import NeuroglancerSession
import numpy as np
from cloudvolume import CloudVolume
import napari
from napari.utils import nbscreenshot

%gui qt5
```

Reading data from s3 path

[3]:
```
"""
s3_path = "s3://open-neurodata/brainlit/brain1_segments"
seg_id,  v_id, mip = 2, 10, 1 # skeleton/neuron id, index/row of df, resolution␣
↪quality

s3_trace = NeuronTrace(path=s3_path,seg_id=seg_id,mip=mip)
df = s3_trace.get_df()
df.head()
"""
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  8.06it/s]
Downloading: 100%|| 1/1 [00:00<00:00,  9.13it/s]
```

[3]:

	sample	structure	x	y	z	r	parent
0	1	0	4717.0	4464.0	3855.0	1.0	-1
1	4	192	4725.0	4439.0	3848.0	1.0	1
2	7	64	4727.0	4440.0	3849.0	1.0	4
3	8	0	4732.0	4442.0	3850.0	1.0	7
4	14	0	4749.0	4439.0	3856.0	1.0	8

Converting dataframe to graph data structure to understand how vertices are connected

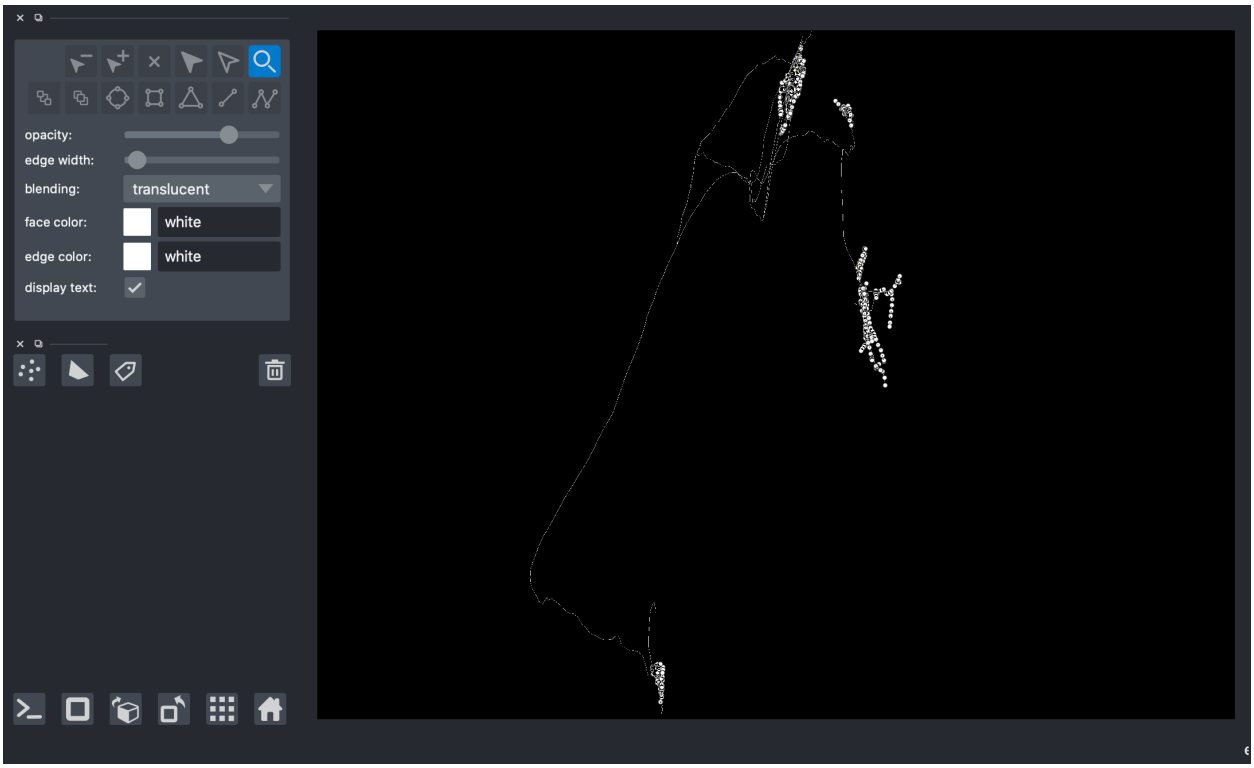
[4]:
```
# G = s3_trace.get_graph()
# paths = s3_trace.get_paths()
# print(f"The graph was decomposed into {len(paths)} paths")
```

```
The graph was decomposed into 179 paths
```

Plotting the entire skeleton/neuron

[5]:
```
# viewer = napari.Viewer(ndisplay=3)
# # it is important that the number of paths put into 'data=' is at the most 1024
# viewer.add_points(data=np.concatenate(paths)[804:], edge_width=2, edge_color='white
↪', name='Skeleton 2')
# viewer.add_shapes(data=paths, shape_type='path', edge_color='white', name='Skeleton␣
↪2')
# nbscreenshot(viewer)
```

[5]:



Bounding Box Method

Creating a bounding box based on a particular vertex of interest in order to get a group of neurons neighboring the vertex of interest

[6]:
```
# url = "s3://open-neurodata/brainlit/brain1"
# mip = 1
# ngl = NeuroglancerSession(url, mip=mip)

# img, bbbox, vox = ngl.pull_chunk(2, 300, 1)
# bbox = bbbox.to_list()
# box = (bbox[:3], bbox[3:])
# print(box)
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  3.20it/s]
Downloading: 52it [00:02, 21.56it/s]([7392, 2300, 3120], [7788, 2600, 3276])
```

Getting all the coordinates of the group surrounding the vertex of interest using get_sub_neuron()
Note: data correction step necessary due to recentering in function!

[7]:
```
# G_sub = s3_trace.get_sub_neuron(box)

# # preventing the re-centring of nodes to the bounding box corner (origin of the new␣
↪coordinate frame)
```

(continues on next page)

(continued from previous page)

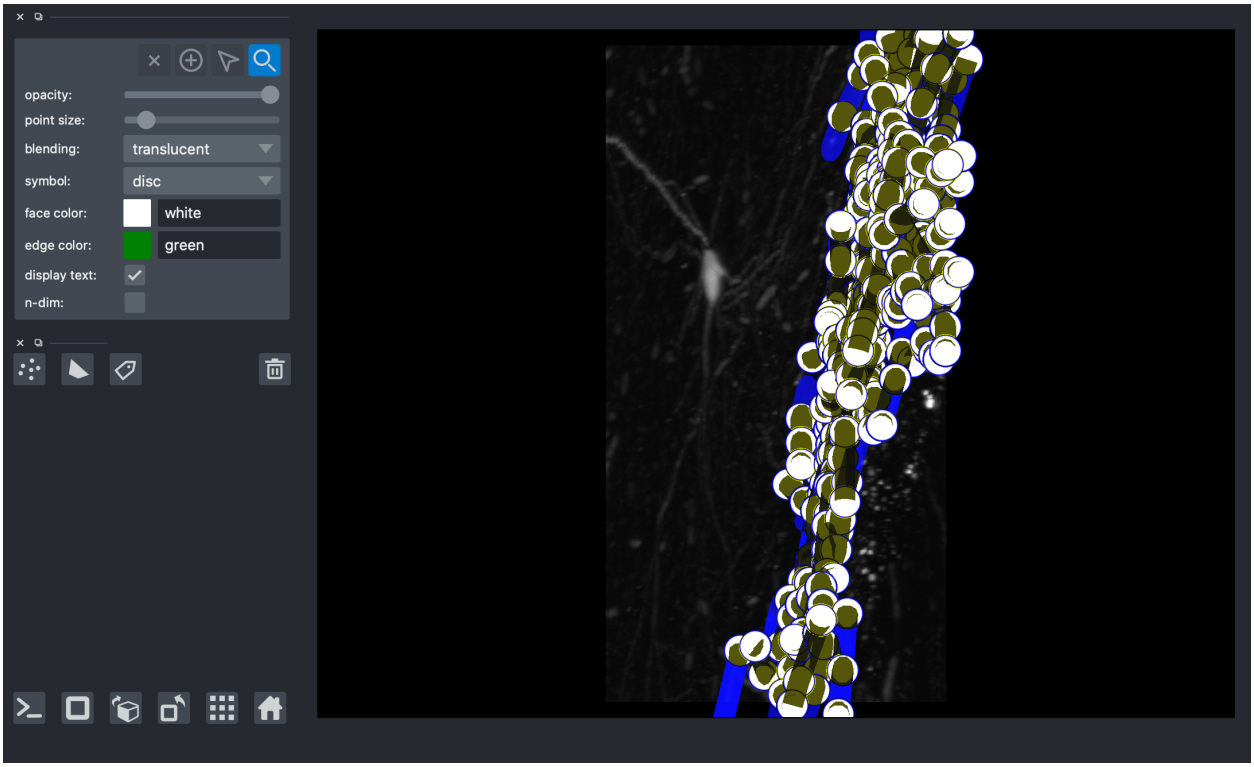
```
# for id in list(G_sub.nodes):
#     G_sub.nodes[id]["x"] = G_sub.nodes[id]["x"] + box[0][0]
#     G_sub.nodes[id]["y"] = G_sub.nodes[id]["y"] + box[0][1]
#     G_sub.nodes[id]["z"] = G_sub.nodes[id]["z"] + box[0][2]

# paths_sub = s3_trace.get_sub_neuron_paths(box)
```

Plotting vertex and vertex neighborhood

[11]:
```
# # grab the coordinates of the vertex from the skeleon
# cv_skel = CloudVolume(s3_path, mip=mip, use_https=True)
# skel = cv_skel.skeleton.get(seg_id)
# vertex = skel.vertices[v_id]/cv_skel.scales[mip]["resolution"]
# print(vertex)

# viewer = napari.Viewer(ndisplay=3)
# viewer.add_image(np.squeeze(np.array(img)))
# viewer.add_points(data=np.concatenate(paths_sub), edge_width=1, edge_color='blue',␣
↪name='Skeleton 2')
# viewer.add_shapes(data=paths_sub, shape_type='path', edge_color='blue', name=
↪'Neighborhood',edge_width=5)

# # display vertex
# viewer.add_points(data=np.array(vertex), edge_width=2, edge_color='green', name=
↪'vertex')
# nbscreenshot(viewer)
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  7.97it/s]
[6263.70139759 6573.55819874 2210.2198857 ]
```

[11]:



Neighbors Method

[12]:
```
# # grab the coordinates of the vertex from the skeleon
# cv_skel = CloudVolume(s3_path, mip=mip, use_https=True)
# skel = cv_skel.skeleton.get(seg_id)
# vertex = skel.vertices[v_id]/cv_skel.scales[mip]["resolution"]
# print(vertex)

# # figure out where the vertex information is stored in the dataframe
# x, y, z = np.round((vertex))[0], np.round((vertex))[1], np.round((vertex))[2]
# slice_df = (df[(df.x == x)&(df.y==y)&(df.z==z)])
# v_idx = np.where((df.x == x)&(df.y==y)&(df.z==z))
# v_idx = v_idx[0][0]
# print(v_idx)
# slice_df.head()
```

```
Downloading: 100%|| 1/1 [00:00<00:00,  7.64it/s][6263.70139759 6573.55819874 2210.
↪2198857 ]
469
```

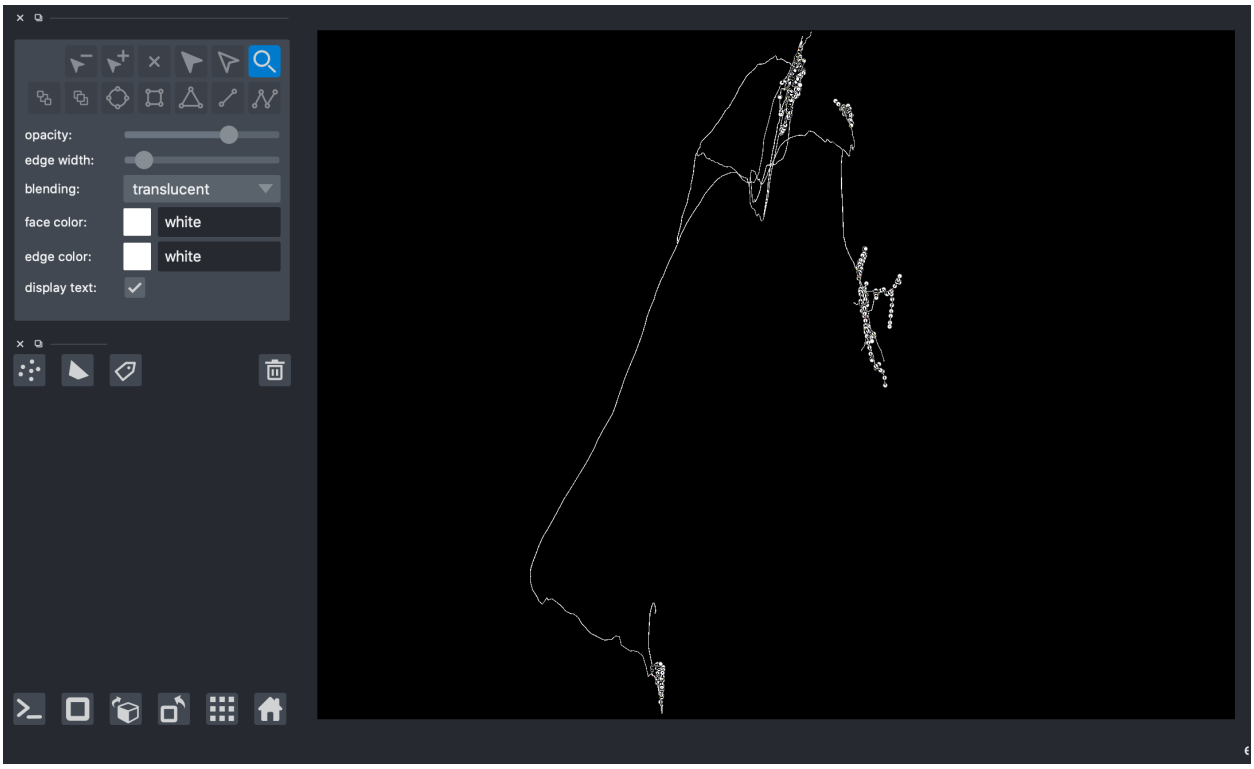
[12]:

	sample	structure	x	y	z	r	parent
469	434	0	6264.0	6574.0	2210.0	1.0	431

On another napari window, plot again the entire neuron/skeleton.

[13]:
```
# viewer = napari.Viewer(ndisplay=3)
# viewer.add_points(data=np.concatenate(paths, axis=0)[1024:], edge_width=2, edge_
↪color='white', name='all_points')
# viewer.add_shapes(data=paths, shape_type='path', edge_color='white', edge_width=3,␣
↪name='skeleton')
# nbscreenshot(viewer)
```

[13]:



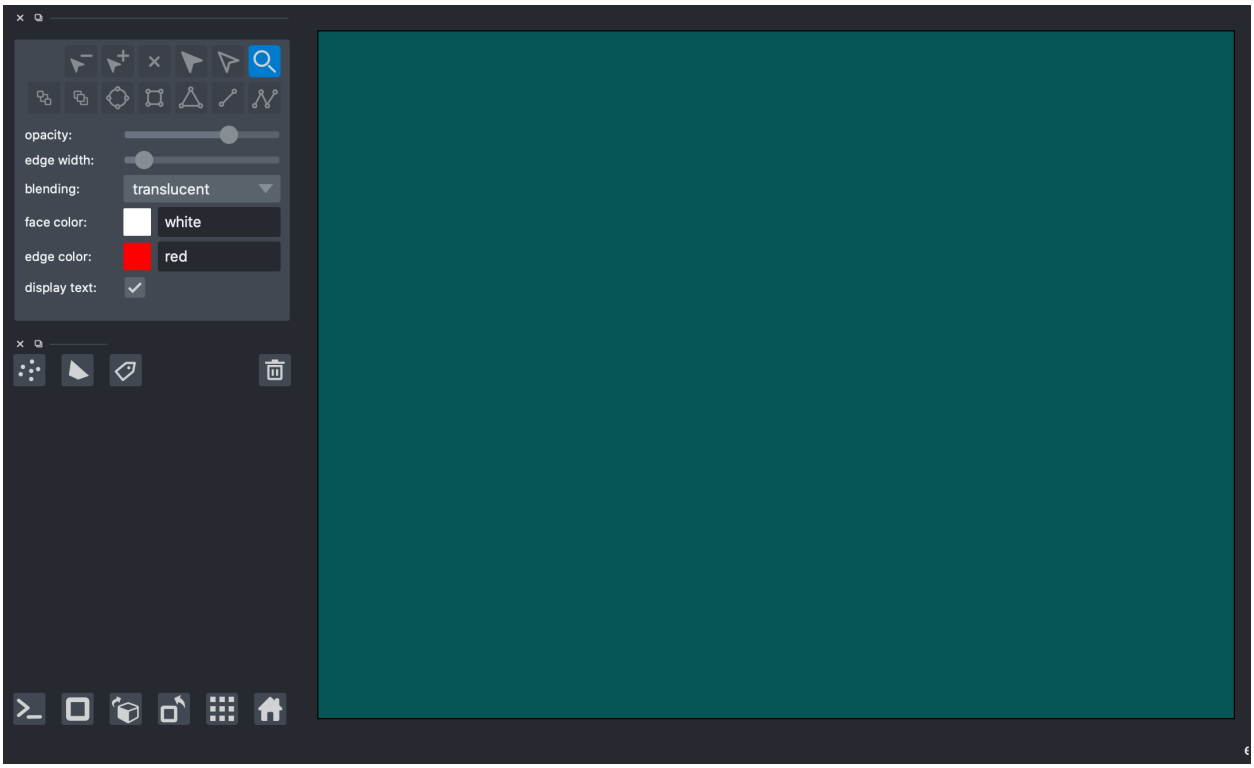
Get the coordinates of the neighobrs around vertex of interest using get_bfs_subgraph() and graphs_to_paths

[14]:
```
# v_id_pos = v_idx  # the row index/number of the data frame
# depth = 10  # the depth up to which the graph must be constructed

# G_bfs, _, paths_bfs =s3_trace.get_bfs_subgraph(int(v_id_pos), depth, df=df)  #␣
↪perform Breadth first search to obtain a graph of interest
```

Plot the vertex and vertex neighborhood

[22]:
```
# x,y,z = df.iloc[v_id_pos]['x'], df.iloc[v_id_pos]['y'], df.iloc[v_id_pos]['z']

# # display vertex
# viewer = napari.Viewer(ndisplay=3)

# viewer.add_points(data=np.array([x,y,z]), edge_width=5, edge_color='orange', name=
↪'bfs_vertex')

# # display all neighbors around vertex
# viewer.add_points(data=np.concatenate(paths_bfs), edge_color='red', edge_width=2,␣
↪name='bfs_points')
# viewer.add_shapes(data=paths_bfs, shape_type='path', edge_color='red', edge_width=3,
↪ name='bfs_sub_skeleton')
# nbscreenshot(viewer)
```

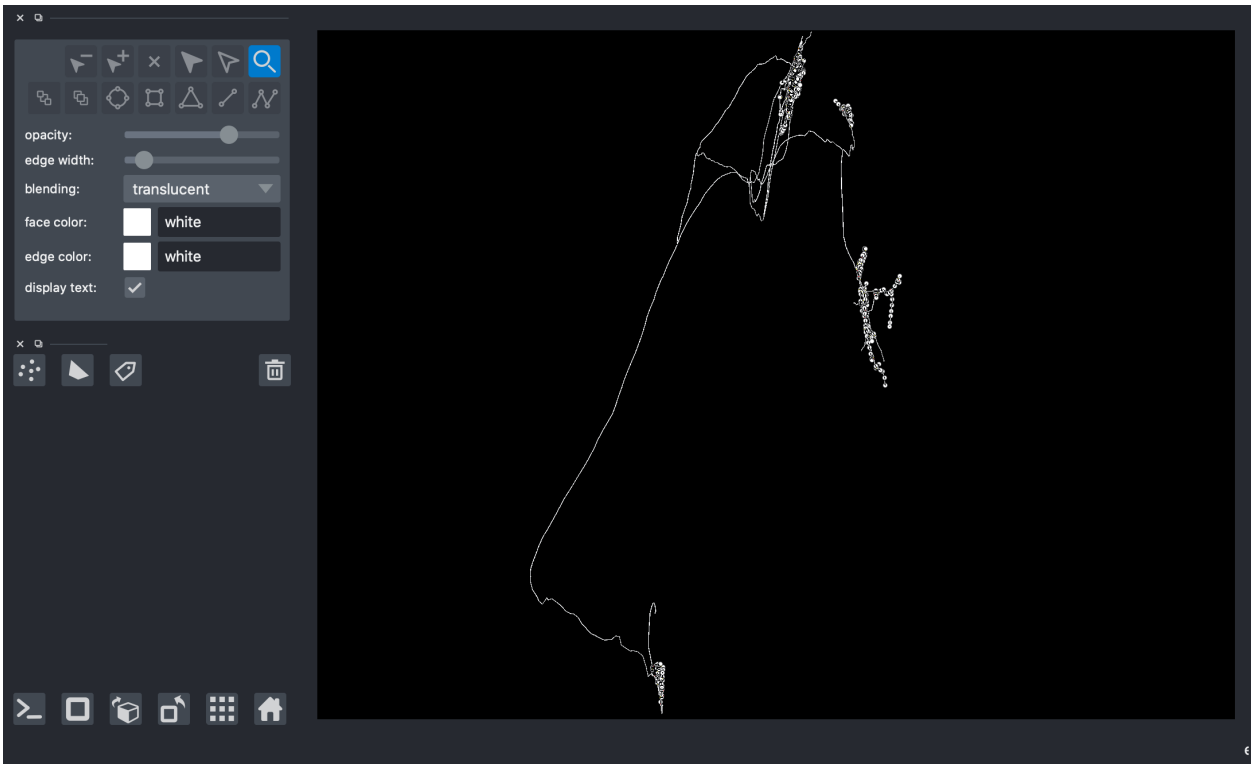
[22]:



Visualizing the output of both methods overlaid

Create new napari window

[21]:
```
# viewer = napari.Viewer(ndisplay=3)
# viewer.add_points(data=np.concatenate(paths, axis=0)[1024:], edge_width=2, edge_
↪color='white', name='all_points')
# viewer.add_shapes(data=paths, shape_type='path', edge_color='white', edge_width=3,␣
↪name='full_skeleton')
# nbscreenshot(viewer)
```

[21]:


Plot vertices and neighborhoods of each method on the same napari window to compare method outputs

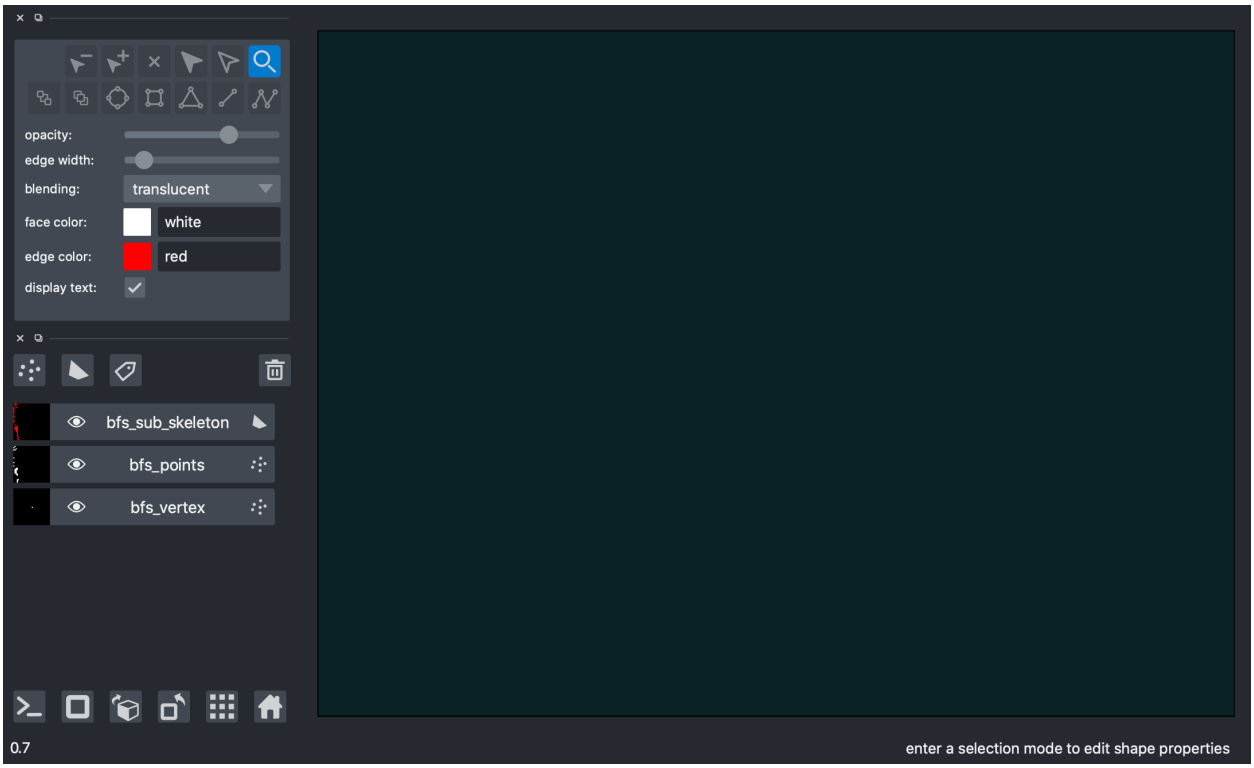
[23]:
```
# # display vertex of the boundary method
# viewer.add_points(data=np.array(vertex), edge_width=5, edge_color='green', name=
↪'boundary_vertex')

# # display all neighbors around vertex of boundary method
# viewer.add_points(data=np.concatenate(paths_sub), edge_width=2, edge_color='blue',␣
↪name='boundary_skeleton_pts')
# viewer.add_shapes(data=paths_sub, shape_type='path', edge_color='blue', name=
↪'boundary_skeleton_lines',edge_width=5)

# # display vertex of the bfs method
# x,y,z = df.iloc[v_id_pos]['x'], df.iloc[v_id_pos]['y'], df.iloc[v_id_pos]['z']
# viewer.add_points(data=np.array([x,y,z]), edge_width=5, edge_color='orange', name=
↪'bfs_vertex')

# # display all neighbors around vertex of bfs method
# viewer.add_points(data=np.concatenate(paths_bfs), edge_color='red', edge_width=2,␣
↪name='bfs_skeleton_pts')
```

(continues on next page)

(continued from previous page)

```
# viewer.add_shapes(data=paths_bfs, shape_type='path', edge_color='red', edge_width=3,
↪ name='bfs_skeleton_lines')
# nbscreenshot(viewer)
```

[23]:



[ ]:

2.2 Reference

2.2.1 Algorithms

Fragment Generation

class brainlit.algorithms.generate_fragments.**state_generation**(*image_path: Union[str, Path]*, *new_layers_dir: Union[str, Path]*, *ilastik_program_path: str*, *ilastik_project_path: str*, *fg_channel: int = 0*, *chunk_size: List[float] = [375, 375, 125]*, *soma_coords: List[list] = []*, *resolution: List[float] = [0.3, 0.3, 1]*, *parallel: int = 1*, *prob_path: Union[str, Path] = None*, *fragment_path: Union[str, Path] = None*, *tiered_path: Union[str, Path] = None*, *states_path: Union[str, Path] = None*)

This class encapsulates the processing that turns an image into a set of fragments with endpoints etc. needed to perform viterbrain tracing.

Parameters

- **image_path** (*str or pathlib.Path*) -- Path to image zarr.
- **new_layers_dir** (*str or pathlib.Path*) -- Path to directory where new layers will be written.
- **ilastik_program_path** (*str*) -- Path to ilastik program.
- **ilastik_project_path** (*str*) -- Path to ilastik project for segmentation of image.
- **fg_channel** (*int*) -- Channel of image taken to be foreground.
- **chunk_size** (*List[float]*) -- Chunk size too be used in parallel processing. Defaults to [375, 375, 125].
- **soma_coords** (*List[list]*) -- List of coordinates of soma centers. Defaults to [].
- **resolution** (*List[float*) -- Resolution of image in microns. Defaults to [0.3, 0.3, 1].
- **parallel** (*int*) -- Number of threads to use for parallel processing. Defaults to 1.
- **prob_path** (*str or pathlib.Path*) -- Path to alrerady computed probability image (ilastik output). Defaults to None.

- **fragment_path** (*str or pathlib.Path*) -- Path to alrerady computed fragment image. Defaults to None.
- **tiered_path** (*str or pathlib.Path*) -- Path to alrerady computed tiered image. Defaults to None.
- **states_path** (*str or pathlib.Path*) -- Path to alrerady computed states file. Defaults to None.

image_path
Path to image zarr.

Type str

new_layers_dir
Path to directory where new layers will be written.

Type str

ilastik_program_path
Path to ilastik program.

Type str

ilastik_project_path
Path to ilastik project for segmentation of image.

Type str

fg_channel
Channel of image taken to be foreground.

Type int

chunk_size
Chunk size too be used in parallel processing.

Type List[float], optional

soma_coords
List of coordinates of soma centers.

Type List[list], optional

resolution
Resolution of image in microns.

Type List[float], optional

parallel
Number of threads to use for parallel processing.

Type int, optional

prob_path
Path to alrerady computed probability image (ilastik output).

Type str, optional

fragment_path
Path to alrerady computed fragment image.

Type str, optional

tiered_path
Path to alrerady computed tiered image.

Type str, optional

states_path
Path to alrerady computed states file.

Type str, optional

Raises

- **ValueError** -- Image must be four dimensional (cxyz)
- **ValueError** -- Chunks must include all channels and be 4D.
- **ValueError** -- Already computed images must match image in spatial dimensions.

compute_bfs(*self*)
Compute bfs from highest degree node

compute_edge_weights(*self*)
Create viterbrain object and compute edge weights

compute_frags(*self*)
Compute all fragments for image

compute_image_tiered(*self*)
Compute entire tiered image then reassemble and save as zarr

compute_soma_lbls(*self*)
Compute fragment ids of soma coordinates.

compute_states(*self*, *alg: str = 'nb'*)
Compute entire collection of states

Parameters **alg** (*string, optional*) -- algorithm to use for endpoint estimation. "nb" for neighborhood method, "pc" for principal curves method. Defaults to "nb"

Raises **ValueError** -- erroneously computed endpoints of soma state

predict(*self*, *data_bin: str*)
Run ilastik on zarr image

Parameters **data_bin** (*str*) -- path to directory to store intermediate files

Connect Fragments

class brainlit.algorithms.connect_fragments.**ViterBrain**(*G: nx.Graph*, *tiered_path: str*, *fragment_path: str*, *resolution: List[float]*, *coef_curv: float*, *coef_dist: float*, *coef_int: float*, *parallel: int = 1*)

compute_all_costs_dist(*self*, *frag_frag_func: Callable*, *frag_soma_func: Callable*)
Splits up transition computation tasks then assembles them into networkx graph

Parameters

- **frag_frag_func** (*function*) -- function that computes transition cost between fragments
- **frag_soma_func** (*function*) -- function that computes transition cost between fragments

compute_all_costs_int(*self*)
Splits up transition computation tasks then assembles them into networkx graph

frag_frag_dist(*self*, *pt1: List[float]*, *orientation1: List[float]*, *pt2: List[float]*, *orientation2: List[float]*, *verbose: bool = False*)
Compute cost of transition between two fragment states

Parameters

- **pt1** (*list of floats*) -- first coordinate
- **orientation1** (*list of floats*) -- orientation at first coordinate
- **pt2** (*list of floats*) -- second coordinate
- **orientation2** (*list of floats*) -- orientation at second coordinate
- **verbose** (*bool, optional*) -- Print transition cost information. Defaults to False.

Raises

- **ValueError** -- if an orientation is not unit length
- **ValueError** -- if distance or curvature cost is nan

Returns cost of transition

Return type [float]

frag_soma_dist(*self*, *point: List[float]*, *orientation: List[float]*, *soma_lbl: int*, *verbose: bool = False*)
Compute cost of transition from fragment state to soma state

Parameters

- **point** (*list of floats*) -- coordinate on fragment
- **orientation** (*list of floats*) -- orientation at fragment
- **soma_lbl** (*int*) -- label of soma component
- **verbose** (*bool, optional*) -- Prints cost values. Defaults to False.

Raises

- **ValueError** -- if either distance or curvature cost is nan
- **ValueError** -- if the computed closest soma coordinate is not on the soma

Returns cost of transition [list of floats]: closest soma coordinate

Return type [float]

shortest_path(*self*, *coord1: List[int]*, *coord2: List[int]*)
Compute coordinate path from one coordinate to another.

Parameters

- **coord1** (*list*) -- voxel coordinate of start point
- **coord2** (*list*) -- voxel coordinate of end point

Raises **ValueError** -- if state sequence contains a soma state that is not at the end

Returns list of voxel coordinates of path

Return type list

Trace Analysis

brainlit.algorithms.trace_analysis.**speed**(*x: np.ndarray*, *t: np.ndarray*, *c: np.ndarray*, *k: np.integer*, *aux_outputs: bool = False*)

Compute the speed of a B-Spline.

The speed is the norm of the first derivative of the B-Spline.

Parameters

- **x** -- A *1xL* array of parameter values where to evaluate the curve. It contains the parameter values where the speed of the B-Spline will be evaluated. It is required to be non-empty, one-dimensional, and real-valued.
- **t** -- A *1xm* array representing the knots of the B-spline. It is required to be a non-empty, non-decreasing, and one-dimensional sequence of real-valued elements. For a B-Spline of degree *k*, at least *2k + 1* knots are required.
- **c** -- A *dxn* array representing the coefficients/control points of the B-spline. Given *n* real-valued, *d*-dimensional points ::math::*x_k = (x_k(1),...,x_k(d))*, *c* is the non-empty matrix which columns are ::math::*x_1^T,...,x_N^T*. For a B-Spline of order *k*, *n* cannot be less than *m-k-1*.
- **k** -- A non-negative integer representing the degree of the B-spline.

Returns A *1xL* array containing the speed of the B-Spline evaluated at *x*

Return type speed

References: .. [Rbbccb9c002d5-1] Kouba, Parametric Equations.

https://www.math.ucdavis.edu/~kouba/Math21BHWDIRECTORY/ArcLength.pdf

brainlit.algorithms.trace_analysis.**curvature**(*x: np.ndarray*, *t: np.ndarray*, *c: np.ndarray*, *k: np.integer*, *aux_outputs: bool = False*)

Compute the curvature of a B-Spline.

The curvature measures the failure of a curve, *r(u)*, to be a straight line. It is defined as

$$k = \|\frac{dT}{ds}\|,$$

where *T* is the unit tangent vector, and *s* is the arc length:

$$T = \frac{dr}{ds}, \quad s = \int_0^t \|r'(u)\| du,$$

where *r(u)* is the position vector as a function of time.

The curvature can also be computed as

$$k = \|r'(t) \times r''(t)\| / \|r'(t)\|^3.$$

Parameters

- **x** -- A *1xL* array of parameter values where to evaluate the curve. It contains the parameter values where the curvature of the B-Spline will be evaluated. It is required to be non-empty, one-dimensional, and real-valued.
- **t** -- A *1xm* array representing the knots of the B-spline. It is required to be a non-empty, non-decreasing, and one-dimensional sequence of real-valued elements. For a B-Spline of degree *k*, at least *2k + 1* knots are required.

- **c** -- A *dxn* array representing the coefficients/control points of the B-spline. Given *n* real-valued, *d*-dimensional points ::math::*x_k = (x_k(1),...,x_k(d))*, *c* is the non-empty matrix which columns are ::math::*x_1^T,...,x_N^T*. For a B-Spline of order *k*, *n* cannot be less than *m-k-1*.
- **k** -- A non-negative integer representing the degree of the B-spline.

Returns A *1xL* array containing the curvature of the B-Spline evaluated at *x*

Return type curvature

References: .. [Rce97e449f49f-1] Máté Attila, The Frenet–Serret formulas.

http://www.sci.brooklyn.cuny.edu/~mate/misc/frenet_serret.pdf

brainlit.algorithms.trace_analysis.**torsion**(*x: np.ndarray*, *t: np.ndarray*, *c: np.ndarray*, *k: np.integer*, *aux_outputs: bool = False*)

Compute the torsion of a B-Spline.

The torsion measures the failure of a curve, *r(u)*, to be planar. If the curvature *k* of a curve is not zero, then the torsion is defined as

$$\tau = -n \cdot b',$$

where *n* is the principal normal vector, and *b'* the derivative w.r.t. the arc length *s* of the binormal vector.

The torsion can also be computed as

$$\tau = |r'(t), r''(t), r'''(t)| / \|r'(t) \times r''(t)\|^2,$$

where *r(u)* is the position vector as a function of time.

Parameters

- **x** -- A *1xL* array of parameter values where to evaluate the curve. It contains the parameter values where the torsion of the B-Spline will be evaluated. It is required to be non-empty, one-dimensional, and real-valued.
- **t** -- A *1xm* array representing the knots of the B-spline. It is required to be a non-empty, non-decreasing, and one-dimensional sequence of real-valued elements. For a B-Spline of degree *k*, at least *2k + 1* knots are required.
- **c** -- A *dxn* array representing the coefficients/control points of the B-spline. Given *n* real-valued, *d*-dimensional points ::math::*x_k = (x_k(1),...,x_k(d))*, *c* is the non-empty matrix which columns are ::math::*x_1^T,...,x_N^T*. For a B-Spline of order *k*, *n* cannot be less than *m-k-1*.
- **k** -- A non-negative integer representing the degree of the B-spline.

Returns A *1xL* array containing the torsion of the B-Spline evaluated at *x*

Return type torsion

References: .. [R8b689f5f8f91-1] Máté Attila, The Frenet–Serret formulas.

http://www.sci.brooklyn.cuny.edu/~mate/misc/frenet_serret.pdf

class brainlit.algorithms.trace_analysis.**CubicHermiteChain**(*x: np.array*, *y: np.array*, *left_dydx: np.array*, *right_dydx: np.array*, *extrapolate=None*)

A third order spline class (continuous piecewise cubic representation), that is fit to a set of positions and one-sided derivatives. This is not a standard spline class (e.g. b-splines), because the derivatives are not necessarily continuous at the knots.

A subclass of PPoly, a piecewise polynomial class from scipy.

class brainlit.algorithms.trace_analysis.**GeometricGraph**(*df: pd.DataFrame = None*, *root=1*)

The shape of the neurons are expressed and fitted with splines in this undirected graph class.

The geometry of the neurons are projected on undirected graphs, based on which the trees of neurons consisted for splines is constructed. It is required that each node has a loc attribute identifying that points location in space, and the location should be defined in 3-dimensional cartesian coordinates. It extends *nx.Graph* and rejects duplicate node input.

fit_spline_tree_invariant(*self*, *spline_type: Union[BSpline, CubicHermiteSpline] = BSpline*, *k=3*)

Construct a spline tree based on the path lengths.

Parameters **spline_type** -- BSpline or CubicHermiteSpline, spline type that will be fit to the data. BSplines are typically used to fit position data only, and CubicHermiteSplines can only be used if derivative, and independent variable information is also known.

Raises

- **ValueError** -- check if every node is unigue in location
- **ValueError** -- check if every node is assigned to at least one edge
- **ValueError** -- check if the graph contains undirected cycle(s)
- **ValueErorr** -- check if the graph has disconnected segment(s)

Returns nx.DiGraph a parent tree with the longest path in the directed graph

Return type spline_tree

Soma Detection

brainlit.algorithms.detect_somas.**find_somas**(*volume: np.ndarray*, *res: list*)

Find bright neuron somas in an input volume.

This simple soma detector assumes that somas are brighter than the rest of the objects contained in the input volume.

To detect somas, these steps are performed:

1. **Check input volume shape.** This detector requires the *x* and *y* dimensions of the input volumes to be larger than *20* pixels.
2. **Zoom volume.** We found that this simple soma detector works best when then input volume has size *160 x 160 x 50*. We use ndimage.zoom to scale the input volume size to the desired shape.
3. **Binarize volume.** We use Otsu thresholding to binarize the image.
4. **Erode the binarized image.** We erode the binarized image with a structuring element which size is directly proportional to the maximum zoom factor applied to the input volume.
5. **Remove unreasonable connected components.** After erosion, we compute the equivalent diameter *d* of each connected component, and only keep those ones such that *5mu m leq d < 21 mu m*
6. **Find relative centroids.** Finally, we compute the centroids of the remaining connected components. The centroids are in voxel units, relative to the input volume.

Parameters

- **volume** (*numpy.ndarray*) -- The 3D image array to run the detector on.

- **res** (*list*) -- A *1 x 3* list containing the resolution of each voxel in *nm*.

Returns

- **label** (*bool*) -- A boolean value indicating whether the detector found any somas in the input volume.
- **rel_centroids** (*numpy.ndarray*) -- A *N x 3* array containing the relative voxel positions of the detected somas.
- **out** (*numpy.ndarray*) -- A *160 x 160 x 50* array containing the detection mask.

Examples

```
>>> # download a volume
>>> dir = "s3://open-neurodata/brainlit/brain1"
>>> dir_segments = "s3://open-neurodata/brainlit/brain1_segments"
>>> volume_keys = "4807349.0_3827990.0_2922565.75_4907349.0_3927990.0_3022565.75"
>>> mip = 1
>>> ngl_sess = NeuroglancerSession(
>>>     mip=mip, url=dir, url_segments=dir_segments, use_https=False
>>> )
>>> res = ngl_sess.cv_segments.scales[ngl_sess.mip]["resolution"]
>>> volume_coords = np.array(os.path.basename(volume_keys).split("_")).
↪astype(float)
>>> volume_vox_min = np.round(np.divide(volume_coords[:3], res)).astype(int)
>>> volume_vox_max = np.round(np.divide(volume_coords[3:], res)).astype(int)
>>> bbox = Bbox(volume_vox_min, volume_vox_max)
>>> img = ngl_sess.pull_bounds_img(bbox)
>>> # apply soma detector
>>> label, rel_centroids, out = find_somas(img, res)
```

2.2.2 Mapping Neurons

Fragment Generation

class brainlit.map_neurons.**DiffeomorphismAction**

Interface for differentiable mappings e.g. transformations that register a brain image to an atlas.

D(*self*, *position: np.array*, *deriv: np.array*, *order: int = 1*)

Evaluate the mapping on a set of derivatives at specified positions.

Parameters

- **position** (*np.array*) -- Coordinates in the original space.
- **deriv** (*np.array*) -- Derivatives at the respective positions
- **order** (*int, optional*) -- Derivative order. Defaults to 1.

Returns Transformed derivatives.

Return type np.array

evaluate(*self*, *position: np.array*)

Evaluate the mapping at specified positions.

Parameters **position** (*np.array*) -- Coordinates in original space.

Returns Transformed coordinates.

Return type np.array

class brainlit.map_neurons.**CloudReg_Transform**(*vpath: str*, *Apath: str*, *direction: str = 'atlas'*)

Object that can read mat files from CloudReg, and compute transformations on points and Jacobians. Implements DiffeomorphismAction which is an interface to transform points and tangent vectors.

D(*self*, *position: np.array*, *deriv: np.array*, *order: int = 1*)

Compute transformed derivatives of mapping at given positions. Only for the non-affine component.

Parameters

- **position** (*np.array*) -- nx3 positions at which to compute derivatives.
- **deriv** (*np.array*) -- nx3 derivatives at the respective positions.
- **order** (*int, optional*) -- Order of derivative (must be 1). Defaults to 1.

Raises **ValueError** -- Only derivative order 1 is allowed here.

Returns Transformed derivatives

Return type np.array

Jacobian(*self*, *pos: np.array*)

Compute Jacobian of transformation at a given point.

Parameters **pos** (*np.array*) -- Coordinate in space.

Returns Jacobian at that coordinate

Return type np.array

apply_affine(*self*, *position: np.array*)

Apply affine transformation in the transformation of positions in target space to atlas space.

Parameters **position** (*np.array*) -- nx3 array with positions in target space.

Returns positions after affine transformation was applied.

Return type np.array

evaluate(*self*, *position: np.array*)

Apply non-affine component of mapping to positions, in direction of self.direction (default from target to atlas).

Parameters **position** (*np.array*) -- Positions at which to compute mappings.

Returns Mappings of the input.

Return type np.array

brainlit.map_neurons.**compute_derivs**(*us: np.array*, *tck: tuple = None*, *positions: np.array = None*, *deriv_method: str = 'difference'*)

Estimate derivatives of a sequence of points. Derivatives can be estimated in three ways: - For curves parameterized by scipy's spline API, spline estimation uses scipy's derivative computation - For a sequence of points, we use the finite-difference method from:

Sundqvist, H., & Veronis, G. (1970). A simple finite-difference grid with non-constant intervals. Tellus, 22(1), 26-31.

- one-sided derivatives are derived from the piecewise linear interpolation.

Parameters

- **us** (*np.array*) -- Parameter values (in form returned by scipy.interpolate.splprep).

- **tck** (*tuple*) -- Knots, bspline coefficients, and degree of spline (in form returned by scipy.interpolate.splprep).
- **positions** (*np.array*) -- nx3 array of positions (for use by difference method).
- **deriv_method** (*str, optional*) -- Method to use (from list above), spline for scipy.interpolate.splev, difference for the finite difference method, two-sided for one-sided derivatives from linear interpolation. Defaults to "difference".

Raises

- **ValueError** -- If the wrong combination of arguments/deriv_method is used.
- **ValueError** -- If derivative method is unrecognized.

Returns Derivative estimates at specified positions, or tuple of np.array for two-sided option.

Return type np.array

2.2.3 BrainLine

Data Helper Functions

Apply Ilastik and Validate Models

class brainlit.BrainLine.**ApplyIlastik**(*ilastk_path: str*, *project_path: str*, *brains_path: str*, *brains: list*)

Applies ilastik to subvolumes for the purpose of validating machine learning algorithms.

Parameters

- **ilastk_path** (*str*) -- Path to ilastik executable.
- **project_path** (*str*) -- Path to ilastik project.
- **brains_path** (*str*) -- Path to directory that contains brain samples subdirectories.
- **brains** (*list*) -- List of brain sample names.

ilastk_path
Path to ilastik executable.

Type str

project_path
Path to ilastik project.

Type str

brains_path
Path to directory that contains brain samples subdirectories.

Type str

brains
List of brain sample names.

Type list

move_results(*self*)
Move results from process_subvols to a new subfolder.

process_subvols(*self*)
Apply ilastik to all validation subvolumes of the specified brain ids in the specified directory

brainlit.BrainLine.**plot_results**(*data_dir: str*, *brain_ids: list*, *object_type: str*, *positive_channel: int*, *doubles: list = []*, *show_plot: bool = True*)

Plot precision recall curve for a specified brain.

Parameters

- **data_dir** (*str*) -- Path to directory where brain subvolumes are stored.
- **brain_id** (*str*) -- Brain id to examine (brain2paths key from _data.py file).
- **object_type** (*str*) -- soma or axon, the type of data to examine.
- **positive_channel** (*int*) -- Channel that represents neuron in the predictions.
- **doubles** (*list, optional*) -- Filenames of soma subvolumes that contain two somas, if applicable. Defaults to [].
- **show_plot** (*bool, optional*) -- Whether to run pyplot, useful for pytests when figures should not be displayed. Defaults to True.

Raises **ValueError** -- _description_

Returns Best f-score across all thresholds. float: Threshold that yields the best validation f-score.

Return type float

brainlit.BrainLine.**examine_threshold**(*data_dir: str*, *brain_id: str*, *threshold: float*, *object_type: str*, *positive_channel: int*, *doubles: list = []*, *show_plot: bool = True*)

Display results in napari of all subvolumes that were below some performance threshold, at a given threshold.

Parameters

- **data_dir** (*str*) -- Path to directory where brain subvolumes are stored.
- **brain_id** (*str*) -- Brain ID to examine (from _data.py file).
- **threshold** (*float*) -- Threshold to examine.
- **object_type** (*str*) -- soma or axon, the data type being examined.
- **positive_channel** (*int*) -- 0 or 1, Channel that represents neuron in the predictions.
- **doubles** (*list, optional*) -- Filenames of soma subvolumes that contain two somas, if applicable. Defaults to [].
- **show_plot** (*bool, optional*) -- Whether to run napari, useful for pytests when figures should not be displayed. Defaults to True.

Raises

- **ValueError** -- If object_type is neither axon nor soma
- **ValueError** -- If positive_channel is not 0 or 1.

class brainlit.BrainLine.**ApplyIlastik_LargeImage**(*ilastik_path: str*, *ilastik_project: str*, *ncpu: int*, *data_file: str*, *results_dir: Union[str, Path] = None*)

Apply ilastik to large image, where chunking is necessary.

Parameters

- **ilastk_path** (*str*) -- Path to ilastik executable.
- **ilastik_project** (*str*) -- Path to ilastik project.
- **ncpu** (*int*) -- Number of cpus to use for applying ilastik in parallel.

- **object_type** (*str*) -- Soma for soma detection or axon for axon segmentaiton.
- **results_dir** -- (str or Path): For soma detection, the directory to write detection results.

ilastk_path
Path to ilastik executable.

Type str

ilastik_project
Path to ilastik project.

Type str

ncpu
Number of cpus to use for applying ilastik in parallel.

Type int

object_type
Soma for soma detection or axon for axon segmentaiton.

Type str

results_dir
(Path): For soma detection, the directory to write detection results.

apply_ilastik_parallel(*self*, *brain_id: str*, *layer_names: list*, *threshold: float*, *data_dir: str*, *chunk_size: list*, *max_coords: list = [- 1, - 1, - 1]*, *min_coords: list = [- 1, - 1, - 1]*)
Apply ilastik to large brain, in parallel.

Parameters

- **brain_id** (*str*) -- Brain ID (key in brain2paths in _data.py file).
- **layer_names** (*list*) -- Precomputed layer names to be appended to the base path.
- **threshold** (*float*) -- Threshold for the ilastik predictor.
- **data_dir** (*str or Path*) -- Path to directory where downloaded data will be temporarily stored.
- **chunk_size** (*list*) -- Size of chunks to be used for parallel application of ilastik.
- **max_coords** (*list, optional*) -- Upper bound of bounding box on which to apply ilastk (i.e. does not apply ilastik beyond these bounds). Defaults to [-1, -1, -1].
- **min_coords** (*list, optional*) -- Lower bound of bounding box on which to apply ilastk (i.e. does not apply ilastik beyond these bounds). Defaults to [-1, -1, -1].

collect_axon_results(*self*, *brain_id: str*, *ng_layer_name: str*)
Generate neuroglancer link with the axon_mask segmentation. Intended for use after apply_ilastik_parallel

Parameters

- **brain_id** (*str*) -- ID to process.
- **ng_layer_name** (*str*) -- Name of neuroglancer layer in val_info URL with image data.

collect_soma_results(*self*, *brain_id: str*)
Combine all soma detections and post to neuroglancer. Intended for use after apply_ilastik_parallel.

Parameters **brain_id** (*str*) -- ID to process.

Results Analysis

class brainlit.BrainLine.**SomaDistribution**(*brain_ids: list*, *data_file: str*, *show_plots: bool = True*)

Object to generate various analysis images for results from a set of brain IDs. An implementation of BrainDistribution class.

Parameters

- **brain_ids** (*list*) -- List of brain IDs (keys of data json file).
- **data_files** (*str*) -- Path to json file that contains information about samples.
- **show_plots** (*bool*) -- Whether to show plots, only works if you have graphics access.

brain2paths
Information about different data samples.

Type dict

show_plots
Whether to show plots, only works if you have graphics access.

Type bool

atlas_points
Key - sample ID, Value - coordinates of soma detection.

Type dict

id_to_regioncounts
Key - sample ID, Value - dictionary of detection counts by region.

Type dict

region_graph
Graph of region hierarchy with soma detection counts as node attributes.

Type nx.DiGraph

brainrender_somas(*self*, *subtype_colors: dict*, *brain_region: str = 'DR'*)
Generate brainrender viewer with soma detections.

Parameters

- **subtype_colors** (*_type_*) -- Mapping of subtypes (in soma_data.py file) to colors for soma plotting.
- **brain_region** (*str, optional*) -- Brain region to display with the detections (e.g. dorsal raphe nucleus). Defaults to "DR".

napari_coronal_section(*self*, *z: int*, *subtype_colors: dict*, *symbols: list = ['o', '+', '^', 'vbar']*, *fold_on: bool = False*)
Generate napari view with allen atlas and points of soma detections.

Parameters

- **z** (*int*) -- index of coronal slice in Allen atlas.
- **subtype_colors** (*dict*) -- Mapping of subtypes (in soma_data.py file) to colors for soma plotting.
- **symbols** (*list*) -- Napari point symbols to use for different samples of the same subtype. Defaults to ["o", "+", "^", "vbar"].

- **fold_on** (*bool, optional*) -- Whether napari views should be a hemisphere, in which case detections from the other side are mirrored. Defaults to False.

region_barchart(*self*, *regions: list*, *composite_regions: dict = {}*, *normalize_region: int = - 1*)
Generate bar charts comparing soma detection counts between regions.

Parameters

- **regions** (*list*) -- List of Allen atlas brain region IDs to display data for (ID's found here: http://api.brain-map.org/api/v2/structure_graph_download/1.json)
- **composite_regions** (*dict, optional*) -- Mapping from a custom composite region (str, e.g. "Amygdala") to a set of regions that compose it (list of ints e.g. [131, 295, 319, 780]). Defaults to {}.
- **normalize_region** (*int, optional*) -- Region ID to normalize data for the normalized bar chart. Defaults to -1.

class brainlit.BrainLine.**AxonDistribution**(*brain_ids: list*, *data_file: str*, *regional_distribution_dir: str*, *show_plots: bool = True*)
Generates visualizations of results of axon segmentations of a set of brains. Implements BrainDistribution.

Parameters

- **brain_ids** (*list*) -- List of brain IDs (keys of data json file).
- **data_files** (*str*) -- Path to json file that contains information about samples.
- **regional_distribution_dir** (*str*) -- Path to directory with pkl files that countain segmentation results by region.
- **show_plots** (*bool*) -- Whether to show plots, only works if you have graphics access.

regional_distribution_dir
Path to directory with pkl files that countain segmentation results by region.

Type str

region_graph
Graph of region hierarchy with segmentation results as node attributes.

Type nx.DiGraph

total_axon_vols
Key - sample ID Value - Total volume of segmented axon.

Type dict

brain2paths
Information about different data samples.

Type dict

show_plots
Whether to show plots, only works if you have graphics access.

Type bool

brainrender_axons(*self*, *subtype_colors: dict*, *brain_region: str = 'DR'*)
Generate brainrender view to show axon segmentations.

Parameters **subtype_colors** (*dict*) -- Mapping of subtype to color to be used in visualization.

napari_coronal_section(*self*, *z: int*, *subtype_colors: dict*, *fold_on: bool = False*)
Generate napari viewer with allen parcellation and heat map of axon segmentations.

Parameters

- **z** (*int*) -- Index of coronal slice of allen atlas.
- **subtype_colors** (*dict*) -- Mapping of subtype to color to be used in visualization.
- **fold_on** (*bool, optional*) -- Whether to plot a single hemisphere, with results from other side mirrored. Defaults to False.

region_barchart(*self*, *regions: list*, *composite_regions: dict = {}*, *normalize_region: int = - 1*)
Generate bar charts with statistical tests to compare segmentations between brains.

Parameters

- **regions** (*list*) -- List of Allen atlas brain region IDs to display data for (ID's found here: http://api.brain-map.org/api/v2/structure_graph_download/1.json)
- **composite_regions** (*dict, optional*) -- Mapping from a custom composite region (str, e.g. "Amygdala") to a set of regions that compose it (list of ints e.g. [131, 295, 319, 780]). Defaults to {}.
- **normalize_region** (*int, optional*) -- Region ID to normalize data for the normalized bar chart. Defaults to -1.

2.2.4 Feature Extraction

class brainlit.feature_extraction.**NeighborhoodFeatures**(*url: str*, *radius: List[int] = [1, 1, 1]*, *offset: List[int] = [15, 15, 15]*, *segment_url: Optional[str] = None*)

Computes features based off neighborhood properties.

Parameters

- **url** -- Precompued path either to a file URI or url URI of image data.
- **radius** -- The radius around each point considered a neighborhood, in each dimension. If radius is [x,y,z], the neighborhood will be a [2x+1, 2y+1, 2z+1] volume centered at the point of interest. Defaults to [1, 1, 1].
- **offset** -- Added to the coordinates of a positive sample to generate a negative sample. Defaults to [15, 15, 15].
- **segment_url** -- Precompued path either to a file URI or url URI of segmentation data.

url
CloudVolumePrecomputedPath to image data.

size
A size hyperparameter. In Neighborhoods, this is the radius.

offset
Added to the coordinates of a positive sample to generate a negative sample.

download_time
Tracks time taken to download the data.

conversion_time
Tracks time taken to convert data to features.

write_time
Tracks time taken to write features to files.

segment_url
CloudVolumePrecomputedPath to segmentation data.

2.2.5 Preprocessing

Image Processing

brainlit.preprocessing.**center**(*data: np.ndarray*)
Centers data by subtracting the mean

Parameters **data** (*array-like*) -- data to be centered

Returns **data_centered** -- centered-data

Return type array-like

brainlit.preprocessing.**contrast_normalize**(*data: np.ndarray*, *centered: bool = False*)
Normalizes image data to have variance of 1

Parameters

- **data** (*array-like*) -- data to be normalized
- **centered** (*boolean*) -- When False (the default), centers the data first

Returns **data** -- normalized data

Return type array-like

brainlit.preprocessing.**whiten**(*img: np.ndarray*, *window_size: np.ndarray*, *step_size: np.ndarray*, *centered: bool = False*, *epsilon: float = 1e-05*, *type: str = 'PCA'*)
Performs PCA or ZCA whitening on an array. This preprocessing step is described in _[1].

Parameters

- **img** (*array-like*) -- image to be vectorized
- **window_size** (*array-like*) -- window size dictating the neighborhood to be vectorized, same number of dimensions as img, based on the top-left corner
- **step_size** (*array-like*) -- step size in each of direction of window, same number of dimensions as img
- **centered** (*boolean*) -- When False (the default), centers the data first
- **epsilon** (*epsilon value for whitening*) --
- **type** (*string*) -- Determines the type of whitening. Can be either 'PCA' (default) or 'ZCA'

Returns

- **data-whitened** (*array-like*) -- whitened data
- **S** (*2D array*) -- Singular value array of covariance of vectorized image

References

brainlit.preprocessing.**window_pad**(*img: np.ndarray*, *window_size: np.ndarray*, *step_size: np.ndarray*)

Pad image at edges so the window can convolve evenly. Padding will be a copy of the edges.

Parameters

- **img** (*array-like*) -- image to be padded
- **window_size** (*array-like*) -- window size that will be convolved, same number of dimensions as img
- **step_size** (*array-like*) -- step size in each of direction of window convolution, same number of dimensions as img

Returns

- **img_padded** (*array-like*) -- padded image
- **pad_size** (*array-like*) -- amount of padding in every direction of the image

brainlit.preprocessing.**undo_pad**(*img: np.ndarray*, *pad_size: np.ndarray*)

Remove padding from edges of images

Parameters

- **img** (*array-like*) -- padded image
- **pad_size** (*array-like*) -- amount of padding in every direction of the image

Returns **img** -- unpadded image

Return type array-like

brainlit.preprocessing.**vectorize_img**(*img: np.ndarray*, *window_size: np.ndarray*, *step_size: np.ndarray*)

Reshapes an image by vectorizing different neighborhoods of the image.

Parameters

- **img** (*array-like*) -- image to be vectorized
- **window_size** (*array-like*) -- window size dictating the neighborhood to be vectorized, same number of dimensions as img, based on the top-left corner
- **step_size** (*array-like*) -- step size in each of direction of window, same number of dimensions as img

Returns **vectorized** -- vectorized image

Return type array-like

brainlit.preprocessing.**imagize_vector**(*img: np.ndarray*, *orig_shape: tuple*, *window_size: np.ndarray*, *step_size: np.ndarray*)

Reshapes a vectorized image back to its original shape.

Parameters

- **img** (*array-like*) -- vectorized image
- **orig_shape** (*tuple*) -- dimensions of original image
- **window_size** (*array-like*) -- window size dictating the neighborhood to be vectorized, same number of dimensions as img, based on the top-left corner

- **step_size** (*array-like*) -- step size in each of direction of window, same number of dimensions as img

Returns **imagized** -- original image

Return type array-like

Image Filters

brainlit.preprocessing.**gabor_filter**(*input: np.ndarray*, *sigma: Union[float, List[float]]*, *phi: Union[float, List[float]]*, *frequency: float*, *offset: float = 0.0*, *output: Optional[Union[np.ndarray, np.dtype, None]] = None*, *mode: str = 'reflect'*, *cval: float = 0.0*, *truncate: float = 4.0*)

Multidimensional Gabor filter. A gabor filter is an elementwise product between a Gaussian and a complex exponential.

Parameters

- **input** (*array_like*) -- The input array.
- **sigma** (*scalar or sequence of scalars*) -- Standard deviation for Gaussian kernel. The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.
- **phi** (*scalar or sequence of scalars*) -- Angles specifying orientation of the periodic complex exponential. If the input is n-dimensional, then phi is a sequence of length n-1. Convention follows https://en.wikipedia.org/wiki/N-sphere#Spherical_coordinates.
- **frequency** (*scalar*) -- Frequency of the complex exponential. Units are revolutions/voxels.
- **offset** (*scalar*) -- Phase shift of the complex exponential. Units are radians.
- **output** (*array or dtype, optional*) -- The array in which to place the output, or the dtype of the returned array. By default an array of the same dtype as input will be created. Only the real component will be saved if output is an array.
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) -- The mode parameter determines how the input array is extended beyond its boundaries. Default is 'reflect'.
- **cval** (*scalar, optional*) -- Value to fill past edges of input if mode is 'constant'. Default is 0.0.
- **truncate** (*float*) -- Truncate the filter at this many standard deviations. Default is 4.0.

Returns **real, imaginary** -- Returns real and imaginary responses, arrays of same shape as *input*.

Return type arrays

Notes

The multidimensional filter is implemented by creating a gabor filter array, then using the convolve method. Also, sigma specifies the standard deviations of the Gaussian along the coordinate axes, and the Gaussian is not rotated. This is unlike skimage.filters.gabor, whose Gaussian is rotated with the complex exponential. The reasoning behind this design choice is that sigma can be more easily designed to deal with anisotropic voxels.

Examples

```
>>> from brainlit.preprocessing import gabor_filter
>>> a = np.arange(50, step=2).reshape((5,5))
>>> a
array([[ 0,  2,  4,  6,  8],
       [10, 12, 14, 16, 18],
       [20, 22, 24, 26, 28],
       [30, 32, 34, 36, 38],
       [40, 42, 44, 46, 48]])
>>> gabor_filter(a, sigma=1, phi=[0.0], frequency=0.1)
(array([[ 3,  5,  6,  8,  9],
        [ 9, 10, 12, 13, 14],
        [16, 18, 19, 21, 22],
        [24, 25, 27, 28, 30],
        [29, 30, 32, 34, 35]]),
 array([[ 0,  0, -1,  0,  0],
        [ 0,  0, -1,  0,  0],
        [ 0,  0, -1,  0,  0],
        [ 0,  0, -1,  0,  0],
        [ 0,  0, -1,  0,  0]]))
```

```
>>> from scipy import misc
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> plt.gray()  # show the filtered result in grayscale
>>> ax1 = fig.add_subplot(121)  # left side
>>> ax2 = fig.add_subplot(122)  # right side
>>> ascent = misc.ascent()
>>> result = gabor_filter(ascent, sigma=5, phi=[0.0], frequency=0.1)
>>> ax1.imshow(ascent)
>>> ax2.imshow(result[0])
>>> plt.show()
```

Segmentation Processing

`brainlit.preprocessing.`**`getLargestCC`**(*segmentation: np.ndarray*)

Returns the largest connected component of a image.

Arguments: segmentation : Segmentation data of image or volume.

Returns: largeCC : Segmentation with only largest connected component.

`brainlit.preprocessing.`**`removeSmallCCs`**(*segmentation: np.ndarray*, *size: Union[int, float]*, *verbose=False*)

Removes small connected components from an image.

Parameters: segmentation : Segmentation data of image or volume. size : Maximum connected component size to remove.

Returns: largeCCs : Segmentation with small connected components removed.

brainlit.preprocessing.**label_points**(*labels: np.array*, *points: list*, *res: list*)
Adjust points so they fall on a foreground component of labels.

Parameters

- **labels** (*array*) -- labeled components, such as output from measure.label
- **points** (*list*) -- points to be adjusted
- **res** (*list*) -- voxel size

Returns adjusted points [list]: labels of adjusted points

Return type [list]

brainlit.preprocessing.**compute_frags**(*soma_coords: list*, *labels: np.array*, *im_processed: np.array*, *threshold: float*, *res: list*, *chunk_size: list = None*, *ncpu: int = 2*)
Preprocesses a neuron image segmentation by splitting up non-soma components into 5 micron segments.

Parameters

- **soma_coords** (*list*) -- list of voxel coordinates of somas
- **labels** (*np.array*) -- image segmentation
- **im_processed** (*np.array*) -- voxel-wise probability predictions for foreground
- **threshold** (*float*) -- threshold used to segment probability predictions into mask
- **res** (*list*) -- voxel size in image
- **chunk_size** (*list*) -- size of image chunks
- **ncpu** (*int*) -- number of cpus to use in parallel mode

Returns new image segmentation - different numbers indicate different fragments, 0 is background

Return type np.array

brainlit.preprocessing.**remove_somas**(*soma_coords: list*, *labels: np.array*, *im_processed: np.array*, *res: list*, *verbose=False*)
Helper function of split_frags. Removes area around somas.

Parameters

- **soma_coords** (*list*) -- list of voxel coordinates of somas
- **labels** (*np.array*) -- image segmentation
- **im_processed** (*np.array*) -- voxel-wise probability predictions for foreground
- **res** (*list*) -- voxel size in image

Returns probability predictions, with the soma regions masked list: coordinates of the points dictionary: map from component in labels, to set of points that were placed there list: masks of the different somas

Return type np.array

brainlit.preprocessing.**rename_states_consecutively**(*new_labels: np.array*)
Helper function of split_frags. Relabel components in image segmentation so the unique values are consecutive.

Parameters **new_labels** (*np.array*) -- new image segmentation - different numbers indicate different fragments, 0 is background

Returns new image segmentation - different numbers indicate different fragments, 0 is background

Return type np.array

2.2.6 Utility

Data Helper Methods

class brainlit.utils.**NeuroglancerSession**(*url: str*, *mip: int = 0*, *url_segments: Optional[str] = None*, *fill_missing: bool = True*, *use_https: bool = False*)

Utility class which pulls and pushes data.

Parameters

- **url** -- Precompued path either to a file URI or url URI. Defaults to mouselight brain1.
- **mip** -- Resolution level to pull and push data at. Defaults to 0, the highest resolution.
- **url_segments** -- Precomputed path to segmentation data. Optional, default None.
- **fill_missing** -- Always passes directly into 'CloudVolume()' function to fill missing segent/image values with 0s.
- **use_https** -- Always passes directly into 'CloudVolume()' function to set use_https to the desired value.

url
CloudVolumePrecomputedPath to image data.

url_segments
CloudVolumePrecomputedPath to segmentation data. Optional, default None. Automatically tries precomputed path url+"_segments" if None.

cv
CloudVolume object for image data.

Type CloudVolumePrecomputed

cv_segments
CloudVolume object for segmentation data. Optional, default None.

Type CloudVolumePrecomputed

cv_annotations
CloudVolume object for segmentation data. Optional, default None.

Type CloudVolumePrecomputed

mip
Resolution level.

chunk_size
The chunk size of the volume at the specified mip, given as (x, y, z).

scales
The resolution of the volume at the specified mip, given as (x, y, z).

create_tubes(*self*, *seg_id: Union[int, float]*, *bbox: Bounds*, *radius: Optional[int] = None*)
Creates voxel-wise foreground/background labels associated with a particular neuron trace, within a given bounding box of voxel coordinates.

Parameters

- **seg_id** -- The id of the .swc file.

- **bbox** -- The bounding box to draw tubes within.
- **radius** -- Euclidean distance threshold used to draw tubes, default None = 1 px thick.
- **rounding** -- Optional, bool, default is True. False if no swc rounding.

Returns A volume within the bounding box, with 1 on tubes and 0 elsewhere.

Return type labels

get_segments(*self*, *seg_id: int*, *bbox: Optional[Bounds] = None*, *rounding: Optional[bool] = True*)
Get a graph of a segmentation annotation within a bounding box.

Parameters

- **The segement to pull.** (*seg_id*) --
- **bbox** -- The bounding box object, default None. If None, uses entire volume.
- **rounding** -- Optional, default True. Whether you want S3 file to be rounded or not.

Returns A networkx subgraph from the specified segment and bounding box.

Return type G

pull_bounds_img(*self*, *bounds: Bounds*)
Pull a volume around a specified bounding box. Works on image channels.

Parameters **bounds** -- Bounding box, or tuple containing (x0, y0, z0, x1, y1, z1) bounds.

Returns Volume pulled according to the bounding box.

Return type img

abstract pull_bounds_seg(*self*, *bounds: Bounds*)
Pull a volume around a specified bounding box. Works on annotation channels.

Parameters **bounds** -- Bounding box, or tuple containing (x0, y0, z0, x1, y1, z1) bounds.

Returns Volume pulled according to the bounding box.

Return type img

pull_chunk(*self*, *seg_id: int*, *v_id: int*, *radius: int = 0*)
Pull a subvolume around a specified skeleton vertex according to chunk size. Each data set has a specified chunk size, which can be found by calling self.cv.info.

Parameters

- **seg_id** -- ID of the segment to use, depends on data in s3.
- **v_id** -- ID of the vertex to use, depends on the segment.
- **radius** -- Radius of pulled volume around central chunk, in chunks. Optional, default is 0 (single chunk which contains the voxel).

Returns A chunk_size[0]*2*nx X chunk_size[1]*2*ny X chunk_size[2]*2*nz volume. bounds: Bounding box object which contains the bounds of the volume. vox_in_img: List of coordinates which locate the initial point in the volume.

Return type img

pull_vertex_list(*self*, *seg_id: int*, *v_id_list: List[int]*, *buffer: List[int] = [1, 1, 1]*, *expand: bool = False*)
Pull a subvolume containing all listed vertices.

Parameters

- **seg_id** -- ID of the segment to use, depends on data in s3.

- **v_id_list** -- list of vertex IDs to use.
- **buffer** -- Buffer around the bounding box (in voxels). Can be int or list of ints. Default [1, 1, 1], set to [0, 0, 0] if expand is True.
- **expand** -- Flag whether to expand subvolume to closest set of chunks.

Returns The image volume containing all vertices. bounds: Bounding box object which contains the bounds of the volume. vox_in_img_list: List of coordinates which locate the vertices in the volume.

Return type img

pull_voxel (*self*, *seg_id: int*, *v_id: int*, *radius: int = 1*)
Pull a subvolume around a specified skeleton vertex with of shape [2r+1, 2r+1, 2r+1], in voxels.

Parameters

- **seg_id** -- ID of the segment to use, depends on data in s3.
- **v_id** -- ID of the vertex to use, depends on the segment.
- **radius** -- Radius of pulled volume around central voxel, in voxels. Optional, default is 1 (3x3 volume is pulled, centered at the vertex).

Returns A 2*nx+1 X 2*ny+1 X 2*nz+1 volume. bounds: Bounding box object which contains the bounds of the volume. vox_in_img: List of coordinates which locate the initial point in the subvolume.

Return type img

abstract push (*self*, *img: np.ndarray*, *bounds: Bounds*)
Push a volume to an annotation channel.

Parameters

- **img** -- Volume to push
- **bounds** -- Bounding box or tuple containing (x0, y0, z0, x1, y1, z1) bounds.

set_url_segments (*self*, *seg_url: str*)
Sets the url_segments and cv_segments attributes.

Parameters **seg_url** -- CloudvolumePrecomputedPath to segmentation data.

class brainlit.utils.**NeuronTrace** (*path: str*, *seg_id: int = None*, *mip: int = None*, *rounding: bool = True*, *read_offset: bool = False*, *fill_missing: bool = True*, *use_https: bool = False*)
Neuron Trace class to handle neuron traces as swcs and s3 skeletons

Parameters

- **path** (*str*) -- Path to either s3 bucket (url) or swc file (filepath).
- **seg_id** (*int*) -- If s3 bucket path is provided, the segment number to pull, default None.
- **mip** (*int*) -- If s3 bucket path is provided, the resolution to use for scaling, default None.
- **rounding** (*bool*) -- If s3 is provided, specifies if it should be rounded, default True
- **read_offset** (*bool*) -- If swc is provided, whether offset should be read from file, default False.
- **fill_missing** (*bool*) -- Always passes directly into 'CloudVolume()' function to fill missing skeleton values with 0s, default True.

- **use_https** (*bool*) -- Always passes directly into 'CloudVolume()' function to set use_https to desired value, default True.

path
Path to either s3 bucket (url) or swc file (filepath)

Type str

input_type
Specifies whether input file is 'swc' or 'skel'

Type bool

df
Indices, coordinates, and parents of each node

Type pandas.DataFrame

args
Stores arguments for df - offset, color, cc, branch

Type tuple

seg_id
If s3 bucket path is provided, the segment number to pull

Type int

mip
If s3 bucket path is provided, the resolution to use for scaling

Type None,int

Example

```
>>> swc_path = "./data/data_octree/consensus-swcs/2018-08-01_G-002_consensus.swc"
>>> s3_path = "s3://open-neurodata/brainlit/brain1_segments"
>>> seg_id = 11
>>> mip = 2
```

```
>>> swc_trace = NeuronTrace(swc_path)
>>> s3_trace = NeuronTrace(s3_path,seg_id,mip)
```

generate_df_subset(*self*, *vox_in_img_list: list*, *subneuron_start: int = None*, *subneuron_end: int = None*)
Read a new subset dataframe in coordinates in img spacing. Specify specific range of vertices from dataframe if desired

Parameters

- **vox_in_img_list** (*list*) -- List of voxels
- **subneuron_start** (*None, int (default = None)*) -- Provides start index, if specified, to apply function to a portion of the dataframe Default is None.
- **subneuron_end** (*None, int (default = None)*) -- Provides end index, if specified, to apply function to a portion of the dataframe Default is None.

Returns **df** -- Indicies, coordinates (in img spacing) and parents of each node. Coordinates are in spatial units.

Return type pandas.DataFrame

Example

```
>>> #swc input, subneuron_start and subneuron_end specified
```

```
>>> subneuron_start = 5
>>> subneuron_end = 8
```

```
>>> #generate vox_in_img_list
>>> my_list = []
>>>for i in range(subneuron_end-subneuron_start):
    my_list.append(10)
>>> vox_in_img_list_2 = list([my_list,my_list,my_list])
```

>>>swc_trace.generate_df_subset(vox_in_img_list_2,subneuron_start,subneuron_end)

```
>>> sample    structure    x    y    z    r    parent
        5    6    0    10    10    10    1.0    5
        6    7    0    10    10    10    1.0    6
        7    8    0    10    10    10    1.0    7
```

get_bfs_subgraph(*self*, *node_id: int*, *depth: int*, *df: pd.DataFrame = None*, *spacing: np.array = None*, *origin: np.array = None*)

Creates a spanning subgraph from a seed node and parent graph using BFS.

Returns

G_sub [networkx.classes.digraph.DiGraph]

Subgraph

tree [DiGraph] The tree returned by BFS.

paths [list] List of Nx3 numpy.array. Rows of the array are 3D coordinates in voxel units. Each array is one path.

Example ..

```
>>> #swc input, specify node_id and depth
    ..
```

```
>>> swc_trace.get_bfs_subgraph(node_id=11,depth=2)
    ..
```

>>>(<networkx.classes.digraph.DiGraph at 0x7f7f2ce65670>,
<networkx.classes.digraph.DiGraph at 0x7f7f2ce65370>, array([array([[4727, 4440, 3849],

[4732, 4442, 3850], [4739, 4455, 3849]]), array([[4732, 4442, 3850], [4749, 4439, 3856]])], dtype=object))

get_df(*self*)

Gets the dataframe providing indices, coordinates, and parents of each node

Returns **self.df** -- dataframe providing indices, coordinates, and parents of each node

Return type pandas.DataFrame

Example

```
>>> swc_trace.get_df()
>>> sample    structure    x    y    z    r    parent
    0    1    0    -52.589700    -1.448032    -1.228827    1.0    -1
    1    2    0    -52.290940    -1.448032    -1.228827    1.0    1
    2    3    0    -51.992181    -1.143616    -0.240423    1.0    2
    3    4    0    -51.095903    -1.143616    -0.240423    1.0    3
    4    5    0    -50.797144    -0.839201    -0.240423    1.0    4
    ...    ...    ...    ...    ...    ...    ...    ...
    148    149    0    45.702088    14.381594    -7.159252    1.0    148
    149    150    0    46.000847    14.686010    -7.159252    1.0    149
    150    151    0    46.897125    14.686010    -7.159252    1.0    150
    151    152    0    47.494643    15.294842    -7.159252    1.0    151
    152    153    6    48.092162    15.294842    -7.159252    1.0    152
    53 rows × 7 columns
```

get_df_arguments (*self*)
Gets arguments for df - offset, color, cc, branch

Returns **self.args** -- list of arguments for df, if found - offset, color, cc, branch

Return type list

Example

```
>>> swc_trace.get_df_arguments()
>>> [[73954.8686, 17489.532566, 34340.365689], [1.0, 1.0, 1.0], nan, nan]
```

get_df_voxel (*self*, *spacing: np.array*, *origin: np.array = np.array([0, 0, 0])*)
Converts coordinates in pd.DataFrame from spatial units to voxel units

Parameters

- **spacing** (`numpy.array`) -- Conversion factor (spatial units/voxel). Assumed to be np.array([x,y,z])
- **origin** (`numpy.array`) -- Origin of the spatial coordinate. Default is (0,0,0). Assumed to be np.array([x,y,z])

Returns **df_voxel** -- Indicies, coordinates, and parents of each node in the swc. Coordinates are in voxel units.

Return type `pandas.DataFrame`

Example

```
>>> swc_trace.get_df_voxel(spacing=np.asarray([2,2,2]))
>>> sample    structure    x    y    z    r    parent
    0    1    0    -26    -1    -1    1.0    -1
    1    2    0    -26    -1    -1    1.0    1
    2    3    0    -26    -1    0    1.0    2
    3    4    0    -26    -1    0    1.0    3
    4    5    0    -25    0    0    1.0    4
    ...    ...    ...    ...    ...    ...    ...    ...
    148    149    0    23    7    -4    1.0    148
```

(continues on next page)

(continued from previous page)

```
    149    150    0    23    7    -4    1.0    149
    150    151    0    23    7    -4    1.0    150
    151    152    0    24    8    -4    1.0    151
    152    153    6    24    8    -4    1.0    152
    153 rows × 7 columns
```

get_graph (*self*, *spacing: np.array = None*, *origin: np.array = None*)
Converts dataframe in either spatial or voxel coordinates into a directed graph. Will convert to voxel coordinates if spacing is specified.

Parameters

- **spacing** (None, `numpy.array` (default = None)) -- Conversion factor (spatial units/voxel). Assumed to be np.array([x,y,z]). Provided if graph should convert to voxel coordinates first. Default is None.
- **origin** (None, `numpy.array` (default = None)) -- Origin of the spatial coordinate, if converting to voxels. Default is None. Assumed to be np.array([x,y,z])

Returns **G** -- Neuron from swc represented as directed graph. Coordinates x,y,z are node attributes accessed by keys 'x','y','z' respectively.

Return type `networkx.classes.digraph.DiGraph`

Example

```
>>> swc_trace.get_graph()
>>> <networkx.classes.digraph.DiGraph at 0x7f81a83937f0>
```

get_paths (*self*, *spacing: np.array = None*, *origin: np.array = None*)
Converts dataframe in either spatial or voxel coordinates into a list of paths. Will convert to voxel coordinates if spacing is specified.

Parameters

- **spacing** (None, `numpy.array` (default = None)) -- Conversion factor (spatial units/voxel). Assumed to be np.array([x,y,z]). Provided if graph should convert to voxel coordinates first. Default is None.
- **origin** (None, `numpy.array`) -- Origin of the spatial coordinate, if converting to voxels. Default is None. Assumed to be np.array([x,y,z])

Returns **paths** -- List of Nx3 numpy.array. Rows of the array are 3D coordinates in voxel units. Each array is one path.

Return type list

Example

```
>>> swc_trace.get_paths()[0][1:10]
>>> array([[-52, -1, -1],
           [-51, -1, 0],
           [-51, -1, 0],
           [-50, 0, 0],
           [-50, 0, 0],
           [-49, 0, 0],
           [-48, 0, 0],
           [-46, 0, 0],
           [-46, 0, 0]], dtype=object)
```

get_skel (*self*, *benchmarking: bool = False*, *origin: np.ndarray = None*)
Gets a skeleton version of dataframe, if swc input is provided

Parameters

- **origin** (*None, numpy array with shape (3,1) (default = None)*) -- origin of coordinate frame in microns, (default: None assumes (0,0,0) origin)
- **benchmarking** (*bool*) -- For swc files, specifies whether swc file is from benchmarking dataset, to obtain skeleton ID

Returns **skel** -- Skeleton object of given SWC file

Return type cloudvolume.Skeleton

Example

```
>>> swc_trace.get_skel(benchmarking=True)
>>> Skeleton(segid=, vertices=(shape=153, float32), edges=(shape=152, uint32),
↪ radius=(153, float32), vertex_types=(153, uint8), vertex_color=(153,␣
↪float32), space='physical' transform=[[1.0, 0.0, 0.0, 0.0], [0.0, 1.0, 0.0,␣
↪0.0], [0.0, 0.0, 1.0, 0.0]])
```

get_sub_neuron (*self*, *bounding_box: Union[tuple, list, None]*, *spacing: np.array = None*, *origin: np.array = None*)
Returns sub-neuron with node coordinates bounded by start and end

Parameters

- **bounding_box** (*tuple or list or None*) -- Defines a bounding box around a sub-region around the neuron. Length 2 tuple/list. First element is the coordinate of one corner (inclusive) and second element is the coordinate of the opposite corner (exclusive). Both coordinates are numpy.array([x,y,z])in voxel units.
- **spacing** (None, `numpy.array` (default = None)) -- Conversion factor (spatial units/voxel). Assumed to be np.array([x,y,z]). Provided if graph should convert to voxel coordinates first. Default is None.
- **origin** (`numpy.array`) -- Origin of the spatial coordinate, if converting to voxels. Default is None. Assumed to be np.array([x,y,z])

Returns **G_sub** -- Neuron from swc represented as directed graph. Coordinates x,y,z are node attributes accessed by keys 'x','y','z' respectively.

Return type `networkx.classes.digraph.DiGraph`

Example

```
>>> bounding_box=[[1,2,4],[1,2,3]]
```

```
>>> #swc input, no spacing and origin
>>> swc_trace.get_sub_neuron(bounding_box)
>>> <networkx.classes.digraph.DiGraph at 0x7f81a95d1e50>
```

get_sub_neuron_paths(*self*, *bounding_box: Union[tuple, list, None]*, *spacing: np.array = None*, *origin: np.array = None*)

Returns sub-neuron with node coordinates bounded by start and end

Parameters

- **bounding_box** (*tuple or list or None*) -- Defines a bounding box around a sub-region around the neuron. Length 2 tuple/list. First element is the coordinate of one corner (inclusive) and second element is the coordinate of the opposite corner (exclusive). Both coordinates are numpy.array([x,y,z])in voxel units.
- **spacing** (None, `numpy.array` (default = None)) -- Conversion factor (spatial units/voxel). Assumed to be np.array([x,y,z]). Provided if graph should convert to voxel coordinates first. Default is None.
- **origin** (`numpy.array`) -- Origin of the spatial coordinate, if converting to voxels. Default is None. Assumed to be np.array([x,y,z])

Returns **paths** -- List of Nx3 numpy.array. Rows of the array are 3D coordinates in voxel units. Each array is one path.

Return type list

Example

```
>>> bounding_box=[[1,2,4],[1,2,3]]
```

```
>>> #swc input, no spacing and origin
>>> swc_trace.get_sub_neuron_paths(bounding_box)
>>> array([], dtype=object)
```

static ssd(*pts1: np.array*, *pts2: np.array*)

Compute significant spatial distance metric between two traces as defined in APP1. :param pts1: array containing coordinates of points of trace 1. shape: npoints x ndims :type pts1: np.array :param pts2: array containing coordinates of points of trace 1. shape: npoints x ndims :type pts2: np.array

Returns significant spatial distance as defined by APP1

Return type [float]

Example

```
>>> pts1 = swc_trace.get_paths()[0][1:10]
>> pts2 = swc_trace.get_paths()[0][11:20]
```

```
>>> NeuronTrace.ssd(pts1,pts2)
```

>>>6.247937554557103

brainlit.utils.**czi_to_zarr**(*czi_path: str*, *out_dir: str*, *fg_channel: int = 0*, *parallel: int = 1*)

Convert 4D czi image to a zarr file(s) at a given directory. Single channel image will produce a single zarr, two channels will produce two.

Parameters

- **czi_path** (*str*) -- Path to czi image.
- **out_dir** (*str*) -- Path to directory where zarr(s) will be written.
- **fg_channel** (*int*) -- Index of foreground channel.
- **parallel** (*int*) -- Number of cpus to use to write zarr.

Returns paths to zarrs that were written

Return type list

brainlit.utils.**zarr_to_omezarr**(*zarr_path: str*, *out_path: str*, *res: list*)

Convert 3D zarr to ome-zarr.

Parameters

- **zarr_path** (*str*) -- Path to zarr.
- **out_path** (*str*) -- Path of ome-zarr to be created.
- **res** (*list*) -- List of xyz resolution values in nanometers.

Raises

- **ValueError** -- If zarr to be written already exists.
- **ValueError** -- If conversion is not 3D array.

S3 Helper Methods

brainlit.utils.**get_data_ranges**(*bin_path: List[List[str]]*, *chunk_size: Tuple[int, int, int]*)

Get ranges (x,y,z) for chunks to be stitched together in volume

Parameters

- **bin_path** -- Binary paths to files.
- **chunk_size** -- The size of chunk to get ranges over.

Returns x-coord int bounds. y_range: y-coord int bounds. z_range: z-coord int bounds.

Return type x_range

2.2.7 Visualization

Trace Visualization

brainlit.viz.**snap_points**(*img: np.ndarray*, *points: pd.DataFrame*, *radius: list = [3, 3, 3]*)
Moves neuron marker points to the highest intensity within a certain radius

Parameters

- **{3d array} -- image** (*img*) --
- **{pandas dataframe} -- dataframe with swc points as output by combine_swc_img.points2voxel** (*points*) --

Keyword Arguments **{list} -- voxel radius within which to search for highest intensity (default** (*radius*) -- {[3,3,3]})

Returns x,y,z, columns are unchanged)

Return type [pandas dataframe] -- dataframe with same format as points, with new xvox, yvox, zvox values (Note

brainlit.viz.**point_threshold**(*img: np.ndarray*, *points: pd.DataFrame*)
Threshold image according to the minimum intensity of a set of points

Parameters

- **{3d array} -- image** (*img*) --
- **{pandas dataframe} -- dataframe with swc points as output by combine_swc_img.points2voxel** (*points*) --

Returns [3d array] -- binary mask from thresholding [int] -- threshold value

brainlit.viz.**skeletonize**(*img: np.ndarray*, *points: pd.DataFrame*)
Draw lines between points that are connected to produce binary mask

Parameters

- **{3d array} -- image** (*img*) --
- **{pandas dataframe} -- dataframe with swc points as output by combine_swc_img.points2voxel** (*points*) --

Returns [3d array] -- binary mask showing skeletonization between points

brainlit.viz.**skeleton_threshold_intersect**(*img: np.ndarray*, *points: pd.DataFrame*)
Compute intersection between two masks: thresholded image and skeletonization of points

Parameters

- **{3d array} -- image** (*img*) --
- **{pandas dataframe} -- dataframe with swc points as output by combine_swc_img.points2voxel** (*points*) --

Returns [3d array] -- binary mask of intersection [int] -- when the threshold is lowered to obtain a single connected component, this indicates the number of iterations used

brainlit.viz.**Bresenham3D**(*x1: int*, *y1: int*, *z1: int*, *x2: int*, *y2: int*, *z2: int*)
Takes two coordinates and gives the set of coordinates that connects them with a straight line

Adapted from https://www.geeksforgeeks.org/bresenhams-algorithm-for-3-d-line-drawing/

Parameters

- **{int} -- first x coodinate** (*x1*) --
- **{int} -- first y coodinate** (*y1*) --
- **{int} -- first z coodinate** (*z1*) --
- **{int} -- second x coodinate** (*x2*) --
- **{int} -- second y coodinate** (*y2*) --
- **{int} -- second z coodinate** (*z2*) --

Returns [list] -- list of x coordinate connecting the points [list] -- list of y coordinate connecting the points [list] -- list of z coordinate connecting the points

2.3 Napari Plugin

2.3.1 ViterBrain Napari Plugin Installation

- First, install `brainlit` [Documentation] (you may need to install from source with *pip install -e .*, since our pypi version may not reflect the latest changes in the repo).
- Second, install napari.
- The *Plugins* tab of napari should automatically find the brainlit plugin (Documentation).

2.3.2 How to Use the ViterBrain Napari Plugin

- Build a ViterBrain object according to an image and some voxel-wise predictions. loading.py can be used for this with sample data found in experiments/ViterBrain/data/sample.zip.
- Make sure you are using Python3.9 to run loading.py
- Open the ViterBrain object in napari.
- Launch the ViterBrain plugin widget.
- Select the labels layer and hover over fragments with your cursor to identify fragment ID numbers in the bottom left of the napari window. Identify the desired start and end fragment and enter the ID's in the widget box.
- Click trace and the plugin should generate a new path layer that shows the trace between the two fragments.

2.4 License

Brainlit is distributed with Apache 2.0 license.

```
Apache License
                        Version 2.0, January 2004
                        http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

    "License" shall mean the terms and conditions for use, reproduction,
    and distribution as defined by Sections 1 through 9 of this document.
```

(continues on next page)

(continued from previous page)

```
    "Licensor" shall mean the copyright owner or entity authorized by
    the copyright owner that is granting the License.

    "Legal Entity" shall mean the union of the acting entity and all
    other entities that control, are controlled by, or are under common
    control with that entity. For the purposes of this definition,
    "control" means (i) the power, direct or indirect, to cause the
    direction or management of such entity, whether by contract or
    otherwise, or (ii) ownership of fifty percent (50%) or more of the
    outstanding shares, or (iii) beneficial ownership of such entity.

    "You" (or "Your") shall mean an individual or Legal Entity
    exercising permissions granted by this License.

    "Source" form shall mean the preferred form for making modifications,
    including but not limited to software source code, documentation
    source, and configuration files.

    "Object" form shall mean any form resulting from mechanical
    transformation or translation of a Source form, including but
    not limited to compiled object code, generated documentation,
    and conversions to other media types.

    "Work" shall mean the work of authorship, whether in Source or
    Object form, made available under the License, as indicated by a
    copyright notice that is included in or attached to the work
    (an example is provided in the Appendix below).

    "Derivative Works" shall mean any work, whether in Source or Object
    form, that is based on (or derived from) the Work and for which the
    editorial revisions, annotations, elaborations, or other modifications
    represent, as a whole, an original work of authorship. For the purposes
    of this License, Derivative Works shall not include works that remain
    separable from, or merely link (or bind by name) to the interfaces of,
    the Work and Derivative Works thereof.

    "Contribution" shall mean any work of authorship, including
    the original version of the Work and any modifications or additions
    to that Work or Derivative Works thereof, that is intentionally
    submitted to Licensor for inclusion in the Work by the copyright owner
    or by an individual or Legal Entity authorized to submit on behalf of
    the copyright owner. For the purposes of this definition, "submitted"
    means any form of electronic, verbal, or written communication sent
    to the Licensor or its representatives, including but not limited to
    communication on electronic mailing lists, source code control systems,
    and issue tracking systems that are managed by, or on behalf of, the
    Licensor for the purpose of discussing and improving the Work, but
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as "Not a Contribution."

    "Contributor" shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
```

(continues on next page)

(continued from previous page)

```
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    copyright license to reproduce, prepare Derivative Works of,
    publicly display, publicly perform, sublicense, and distribute the
    Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    (except as stated in this section) patent license to make, have made,
    use, offer to sell, sell, import, and otherwise transfer the Work,
    where such license applies only to those patent claims licensable
    by such Contributor that are necessarily infringed by their
    Contribution(s) alone or by combination of their Contribution(s)
    with the Work to which such Contribution(s) was submitted. If You
    institute patent litigation against any entity (including a
    cross-claim or counterclaim in a lawsuit) alleging that the Work
    or a Contribution incorporated within the Work constitutes direct
    or contributory patent infringement, then any patent licenses
    granted to You under this License for that Work shall terminate
    as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
    Work or Derivative Works thereof in any medium, with or without
    modifications, and in Source or Object form, provided that You
    meet the following conditions:

    (a) You must give any other recipients of the Work or
        Derivative Works a copy of this License; and

    (b) You must cause any modified files to carry prominent notices
        stating that You changed the files; and

    (c) You must retain, in the Source form of any Derivative Works
        that You distribute, all copyright, patent, trademark, and
        attribution notices from the Source form of the Work,
        excluding those notices that do not pertain to any part of
        the Derivative Works; and

    (d) If the Work includes a "NOTICE" text file as part of its
        distribution, then any Derivative Works that You distribute must
        include a readable copy of the attribution notices contained
        within such NOTICE file, excluding those notices that do not
        pertain to any part of the Derivative Works, in at least one
        of the following places: within a NOTICE text file distributed
        as part of the Derivative Works; within the Source form or
        documentation, if provided along with the Derivative Works; or,
        within a display generated by the Derivative Works, if and
        wherever such third-party notices normally appear. The contents
        of the NOTICE file are for informational purposes only and
        do not modify the License. You may add Your own attribution
        notices within Derivative Works that You distribute, alongside
        or as an addendum to the NOTICE text from the Work, provided
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
```

(continues on next page)

(continued from previous page)

```
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
    any Contribution intentionally submitted for inclusion in the Work
    by You to the Licensor shall be under the terms and conditions of
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or modify
    the terms of any separate license agreement you may have executed
    with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
    names, trademarks, service marks, or product names of the Licensor,
    except as required for reasonable and customary use in describing the
    origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
    result of this License or out of the use or inability to use the
    Work (including but not limited to damages for loss of goodwill,
    work stoppage, computer failure or malfunction, or any and all
    other commercial damages or losses), even if such Contributor
    has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
    the Work or Derivative Works thereof, You may choose to offer,
    and charge a fee for, acceptance of support, warranty, indemnity,
    or other liability obligations and/or rights consistent with this
    License. However, in accepting such obligations, You may act only
    on Your own behalf and on Your sole responsibility, not on behalf
    of any other Contributor, and only if You agree to indemnify,
    defend, and hold each Contributor harmless for any liability
    incurred by, or claims asserted against, such Contributor by reason
    of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

    To apply the Apache License to your work, attach the following
    boilerplate notice, with the fields enclosed by brackets "[]"
```

(continues on next page)

(continued from previous page)

```
    replaced with your own identifying information. (Don't include
    the brackets!)  The text should be enclosed in the appropriate
    comment syntax for the file format. We also recommend that a
    file or class name and description of purpose be included on the
    same "printed page" as the copyright notice for easier
    identification within third-party archives.

Copyright [2018] [Jaewon Chung, Benjamin Pedigo, Eric Bridgeford]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

2.5 Fitting Splines to Axonal Arbors Quantifies Relationship between Branch Order and Geometry

2.5.1 Publication

Athey, T. L., Teneggi, J., Vogelstein, J. T., Tward, D. J., Mueller, U., & Miller, M. I. (2021). Fitting splines to axonal arbors quantifies relationship between branch order and geometry. *Frontiers in Neuroinformatics*, 38.

DOI: https://doi.org/10.3389/fninf.2021.704627

2.5.2 Relevant directory

`brainlit/experiemnts/axon_geometry`

2.5.3 How to reproduce this experiment

Before reproducing this experiment, make sure that:

- You have installed the `brainlit` package [Documentation]. Currently, you need to install the package from source to execute these codes.
- You have installed PyTorch [Documentation].

This experiment does *not* require a GPU (CUDA is not needed for PyTorch).

N.B. if you are using conda, make sure that both `brainlit` and PyTorch are installed within the same environment.

Now, follow these steps to reproduce the results of the experiment:

1. Download segments, which are stored in the publicly available S3 bucket open-neurodata

```
python scripts/download_segments.py
```

This script will prepare the experiment folder scaffolding

```
axon_geometry
├── data
│   ├── brain1
│   │   ├── segments_swc
│   │   └── trace_data
│   └── brain2
│       ├── segments_swc
│       └── trace_data
├── figures
│
... etc.
```

and download data from S3 (no credentials are required).

2. Compute and save trace analysis data

```
python scripts/generate_trace_data.py
```

3. Run any of the notebooks in the `notebooks` folder, which will save the results in the `figures` folder

2.6 Hidden Markov Modeling for Maximum Likelihood Neuron Reconstruction

2.6.1 Manuscript

Athey, T.L., Tward, D.J., Mueller, U. et al. Hidden Markov modeling for maximum probability neuron reconstruction. Commun Biol 5, 388 (2022). https://doi.org/10.1038/s42003-022-03320-0.

2.6.2 Relevant directory

`brainlit/experiemnts/ViterBrain`

2.6.3 How to use ViterBrain

- First, make sure that you have installed the `brainlit` package [Documentation].
- Second, uncompress the data `brainlit/experiments/ViterBrain/data/example.zip`. `brainlit/experiemnts/ViterBrain/data/sample.zip` can also be used.
- Make sure you are using Python3.9
- **Then, you can run some of the tutorial notebooks in the `notebooks` folder:**
  - `ViterBrain.ipynb` - shows a programmatic example of the pipeline, based on zarr inputs.
  - `fig3-voxels.ipynb` - generates Figure 3 from the paper.
  - `fig7-results.ipynb` - generates Figure 7 from the paper.
  - other notebooks can be useful for referemce, they were used in generating results in the paper.

- **The files in the `scripts` folder also can be useful:**
  - **`napari_gui.py` - shows the GUI prototype.**
    - click on colored fragment to select, red arrow will identify orientation.
    - o-key or *switch states* button to switch orientation of selected fragment.
    - click on another colored fragment (and hit o-key if necessary to switch orientation).
    - click on the labels layer in the left hand pane, then click somewhere on the image (not on a fragment)
    - t-key or *trace* button to trace between fragments.
    - c-key or *clear selected states* button to clear the selected fragments.
    - q-key or *clear all* button to clear all annotations.
    - n-key or *next color* button to change colors (3 total colors).
  - other scripts are for reference for benchmarking the timing of the pipeline.

2.7 Preserving Derivative Information while Transforming Neuronal Curves

2.7.1 Manuscript

Athey, T. L., Tward, D. J., Mueller, U., Younes, L., Vogelstein, J. T., & Miller, M. I. (2023). Preserving Derivative Information while Transforming Neuronal Curves. arXiv:2303.09649.

2.7.2 Neuron Mapping

- First, make sure that you have installed the `brainlit` package [Documentation].
- Second, uncompress the data `brainlit/experiments/map_neurons/data/mapping-files.zip`.
- Then, you can run the notebook that generates figures: `brainlit/experiments/map_neurons/map_neurons.ipynb`
- The files in the `other` are more for scratch work, and are unlikely to be useful to new users.

2.7.3 Poster

Athey, T. L., Vogelstein, J. T., & Miller, M. I. (2022). Nyquist Sampling Rate for Projection Neuron Reconstruction. Poster at Society for Neuroscience Meeting.

2.7.4 Relevant directory

`brainlit/experiemnts/map_neurons`

2.7.5 Nyquist Sampling Rate for Projection Neuron Reconstruction

- First, make sure that you have installed the `brainlit` package [Documentation].
- Results are in the notebook: `brainlit/experiments/map_neurons/sampling.ipynb`

2.8 BrainLine: An Open Pipeline for Connectivity Analysis of Heterogeneous Whole-Brain Fluorescence Volumes

2.8.1 Publication

Athey, T.L., Wright, M. A., Pavlovic, M., Chandrashekhar, V., Deisseroth, K., Miller, M.I., Vogelstein, J.T. (2023). BrainLine: An Open Pipeline for Connectivity Analysis of Heterogeneous Whole-Brain Fluorescence Volumes.

BioRxiv: https://www.biorxiv.org/content/10.1101/2023.02.28.530429v2

2.8.2 BrainLine: Whole-Brain Fluorescence Volume Analysis Pipeline

- First, make sure that you have installed the `brainlit` package: *Installation*.
- Some figure panels in our paper were created in `brainlit/experiments/BrainLine/BrainLine_transfer_learning.ipynb`.
- For information about the pipeline, see *BrainLine Tutorials*

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

BIBLIOGRAPHY

[1] http://ufldl.stanford.edu/tutorial/unsupervised/PCAWhitening/

INDEX

T

U

V

W

Z