
ConWhAt

Release d25712c

John Griffiths

May 14, 2018

About ConWhAt

1	Overview	1
2	Ontology & Representation	1
3	ConWhAt Atlases	3
4	Installation	4
5	Downloading ConWhAt Atlases	5
6	Exploring ConWhAt Atlases	6
7	Defining a Lesion	12
8	Assess network impact of lesion	13

1 Overview

Classical and modern schemas for defining and characterizing neuroanatomical structures in white matter tissue can be categorized along two main dimensions *Ontology* and *Representation*. Each of these has two main flavours: tract-based/connectivity-based, and image-based/streamline-based. This perspective is a key part of the rationale for, and design of, the ConWhAt software and atlases. Read more about these concepts [here](http://conwhat.readthedocs.io/en/latest/about_conwhat/ontology_and_representation.html)¹.

For info about the design and construction of the ConWhAt volumetric and streamlinetric atlases, see [here](http://conwhat.readthedocs.io/en/latest/about_conwhat/conwhat_atlases.html)².

2 Ontology & Representation

Classical and modern schemas for defining and characterizing neuroanatomical structures in white matter tissue can be categorized along two main dimensions *Ontology* and *Representation*. Each of these has two main flavours: tract-based/connectivity-based, and image-based/streamline-based.

¹ http://conwhat.readthedocs.io/en/latest/about_conwhat/ontology_and_representation.html

² http://conwhat.readthedocs.io/en/latest/about_conwhat/conwhat_atlases.html

2.1 Ontology

Conventional approaches to atlas white matter structures follow a *tract-based* ontology: they assign locations in stereotaxic space to a relatively small number of gross white matter tracts from the classical neuroanatomy literature.

This has been an extremely successful program of research, particularly in relation to post-mortem dissections and MR image or tractography streamline segmentation methodologies, as it draws on some of the brain's most salient and consistent macroscopic structural features.

Unfortunately, however, tract-based ontologies aren't particularly well-suited to network-based descriptions of brain organization. The reason for this is that identifying that a given spatial location falls within one or other canonical white matter tract (e.g. the inferior longitudinal fasciculus) doesn't in itself say very much about the specific grey matter connectivity of that location. Although they consist of many hundreds of thousands of structural connections (axons), the white matter tracts per se are not descriptions of connectivity, but rather of large 3D geometric structures that can be located relative to certain anatomical landmarks.

The second flavour of white matter ontology, which is becoming increasingly prominent in modern modern neuroscientific research, is a *connectivity-based* one. The idea here is that rather than following the classical anatomical tract nomenclature, to label white matter voxels according to the grey matter regions that their constituent fibers interconnect. Combining this with the modern macro-connectomics tractography approach (whole-brain tractography, segmented using region pairs from a given grey matter parcellation), gives the *connectome-based white matter atlas* methodology, which is what ConWhAt (and other earlier tools, notably NeMo) is designed to support.

The benefit of this approach is that a scientist/clinician/citizen can take a set of (standard space) coordinates, or a nifti-format ROI mask such as a binary lesion map, and straightforwardly query which grey matter region pairs (i.e. connectome-edges) have fibers passing through those locations.

That information can then be used together with the used parcellation's *canonical connectome* (normative group-averaged connectivity matrix), to obtain a lesion-modified structural (macro) connectome. This can be done very quickly with zero tractography data or analysis required, and as little as a list of numbers (voxel coordinates) as input.

An important point to emphasize is that the tract-based and connectivity-based ontologies are not diametrically opposed; in fact they should be regarded as highly complementary. This is why we have also included support in ConWhAt for standard tract-based atlas analyses.

2.2 Representation

Traditionally, anatomical atlases have (as the name suggests) existed as collections of more-or-less schematic two- or three-dimensional depictions, printed on the pages of a (generally quite large) book. Whilst this mode of representation is by no means uncommon, atlases in modern neuroimaging are generally understood to be digital data structures, which bear varying degrees of resemblance to their paper-based forebears.

In particular, representations of white matter anatomical data in neuroimaging come in two flavours: *image-based* (which we refer to as *volumetric*), and (tractography) *streamline-based* (which we refer to neologically as *streamlinetric*). These two forms of representation are very different beasts, each with its own set of distinctive features and pros/cons (which is why we make a major effort to support both in ConWhAt)

For example: the basic units of volumetric representations are scalar-valued (voxel intensities), which when taken as a set can code for complex and rich encoding of 3D shapes in virtue of their arrangement on a regular 3D grid. In contrast, the basic units of streamlinetric representations are vector-valued; namely lists of coordinates in 3D space. Each individual streamline (unlike each individual voxel) therefore provides some holistic 3D shape information. The closest equivalent of voxel intensities for

streamlines would be the presence of overlapping multiple streamlines; although this is much less compressed than scalar intensity values.

The definition and interpretation of ‘damage’ also turns out to be somewhat different for volumetric vs. streamlinetric representations. In the volumetric case, damage (defined as e.g. proportional overlap with a lesion) is evaluated independently for every voxel. In the streamlinetric case, damage is instead evaluated independently for every streamline, with the important corollary that evaluations at different spatial locations are not independent of each other. In short, if an upstream part of a streamline is considered to be damaged, then downstream parts are also considered to be damaged, even if they themselves are nowhere near the damaged area. Which is, of course, how one would expect real damage to axons to operate. Streamlinetric quantifications of damage are somewhat more difficult to work with than their volumetric equivalents, however.

There has been relatively little work done on direct comparisons of volumetric and streamlinetric characterizations of lesions, or indeed of white matter in general. ConWhAt is to our knowledge the first and only atlas-based tool that allows direct comparison between the two.


3 ConWhAt Atlases

A central component of ConWhAt is the large set of connectome-based white matter atlases we have developed for use with the software. The atlas construction methodology is described in detail in Griffiths & McIntosh (in prep). Here we give a brief summary:

All of the ConWhAt atlases were constructed from on [dipy](http://nipy.org/dipy/)³ deterministic whole-brain HARDI tractography reconstructions using the HCP WU-Minn corpus. Whole-brain streamline sets were segmented with region pairs using a broad set of brain parcellations, yielding anatomical connectivity matrices and connectome edge-labelled streamline sets. The streamlines are then entered into both volumetric and a streamlinetric atlas construction pipelines:

- Volumetric workflow: convert streamlines to track density images (visitation maps), spatially normalize, average
- Streamlinetric workflow: spatially normalize streamlines, concatenate, cluster

³ <http://nipy.org/dipy/>



`about_conwhat/../../figs/atlas_construction_fig.png`

4 Installation

4.1 Using latest github master source

Clone latest version from github

Now go to the cloned folder and install manually

Alternatively, simply add the cloned path to your pythonpath.

4.2 Using pypi

(coming soon)

4.3 Using with docker

(coming soon)

- Install docker-engine (instructions here)⁴
- Build the docker container
- Start Jupyter notebook server in the container

The following section was generated from `doc/examples/downloading_conwhat_atlases.ipynb`

5 Downloading ConWhAt Atlases

Import the fetcher function

```
In [34]: from conwhat.utils.fetchers import fetch_conwhat_atlas
import glob
```

Define the output directory

```
In [ ]: atlas_dir = '/scratch/hpc3230/Data/conwhat_atlases'
```

Define which atlas to grab

```
In [4]: atlas_name = 'CWL2k8Sc33Vol3d100s_v01'
```

Break a leg

```
In [30]: fetch_conwhat_atlas(atlas_name,atlas_dir,remove_existing=True);
```

removing existing folder

duplicate file detected - removing...

```
rm CWL2k8Sc33Vol3d100s_v01.zip
```

downloading data_file CWL2k8Sc33Vol3d100s_v01.zip...

```
wget https://www.nitrc.org/frs/download.php/10381/CWL2k8Sc33Vol3d100s_v01.zip
```

unzipping file...

```
unzip CWL2k8Sc33Vol3d100s_v01.zip
```

finished unzipping.

The zipped and unzipped atlas folders are now there in the top-level atlas directory;

```
In [35]: glob.glob(atlas_dir + '/*')
```

```
Out[35]: ['/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01.zip',
          '/scratch/hpc3230/Data/conwhat_atlases/README.md']
```

The .zip file can be optionally removed automatically if desired.

volumetric atlas folders contain a small number of fairly small .txt files

```
In [36]: glob.glob('%s/%s/*.txt' %(atlas_dir,atlas_name))
```

⁴ <https://docs.docker.com/engine/installation/>

```

Out[36]: ['/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/region_mapping_fsav_rh.txt',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/weights.txt',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/mappings.txt',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/region_labels.txt',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/bounding_boxes.txt',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/hemispheres.txt',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/cortex.txt',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/region_mapping_fsav_lh.txt']

...and a larger number of nifti images; one for each atlas structure

In [38]: glob.glob('%s/%s/*.nii.gz' %(atlas_dir, atlas_name))[:5]

Out[38]: ['/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/vismap_grp_7-64_norm.nii.gz',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/vismap_grp_23-30_norm.nii.gz',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/vismap_grp_65-69_norm.nii.gz',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/vismap_grp_55-70_norm.nii.gz',
          '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/vismap_grp_28-64_norm.nii.gz']

In [39]: len(glob.glob('%s/%s/*.nii.gz' %(atlas_dir, atlas_name)))

Out[39]: 2445

```

End of doc/examples/downloading_conwhat_atlases.ipynb

The following section was generated from doc/examples/exploring_conwhat_atlases.ipynb

6 Exploring ConWhAt Atlases

There are four different atlas types in ConWhat, corresponding to the 2 ontology types (Tract-based / Connectivity-Based) and 2 representation types (Volumetric / Streamlinetric).

(More on this schema [here](#)⁵)

```

In [1]: # ConWhAt stuff
        from conwhat import VolConnAtlas, StreamConnAtlas, VolTractAtlas, StreamTractAtlas
        from conwhat.viz.volume import plot_vol_scatter, plot_vol_and_rois_nilearn

        # Neuroimaging stuff
        import nibabel as nib
        from nilearn.plotting import plot_stat_map, plot_surf_roi

        # Viz stuff
        %matplotlib inline
        from matplotlib import pyplot as plt
        import seaborn as sns

        # Generic stuff
        import glob, numpy as np, pandas as pd, networkx as nx

```

We'll start with the scale 33 lausanne 2008 volumetric connectivity-based atlas.

Define the atlas name and top-level directory location

```

In [2]: atlas_dir = '/scratch/hpc3230/Data/conwhat_atlases'
        atlas_name = 'CWL2k8Sc33Vol3d100s_v01'

```

Initialize the atlas class

```

In [3]: vca = VolConnAtlas(atlas_dir=atlas_dir + '/' + atlas_name,
                           atlas_name=atlas_name)

```

⁵ http://conwhat.readthedocs.io/en/latest/about_conwhat/ontology_and_representation.html

```
loading file mapping
loading vol bbox
loading connectivity
```

This atlas object contains various pieces of general information

```
In [9]: vca.atlas_name
```

```
Out[9]: 'CWL2k8Sc33Vol3d100s_v01'
```

```
In [8]: vca.atlas_dir
```

```
Out[8]: '/scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01'
```

Information about each atlas entry is contained in the vfms attribute, which returns a pandas dataframe

```
In [14]: vca.vfms.head()
```

```
Out[14]:
```

	name	nii_file	nii_file_id	4dvolind	
0	61_to_80	vismap_grp_62-81_norm.nii.gz		0	NaN
1	38_to_55	vismap_grp_39-56_norm.nii.gz		1	NaN
2	28_to_38	vismap_grp_29-39_norm.nii.gz		2	NaN
3	18_to_19	vismap_grp_19-20_norm.nii.gz		3	NaN
4	26_to_55	vismap_grp_27-56_norm.nii.gz		4	NaN

Additionally, connectivity-based atlases also contain a networkx graph object vca.Gnx, which contains information about each connectome edge

```
In [62]: vca.Gnx.edges[(10,35)]
```

```
Out[62]: {'attr_dict': {'4dvolind': nan,
  'fullname': 'L_paracentral_to_L_caudate',
  'idx': 1637,
  'name': '10_to_35',
  'nii_file': 'vismap_grp_11-36_norm.nii.gz',
  'nii_file_id': 1637,
  'weight': 50.240000000000002,
  'xmax': 92,
  'xmin': 61,
  'ymax': 167,
  'ymin': 75,
  'zmax': 92,
  'zmin': 62}}
```

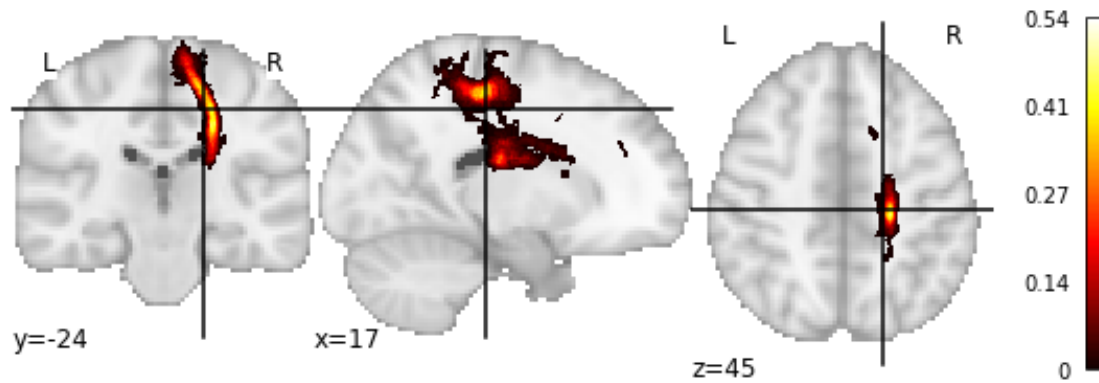
Individual atlas entry nifti images can be grabbed like so

```
In [144]: img = vca.get_vol_from_vfm(1637)
```

getting atlas entry 1637: image file /scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/vismap_

```
In [146]: plot_stat_map(img)
```

```
Out[146]: <nilearn.plotting.displays.OrthoSlicer at 0x7fb19fada410>
```



Or alternatively as a 3D scatter plot, along with the x,y,z bounding box

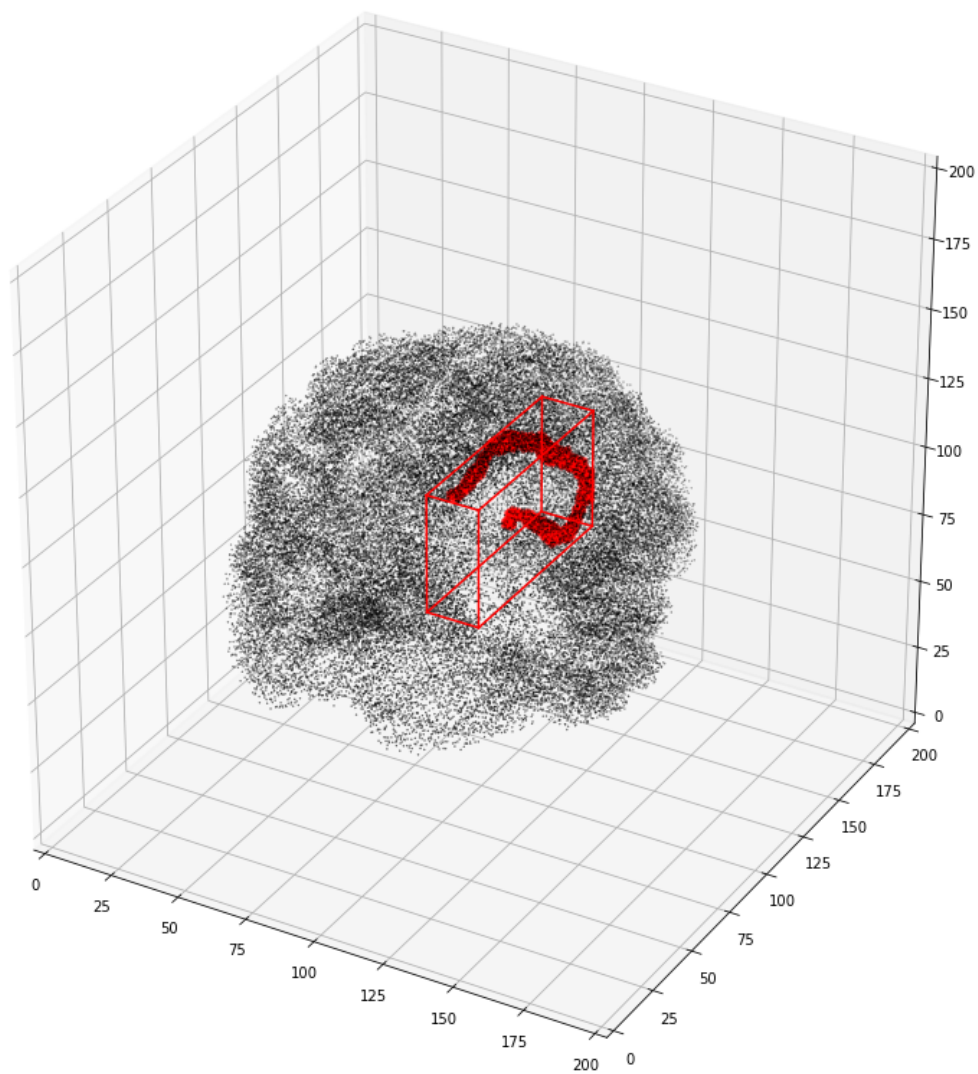
```
In [155]: vca.bbox.ix[1637]
```

```
Out[155]: xmin      61
          xmax      92
          ymin      75
          ymax     167
          zmin      62
          zmax      92
          Name: 1637, dtype: int64
```

```
In [134]: ax = plot_vol_scatter(vca.get_vol_from_vfm(1),c='r',bg_img='nilearn_destrieux',
                               bg_params={'s': 0.1, 'c':'k'},figsize=(20, 15))
```

```
ax.set_xlim([0,200]); ax.set_ylim([0,200]); ax.set_zlim([0,200]);
```

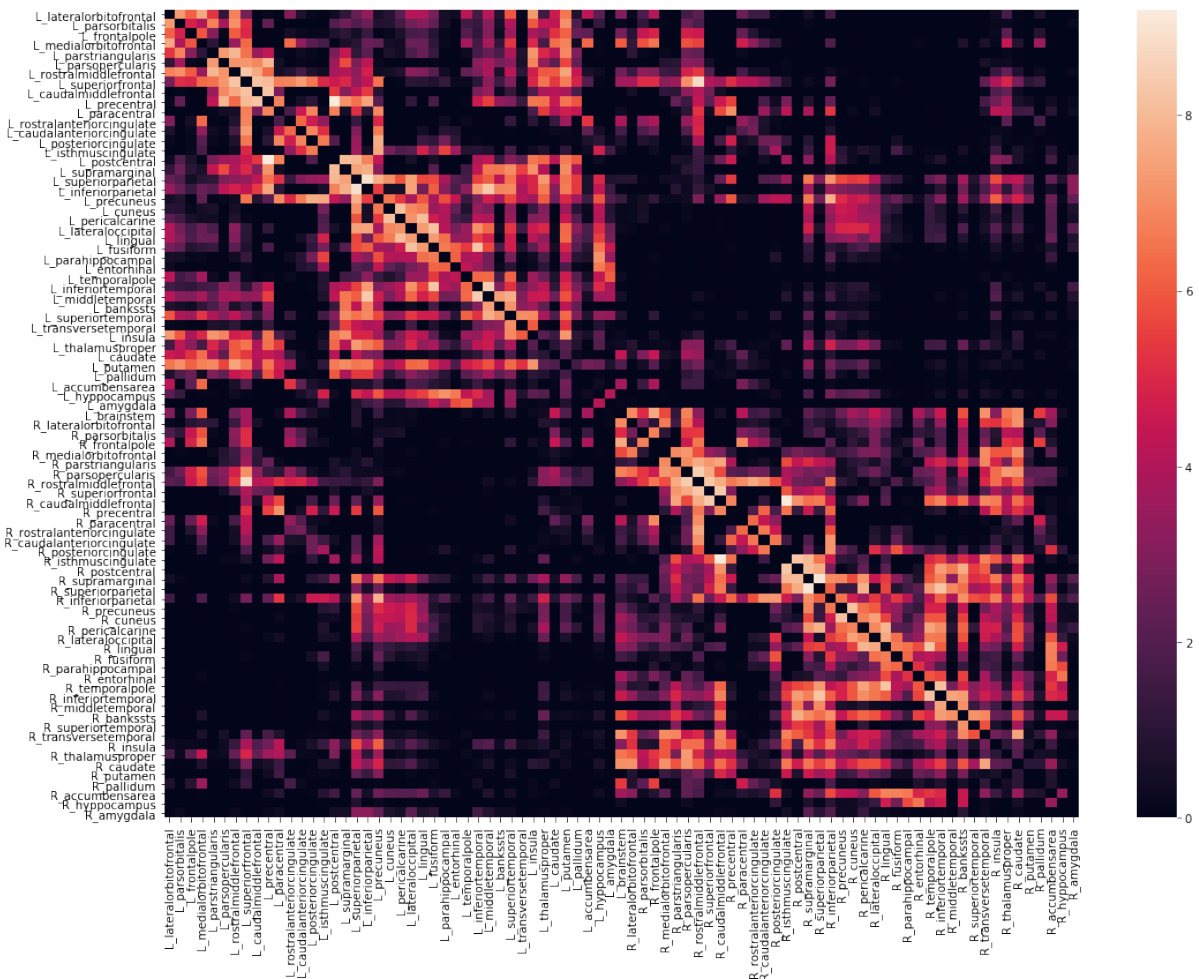
getting atlas entry 1: image file /scratch/hpc3230/Data/conwhat_atlases/CWL2k8Sc33Vol3d100s_v01/vismap_grp



We can also view the weights matrix like so:

```
In [38]: fig, ax = plt.subplots(figsize=(16,12))

sns.heatmap(np.log1p(vca.weights),xticklabels=vca.region_labels,
            yticklabels=vca.region_labels,ax=ax);
plt.tight_layout()
```



The `vca` object also contains `x,y,z` bounding boxes for each structure

We also stored additional useful information about the ROIs in the associated parcellation, including cortical/subcortical labels

```
In [156]: vca.cortex
```

```
Out[156]: array([[ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                   1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                   1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,
                   0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                   1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                   1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,
                   0.,  0.,  0.,  0.,  0.]])
```

...hemisphere labels

```
In [157]: vca.hemispheres
```

```
Out[157]: array([[ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                   1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                   1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                   1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                   0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                   0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                   0.,  0.,  0.,  0.,  0.,  0.]])
```

...and region mappings to freesurfer's fsaverage brain

```
In [158]: vca.region_mapping_fsav_lh
```

```
Out[158]: array([ 24.,  29.,  28., ...,  16.,   7.,   7.]
```

```
In [159]: vca.region_mapping_fsav_rh
```

```
Out[159]: array([ 24.,  29.,  22., ...,   9.,   9.,   9.]
```

which can be used for, e.g. plotting ROI data on a surface

```
In [167]: f = '/opt/freesurfer/freesurfer/subjects/fsaverage/surf/lh.inflated'  
          vtx,tri = nib.freesurfer.read_geometry(f)  
          plot_surf_roi([vtx,tri],vca.region_mapping_fsav_lh);
```



7 Defining a Lesion

Conducting a lesion analysis in ConWhAt is extremely simple. All that is needed is a binary .nii format lesion mask, with ones indicating lesioned tissue, and zeros elsewhere.

(Note: we terms like ‘lesion’ and ‘damage’ throughout most of this documentation, as that is the most natural primary context for ConWhAt analyses. Remember however that all we are doing at the end of the day is doing a set of look-up operations between a list of standard space coordinates on the one hand (as defined by non-zero values in a .nii image), and the spatial locations of each ‘connectome edge’ - i.e. each entry in our anatomical connectivity matrix. One can envisave many alternative interpretations/applications of this procedure; for example to map the connectivity effects of magnetic field or current distributions from nonivasive brain stimulation). Still, for concreteness and simplicity, we stick with ‘lesion’, ‘damage’, etc. for the most part.)

A common way to obtain a lesion map is to from a patient’s T1-weighted MR image. Although this can be done manually, it is strongly recommended to use an automated lesion segmentation tools, followed by manual editing.

An alternative way is simply to define a lesion location using standard space coordinates, and build a ‘lesion’ mask *de-novo*. This is what we do in the following example. On the next page we do a ConWhAt connectome-based decomposition analysis on this ‘synthetic’ lesion mask.

```
In [18]: # ConWhAt stuff
        from conwhat import VolConnAtlas,StreamConnAtlas,VolTractAtlas,StreamTractAtlas
        from conwhat.viz.volume import plot_vol_and_rois_nilearn

        # Neuroimaging stuff
        import nibabel as nib
        from nilearn.plotting import plot_roi
        from nipy.labs.spatial_models.mroi import subdomain_from_balls
        from nipy.labs.spatial_models.discrete_domain import grid_domain_from_image

        # Viz stuff
        %matplotlib inline
        from matplotlib import pyplot as plt

        # Generic stuff
        import numpy as np
```

Define some variables

```
In [25]: # Locate the standard space template image
        fsl_dir = '/global/software/fsl/5.0.10'
        t1_mni_file = fsl_dir + '/data/standard/MNI152_T1_1mm_brain.nii.gz'
        t1_mni_img = nib.load(t1_mni_file)

        # This is the output we will save to file and use in the next example
        lesion_file = 'synthetic_lesion_20mm_sphere_-46_-60_6.nii.gz'
```

Define the ‘synthetic lesion’ location and size using standard (MNI) space coordinates

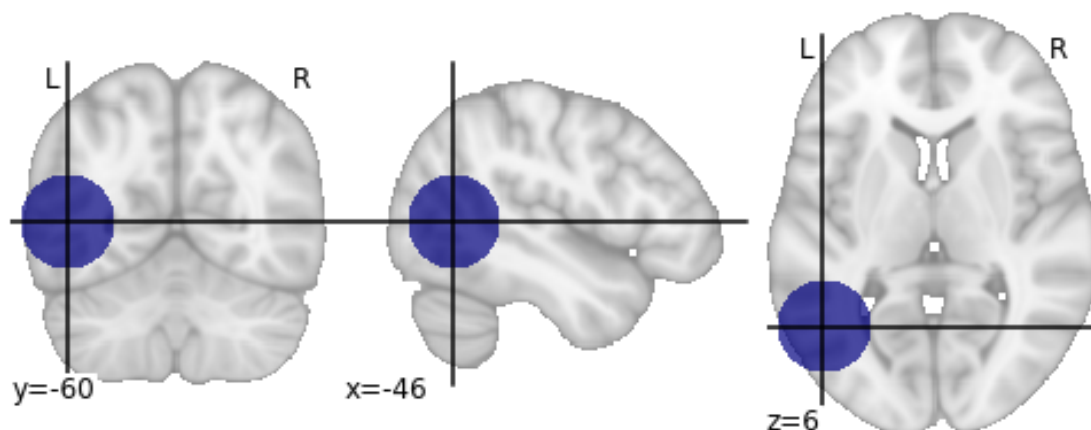
```
In [19]: com = [-46,-60,6] # com = centre of mass
         rad = 20          # radius
```

Create the ROI

```
In [20]: domain = grid_domain_from_image(t1_mni_img)
         lesion_img = subdomain_from_balls(domain,np.array([com]), np.array([rad])).to_image()
```

Plot on brain slices

```
In [28]: plot_roi(lesion_img,bg_img=t1_mni_img,black_bg=False);
```



Save to file

```
In [29]: lesion_img.to_filename(lesion_file)
```

...now we move on to doing a lesion analysis with this file.

End of doc/examples/defining_{as}synthetic_{lesion}.ipynb

The following section was generated from doc/examples/assessing_{the}network_{impact}of_{lesions}.ipynb

8 Assess network impact of lesion

```
In [1]: # ConWhat stuff
        from conwhat import VolConnAtlas,StreamConnAtlas,VolTractAtlas,StreamTractAtlas
        from conwhat.viz.volume import plot_vol_scatter

        # Neuroimaging stuff
        import nibabel as nib
        from nilearn.plotting import (plot_stat_map,plot_surf_roi,plot_roi,
                                     plot_connectome,find_xyz_cut_coords)
        from nilearn.image import resample_to_img

        # Viz stuff
        %matplotlib inline
        from matplotlib import pyplot as plt
        import seaborn as sns

        # Generic stuff
        import glob, numpy as np, pandas as pd, networkx as nx
        from datetime import datetime
```

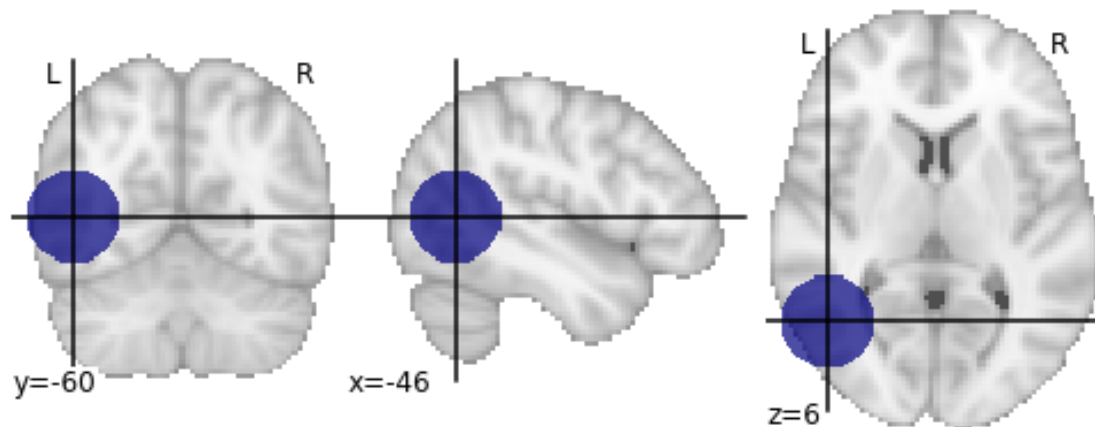
```
/global/home/hpc3230/Software/miniconda2/envs/jupyter/lib/python2.7/site-packages/h5py/_init__.py:36: Fut
from ._conv import register_converters as _register_converters
```

We now use the synthetic lesion constructed in the previous example in a ConWhat lesion analysis.

```
In [2]: lesion_file = 'synthetic_lesion_20mm_sphere_-46_-60_6.nii.gz' # we created this file from scratch
```

Take another quick look at this mask:

```
In [3]: lesion_img = nib.load(lesion_file)
        plot_roi(lesion_file);
```



Since our lesion mask does not (by construction) have a huge amount of spatial detail, it makes sense to use one of the lower-resolution atlas. As one might expect, computation time is considerably faster for lower-resolution atlases.

```
In [4]: cw_atlases_dir = '/global/scratch/hpc3230/Data/conwhat_atlases' # change this accordingly
        atlas_name = 'CWL2k8Sc33Vol13d100s_v01'
        atlas_dir = '%s/%s' %(cw_atlases_dir, atlas_name)
```

See the previous tutorial on ‘exploring the conwhat atlases’ for more info on how to examine the components of a given atlas in *ConWhat*.

Initialize the atlas

```
In [5]: cw_vca = VolConnAtlas(atlas_dir=atlas_dir)
```

```
loading file mapping
loading vol bbox
loading connectivity
```

Choose which connections to evaluate.

This is normally an array of numbers indexing entries in `cw_vca.vfms`.

Pre-defining connection subsets is a useful way of speeding up large analyses, especially if one is only interested in connections between specific sets of regions.

As we are using a relatively small atlas, and our lesion is not too extensive, we can assess all connections.

```
In [6]: idxs = 'all' # alternatively, something like: range(1,100), indicates the first 100 cnxns (rows in
```

Now, compute lesion overlap statistics.

```
In [18]: jlc_dir = '/global/scratch/hpc3230/joblib_cache_dir' # this is the cache dir where joblib writes
        lo_df, lo_nx = cw_vca.compute_hit_stats(lesion_file, idxs, n_jobs=4, joblib_cache_dir=jlc_dir)
```

computing hit stats for roi synthetic_lesion_20mm_sphere_-46_-60_6.nii.gz

This takes about 20 minutes to run.

`vca.compute_hit_stats()` returns a pandas dataframe, `lo_df`, and a networkx object, `lo_nx`.

Both contain mostly the same information, which is sometimes more useful in one of these formats and sometimes in the other.

`lo_df` is a table, with rows corresponding to each connection, and columns for each of a wide set of **statistical metrics**⁶ for evaluating sensitivity and specificity of binary hit/miss data:

In [28]: `lo_df.head()`

```
Out[28]:
```

metric	ACC	BM	F1	FDR	FN	FNR	FP	\
idx								
0	0.990646	0.104859	0.098135	0.911501	29696.0	0.889874	37851.0	
3	0.987324	0.011683	0.014279	0.988855	32708.0	0.980132	58828.0	
7	0.987160	-0.006617	0.001185	0.999075	33316.0	0.998352	59404.0	
10	0.994367	-0.000926	0.000147	0.999589	33368.0	0.999910	7305.0	
11	0.989105	0.048907	0.044941	0.962227	31520.0	0.944533	47152.0	

metric	FPR	Kappa	MCC	MK	NPV	PPV	TN	\
idx								
0	0.005266	0.330534	0.094054	0.084363	0.995864	0.088499	7149810.0	
3	0.008185	0.329134	0.008766	0.006577	0.995433	0.011145	7128833.0	
7	0.008265	0.329023	-0.004966	-0.003727	0.995348	0.000925	7128257.0	
10	0.001016	0.331450	-0.001976	-0.004215	0.995374	0.000411	7180356.0	
11	0.006560	0.329846	0.040403	0.033378	0.995605	0.037773	7140509.0	

metric	TNR	TP	TPR	corr_nothr	corr_thr	corr_thrbin
idx						
0	0.994734	3675.0	0.110126	0.042205	0.042205	0.094054
3	0.991815	663.0	0.019868	-0.001487	-0.001487	0.008766
7	0.991735	55.0	0.001648	-0.003549	-0.003549	-0.004966
10	0.998984	3.0	0.000090	-0.001975	-0.001975	-0.001976
11	0.993440	1851.0	0.055467	0.017664	0.017664	0.040403

Typically we will be mainly interested in two of these metric scores:

TPR - True positive (i.e. hit) rate: number of true positives, divided by number of true positives + number of false negatives

corr_thrbin - Pearson correlation between the lesion amge and the thresholded, binarized connectome edge image (group-level visitation map)

In [27]: `lo_df[['TPR', 'corr_thrbin']].iloc[:10].T`

```
Out[27]:
```

idx	0	3	7	10	11	13	\
metric							
TPR	0.110126	0.019868	0.001648	0.000090	0.055467	0.002128	
corr_thrbin	0.094054	0.008766	-0.004966	-0.001976	0.040403	0.005801	

idx	14	15	18	19
metric				
TPR	0.000569	0.000000	0.098469	0.023523
corr_thrbin	0.000641	-0.002543	0.169234	0.029414

We can obtain these numbers as a 'modification matrix' (connectivity matrix)

```
In [33]: tpr_adj = nx.to_pandas_adjacency(lo_nx,weight='TPR')
         cpr_adj = nx.to_pandas_adjacency(lo_nx,weight='corr_thrbin')
```

These two maps are, unsurprisingly, very similar:

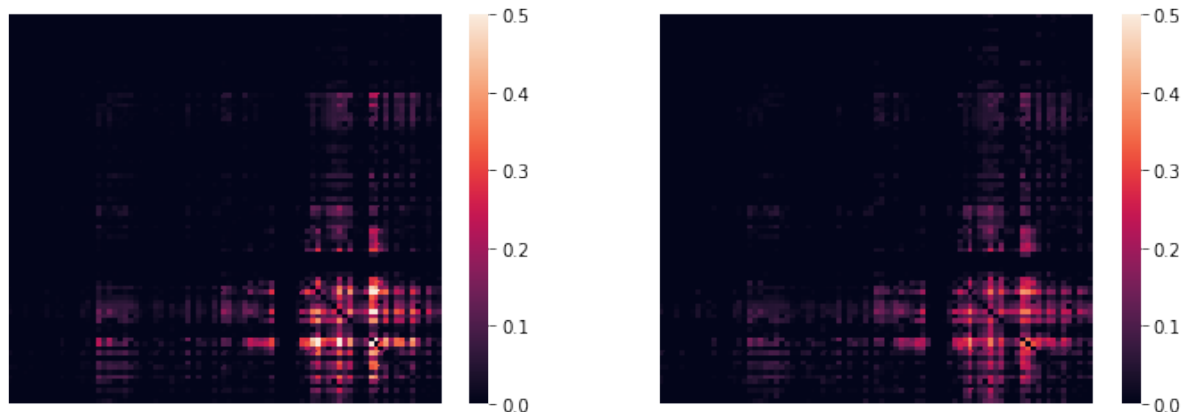
⁶ https://en.wikipedia.org/wiki/Sensitivity_and_specificity

```
In [104]: np.corrcoef(tpr_adj.values.ravel(), cpr_adj.values.ravel())
Out[104]: array([[1.          , 0.96271946],
                [0.96271946, 1.          ]])

In [79]: fig, ax = plt.subplots(ncols=2, figsize=(12,4))

sns.heatmap(tpr_adj,xticklabels='',yticklabels='',vmin=0,vmax=0.5,ax=ax[0]);

sns.heatmap(cpr_adj,xticklabels='',yticklabels='',vmin=0,vmax=0.5,ax=ax[1]);
```

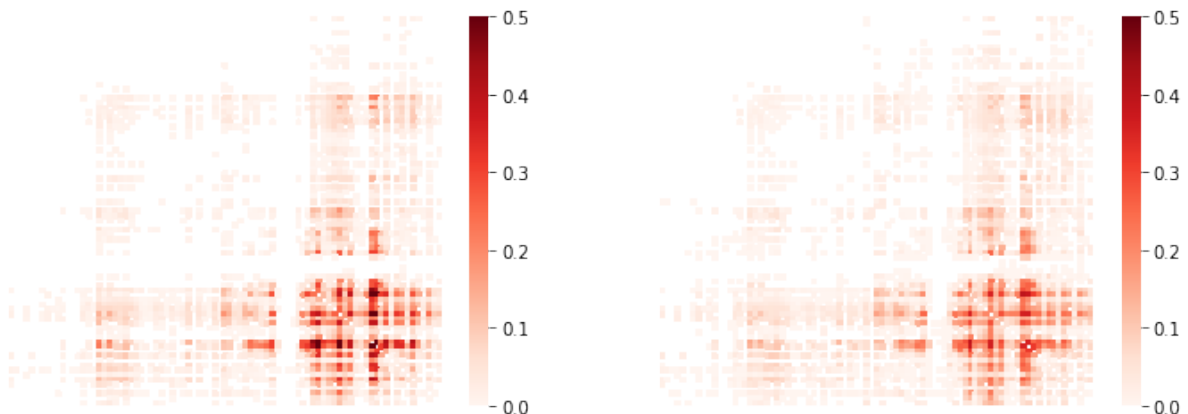


(...with an alternative color scheme...)

```
In [70]: fig, ax = plt.subplots(ncols=2, figsize=(12,4))

sns.heatmap(tpr_adj, xticklabels='',yticklabels='',cmap='Reds',
            mask=tpr_adj.values==0,vmin=0,vmax=0.5,ax=ax[0]);

sns.heatmap(cpr_adj,xticklabels='',yticklabels='',cmap='Reds',
            mask=cpr_adj.values==0,vmin=0,vmax=0.5,ax=ax[1]);
```



We can list directly the most affected (greatest % overlap) connections,

```
In [85]: cw_vca.vfms.loc[lo_df.index].head()

Out[85]: name                nii_file  nii_file_id  4dvolind
idx
0    61_to_80  vismap_grp_62-81_norm.nii.gz          0      NaN
3    18_to_19  vismap_grp_19-20_norm.nii.gz          3      NaN
7    45_to_48  vismap_grp_46-49_norm.nii.gz          7      NaN
10   19_to_68  vismap_grp_20-69_norm.nii.gz         10      NaN
11   21_to_61  vismap_grp_22-62_norm.nii.gz         11      NaN
```

To plot the modification matrix information on a brain, we first need to some

spatial locations to plot as nodes. For these, we calculate (an approximation to) each atlas region's centroid location:

```
In [101]: parc_img = cw_vca.region_nii
          parc_dat = parc_img.get_data()
          parc_vals = np.unique(parc_dat)[1:]

          ccs = {roival: find_xyz_cut_coords(nib.Nifti1Image((dat==roival).astype(int),img.affine),
                                             activation_threshold=0) for roival in roivals}

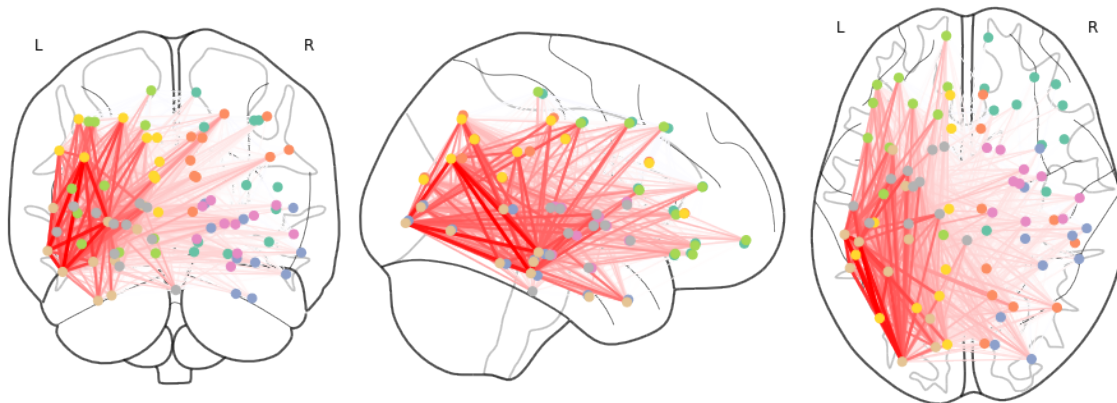
          ccs_arr = np.array(ccs.values())
```

Now plotting on a glass brain:

```
In [ ]: fig, ax = plt.subplots(figsize=(16,6))
        plot_connectome(tpr_adj.values,ccs_arr,axes=ax,edge_threshold=0.2,colorbar=True,
                        edge_cmap='Reds',edge_vmin=0,edge_vmax=1.,
                        node_color='lightgrey',node_kwargs={'alpha': 0.4});
        #edge_vmin=0,edge_vmax=1)
```

```
In [118]: fig, ax = plt.subplots(figsize=(16,6))
          plot_connectome(cpr_adj.values,ccs_arr,axes=ax)
```

```
Out[118]: <nilearn.plotting.displays.OrthoProjector at 0x7f454cea5b90>
```



End of doc/examples/assessing_the_network_impact_of_lesions.ipynb