Appendix

1 Dataset Details

1.1 Spatial autocorrelation between images

We computed the cross covariance of the images in our main subvolumes to see how much correlation exists between consecutive or nearby image slices. This analysis confirmed that after 5-7 slices, the cross-correlation between images drops off consistently (see Figure A1). Based upon this analysis, we crop out 10 slices between the training and testing sets in all the subvolumes.

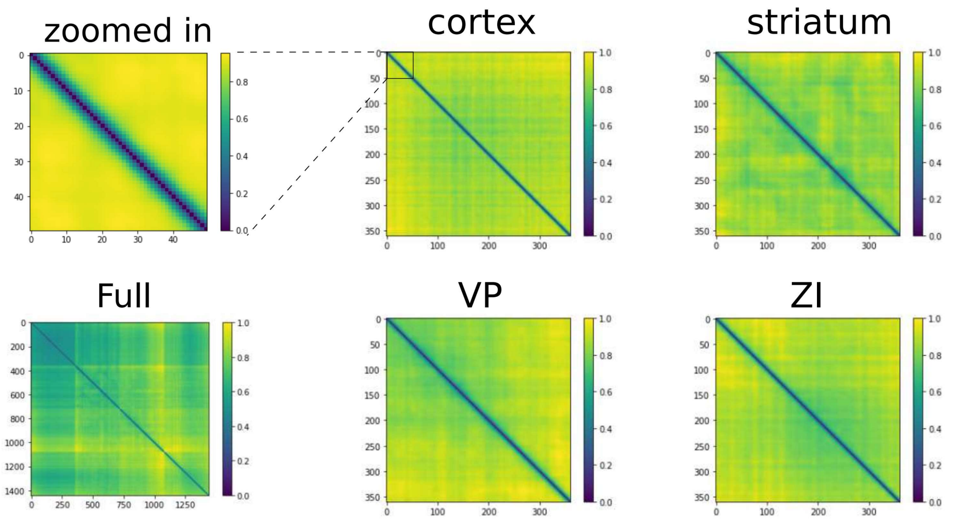

Figure A1: *Cross-covariance matrices.* Per-class cross-covariance of all 4 considered regions (right), zoomed-in view of cortex cross-covariance(top left) and cross-covariance of entire dataset (bottom left).

1.2 Dense pixel-level annotations

Utilizing the sparse annotations from (15), we trained a 2D UNet to segment the image data into 4 classes (blood vessels, cell bodies, axons, and background) and applied it to the slices in the four volumes which were not already annotated. A proofreader then reviewed the annotations for each slice in the volume using ITK-Snap (66) in the y-z plane. During the proofreading process of a slice, any structure with labels that were split between classes (errors in the UNet) were first corrected. Then, any structures without split labels were reviewed for correctness and changed if incorrect. Finally, the annotator checked the following Z slice to ensure continuity of components. This process continued for each slice in the volume. To ensure consistency of the annotations, the volume was proofread by the same annotator a second time, now going through the x-z plane. After proofreading we generated densely annotated volumes of size 256x256x360 in each of the four ROIs.

1.3 Interpolated ROI annotations

A full ROI annotation of the x-ray microtomography dataset was interpolated from manually annotated z-level slices that were roughly 50 pixels apart. To generate the interpolated layers, a recursive algorithm was implemented to linearly shift the boundary between two annotation classes from one pre-existing layer to the next: once a shifting boundary region was located, the midpoint of the shift was identified and full, one voxel-wide annotations were created along the midpoint spanning all necessary z levels. This process split transition regions into two roughly equal spaces, in which the recursive process repeated for the two respective transition spaces until the entire transition region was fully annotated.

However, this process was not able to account for all border transitions. Thus, about 10% of the border transitions were annotated manually using the visualization software application ITK-SNAP. Fully interpolated ROI annotations are available for z-levels between 109 and 459, inclusive.

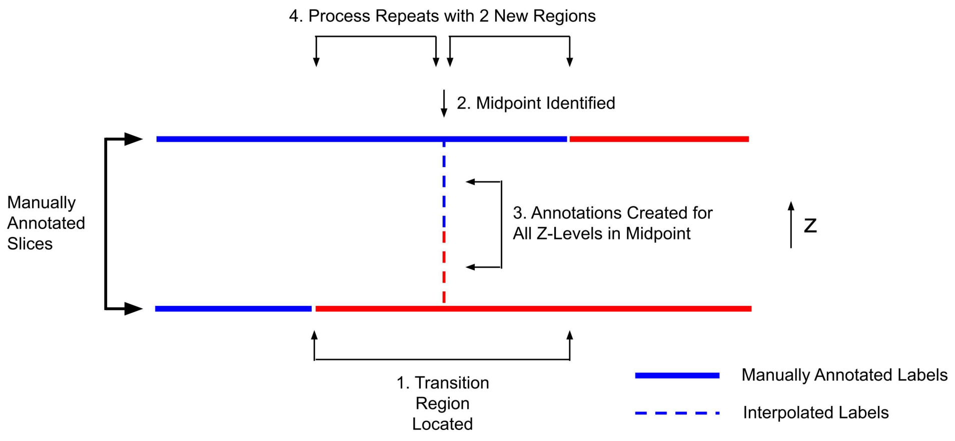
4. Process Repeats with 2 New Regions
2. Midpoint Identified
Manually Annotated Slices
3. Annotations Created for All Z-Levels in Midpoint
Z
1. Transition Region Located
Manually Annotated Labels
Interpolated Labels

Figure A2: *The Recursive Process of the Interpolation of ROI Annotations.* A four step recursive process was implemented to produce interpolated ROI annotation layers from manually annotated layers. This process recursively repeated until the entire transition region was interpolated with ROI annotations.

2 Data Access

To allow for easy, publicly accessible data, the dataset is stored in the BossDB system. Critically, this database enables efficient access of arbitrary cutouts of large volumetric datasets. This project page is available at `https://bossdb.org/project/prasad2020`. No user account is required. Public credentials will allow read-only access of the data without the need for user account creation.

The raw images are available with the resource identifier `bossdb://prasad/prasad2020/image`, which is the "prasad" experiment, "prasad2020" experiment, and "image" channel. The raw image data are in an unsigned 8-bit integer format, with preferred indexing in ZYX format. Areas with invalid data (e.g. outside the image volume) are assigned the value 0. The maximum and minimum indices are $z = [0, 720]$, $y = [0, 1420]$, and $x = [0, 5805]$. The data have $1.17\mu m$ isotropic resolution.

The annotation images are available with the resource identifier `bossdb://prasad/prasad_analysis/pixel_labels` and `bossdb://prasad/prasad_analysis/roi_labels`, which is the "prasad" collection, "prasad_analysis" experiment, and "pixel_labels" and "roi_labels" channels. The annotation data are in an unsigned 64-bit integer format, with preferred indexing in ZYX format. Areas with invalid data (e.g. outside the image volume) are assigned the value 0. As for the raw images channel, the maximum and minimum indices are $z = [0, 720]$, $y = [0, 1420]$, and $x = [0, 5805]$. The data have $1.17\mu m$ isotropic resolution.

Access is available through the Python intern array API, which provides numpy-like referencing. Using this library, the user creates an intern array, `image_array = array(boss_url)`, where `boss_url` is one of the resource identifiers listed above. The user can then access data from the channel using the numpy-like syntax, demonstrated by this code example to download a cutout corresponding to cortex.

```
from intern import array
image = array(boss_url)
data = image[110:379, 900:1156, 4600:4856]
```

A data loader is provided for rapid development of Pytorch models, which is used for all models tested in this work. On creation, the data loader loads a task configuration '.json' file which specifies the parameters of the task. Two modes of operation are allowed, `download=true` and `download=false`. For the former, on creation of the data loader, data are pulled locally and stored as a numpy file. If the numpy file already exists, it is instead loaded from disk. In the later case, data are downloaded to memory and not written to disk. The data are stored in a numpy tensor containing the concatenated slices from cortex, striatum, vp, zi (for four class problems). The item retrieval function can serve up integer region labels (for task 1), or microstructure masks (for task 2), as dictated by the configuration. The basic input data configuration parameters are:

- **image_chan** - BossDB channel from which to pull the raw image data.
- **annotation_chan** - BossDB channel from which to pull the image annotations (either macro- or micro-structures).
- **xrange_cor** - range along the x-axis (on the full data) for the slices from Cortex.
- **yrange_cor** - range along the y-axis (on the full data) for the slices from Cortex.
- **xrange_stri** - range along the x-axis (on the full data) for the slices from Striatum.
- **yrange_stri** - range along the y-axis (on the full data) for the slices from Striatum.
- **xrange_vp** - range along the x-axis (on the full data) for the slices from VP.
- **yrange_vp** - range along the y-axis (on the full data) for the slices from VP.
- **xrange_zi** - range along the x-axis (on the full data) for the slices from ZI.
- **yrange_zi** - range along the y-axis (on the full data) for the slices from ZI.
- **z_train** - the range (slices) along the z-axis to use as the train set.
- **z_val** - the range (slices) along the z-axis to use as the val set.
- **z_test** - the range (slices) along the z-axis to use as the test set. A buffer of 10 slices is recommended between the train/val set and the test set.
- **volume_z** - the number of slices to include in a volume slice. For the 2D cases, the individual slices are the input to the models (number of slices in a volume slice is 1). In the 3D case, volume slices are the input to the models.

The repository, `https://github.com/MTNeuro/MTNeuro`, contains scripts and python notebooks for data access and running models. Python notebooks which download the numpy files for images and annotations are provided for users not using Pytorch. These are based on the original data access notebooks developed for this dataset `https://github.com/nerdslab/xray-thc`. The repository and requirements can be installed via the `pip` package manager. The repository is structured as

- MTNeuro: the core code for running pytorch dataloaders and pytorch models
- Notebooks: access notebooks for users who do not use pytorch
- Scripts: example scripts for running pytorch models for each task, which form the basis for new algorithm development

3 Further Details on Task 1

3.1 Models

3.1.1 Supervised models

- Resnet18: 18-layer version of a model that reformulates the layers as learning residual functions with reference to the layer inputs in order to efficiently train deeper architectures (44).
- Resnet18 + Mixup: same model architecture described above (Resnet18), but trained using mixup, an augmentation that mixes inputs as well as their corresponding labels through a convex combination, and has been shown to yield significant improvement in supervised classification models (45)

3.1.2 Self-supervised models

We test several contrastive-based SSL methods that do not need negative examples to generate good representations of data (46). All SSL models are trained using a Resnet18 encoder backbone in order to make them comparable to the evaluated supervised models.

We tested the following SSL models:

- BYOL: BYOL relies on a mirrored structure of online and target networks that learn from each other (46). In particular, the online network tries to predict the encoding of a target image view x by minimizing the distance in latent space to an augmented view x_a.
- MYOW: MYOW builds on BYOL by also minizing the distance in latent space between views, but incorporates a view mining approach in order to search the dataset for neighboring samples in representations space. The augmented and mined samples are then integrated into a unified latent space through the use of an additional predictor network (47).
- MYOW-m: MYOW-m is an extension of MYOW that integrates MYOW's sample mining procedure, but uses a single predictor on both the augmented and mined views rather than using the cascaded projector design proposed in MYOW.

Since all of these SSL methods rely on some form of transformation to process the views during training, we choose the simplest possible augmentation for all three methods: randomly cropping patches half the width and height of each image sample to generate augmented and mined views (see Figure A3). In order to standardize the evaluation of the methods, we evaluate the trained encoder's performance in classifying which class each of the four corners of an image sample belongs to.

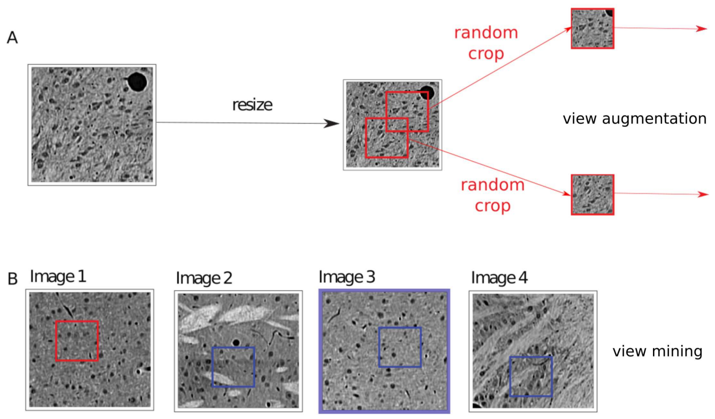

Figure A3: *View generation for self-supervised approaches.* In A we show how augmented views are generated by extracting random crops from a given image sample. In B, we schematize the view mining process, involving comparing different random crops from a pool of samples to a target view (red) and choosing the closest one in representation space as the mined view (Image 3, highlighted).

3.2 More Details on Training

3.2.1 Configuration Files

Two types configuration files are provided as input for training:

- **Network Configuration** - a '.json' file containing settings for the optimizer, augmentations to use, the parameters for the corresponding supervised or self-supervised models, and the seed to use for the training.
- **Task Configuration** - a '.json' file specifying information regarding the data slices to access for the training. This includes the database channel to access for the data and region-level annotations, the x and y ranges for the slices from each region, the size of the slices (corresponding to the level of downsampling applied), the train-val-test split, the depth of each slice sample (1, since we focus on 2D processing for this task).

Setting	Values cube set A
xrange_cor	[4600,4856]
yrange_cor	[900,1156]
xrange_stri	[3700,3956]
yrange_stri	[500,756]
xrange_vp	[3063,3319]
yrange_vp	[850,1106]
xrange_zi	[1543,1799]
yrange_zi	[650,906]
z_train	[110, 379]
z_val	[380, 409]
z_test	[420, 470]

Table A1: *Dataset configuration used for task 1 (ROI-C1) and task 2*

Setting	Values cube set B	Values cube set C	Values cube set D
xrange_cor	[5312,5568]	[5056,5312]	[4600,4856]
yrange_cor	[388,644]	[644,900]	[400,656]
xrange_stri	[3828,4084]	[3344,3600]	[3800,4056]
yrange_stri	[912,1168]	[400,656]	[244,500]
xrange_vp	[2551,2807]	[2151,2407]	[2295,2551]
yrange_vp	[850,1106]	[950,1206]	[694,950]
xrange_zi	[1287,1543]	[1031,1287]	[1799,2055]
yrange_zi	[906,1162]	[906:1162]	[650:906]

Table A2: *Coordinates for the additional cube sets used in ROI-C2 (C and D) and ROI-C3 (B, C and D)*

3.2.2 Model Settings

The basic model settings are:

- **model** - either the classifier (if supervised) or encoder backbone (if SSL) to train for the classification task.
- **classes** - the number of output classes of the model.
- **method** - (only for SSL) the self-supervised approach to train.

3.2.3 Input Data Configuration

The dataset configurations used for the results in Table 1, column ROI-C1 are listed in Table A1. The coordinates for the additional cubes used in ROI-C2 and ROI-C3 are shown in Table A2.

3.2.4 Training and evaluation setup

We train all supervised and self-supervised models using a 0.03 learning rate, a batch size of 256 (chosen for a stable linear layer evaluation) and 5 different random seed values: 1, 100, 350, 631 and 872. We compile these 5 results in order to calculate the mean and standard deviation of the classification accuracy for each considered approach. For the Resnet18-Mixup setting, we evaluate 5 different probabilities of applying the augmentation during training: 0.01, 0.1, 0.3, 0.5 and 0.7; and report the setting with the best performing mean accuracy.

3.3 Additional experiments

We report additional tests in Table A3, where we evaluate all considered models under the ROI-C1 training setting, but using different downsampling factors on the image samples: 4x, 2x and full resolution (no downsampling). We observe that performance increases with the resolution (as more information is available to the models). Surprisingly, we see a significantly larger performance jump in the SSL methods when moving from 2x downsampling to full resolution tests with respect to the supervised methods (which increase their performance only by a slight margin). This further

Table A3: *Results on image classification for brain area prediction (Task 1).*

	ROI-C1		
	Downsampling =4	*Downsampling =2*	*Full-res*
Resnet18	0.83 ± 0.06	0.88 ± 0.03	0.87 ± 0.1
Resnet18-Mixup	0.85 ± 0.06	0.90 ± 0.04	0.91 ± 0.03
BYOL	0.83 ± 0.04	0.88 ± 0.02	0.98 ± 0.01
MYOW	0.84 ± 0.04	0.90 ± 0.02	0.96 ± 0.03
MYOW-m	0.84 ± 0.05	0.94 ± 0.02	0.99 ± 0.01
PCA	0.59	0.59	0.59
NMF	0.55	0.62	0.54

supports our observation that SSL models benefit more from exposure to additional data than their supervised counterparts. Each SSL training instance took on average 30 minutes to train under 4x downsampling, 1.5 hours under 2x downsampling, and 4 hours under full resolution setting (runtime for a single random seed, trained on an RTX 3090 GPU node). We chose to use 2x downsampling for the rest of our experiments in task 1, since it provided a good compromise between performance and runtime.

4 Further Details on Task 2

4.1 Baselines

4.1.1 2D Models

The models we use for the 2D segmentation task are:

- A standard **2D U-Net** model (48; 49)
- Selected models from the 'segmentation_models.pytorch' library (50):
  - **MAnet** - a model utilizing a multi-scale attention mechanism, originally design for liver and liver tumor segmentation (51),
  - **FPN** - the Feature Pyramid Network architecture for object detection modified for image segmentation (52),
  - **U-Net++** - a nested U-Net architecture developed for medical image segmentation (53),
  - **PAN** (Pyramid Attention Network) - a model incorporating spatial pyramid attention structure, designed to utilize global contextual information in semantic segmentation (54),
  - **PSPNet** (Pyramid Scene Parsing Network) - a model utilizing pyramid pooling and a scene parsing network to learn global context information better (55).

4.1.2 3D Models

The models we use for the 3D segmentation task are:

- A standard **3D U-Net** model (48),
- Selected models from 'MedicalZooPytorch' (56):
  - **VNetLight** - a lighter version of the V-Net convolutional network architecture developed to perform volumetric segmentation (57; 56),
  - **HighResNet** - a compact, high-resolution convolutional network for volumetric segmentation, originally demonstrated on brain parcellation pretext task on brain MR images (58).

4.2 More Details on Training

4.2.1 Configuration Files

Two types configuration files are provided as input to training:

- **Network Configuration** - a '.json' file containing settings for the optimizer, augmentation setting, the model settings, evaluation settings, settings for saving the output and the seed to use for the training.
- **Task Configuration** - a '.json' file specifying information regarding the data slices to access for the training. This includes the database channel to access for the data and annotations, the x and y ranges for the slices from each region, the size of the slices, the train-val-test split, the size of the volume slice (3D) and the whether the training is for 3-Class or 4-Class setting. The task configuration used for the results in Table 2 are listed in Table A1.

Together, these two files completely specify the configurations for the training run.

4.2.2 Model Settings

The basic model settings are:

- **encoder_name** - the encoder to use with the model. This is applicable only for the 2D models and the 'UNet_3D' model. For more information visit 'segmentation_models.pytorch' library ((50)). For the training results in Table 2, 'efficientnet-b7' was used as the encoder as it gave the best performance among several other encoders that were tried.
- **encoder_weights** - the pre-trained weights to use for the model. For the 2D model and 'UNet_3D' results in Table 2, weights trained on ImageNet were used.
- **in_channels** - the number of input channels.
- **classes** - the number of output classes.

4.2.3 Details on Class Proportions

At the microstructure level/scale, the frequency of each class across the brain regions are as follows:

- Cortex - Background: 93%; Blood Vessel: 2.33%; Cell: 4.64%; Axons: 0%
- Striatum - Background: 72.63%; Blood Vessel: 2.5%; Cell: 5.66%; Axons: 19.22%
- VP - Background: 22.75%; Blood Vessel: 4.73%; Cell: 5.73%; Axons: 66.8%
- ZI - Background: 32.15%; Blood Vessel: 5.4%; Cell: 9.41%; Axons: 53.04%
- Total - Background: 55.13%; Blood Vessel: 3.74%; Cell: 6.36%; Axons: 34.77%

4.2.4 Hyper-Parameters and Random Seeds

For the selection of optimal hyper parameters for each model a hyper parameter search is performed by training the models for the following learning rates: 0.1, 0.05, 0.01, 0.005, 0.001; and the following batch sizes: 2,4,6,8,10. The best performing learning rate and batch size is chosen and 5 separate instances of each model are trained with this optimal learning rate and batch size, with seeds: 1, 100, 350, 631 and 872 respectively. These 5 results (for each model) are used to calculate the mean and SD ($Mean \pm SD$) of the performance metrics (which is reported in Table 2). Also across several models it was seen that Adam optimizer was yielding a better result than SGD, so Adam optimizer was used for all the training runs. The optimal hyperparameters found for each model in each setting are listed in Table A4.

5 Further Details on Task 3

In order to extract the semantic features from the microstructure annotations in the 4 densely connected cubes, we first isolate the relevant corresponding pixel-level labels (either cells, blood vessels or axons). Once we have extracted the desired class and labeled everything else as background, we can perform a connected component analysis to compute the desired semantic features (cell count, size, and distance, as well as axon and vessel density) for each image sample.

Once the semantic features for different tasks are calculated, we extract the representations of all samples across the 4 cubes using the models trained in task 1 ROI-C1 (Table 3, top) and ROI-C3

Table A4: *Optimal Hyperparameters used for Task 2 models in Table 2*

I. 2D Pixel-level Segmentation

	3-Class		*4-Class without ZI*	
Model	Learning Rate	Batch Size	Learning Rate	Batch Size
2D U-Net	0.1	8	0.01	4
MA-Net	0.01	2	0.001	8
FPN	0.01	2	0.01	4
U-Net++	0.001	8	0.01	8
PAN	0.01	10	0.001	8
PSPNet	0.01	4	0.1	2

II. 3D Pixel-level Segmentation

	3-Class		*4-Class without ZI*	
Model	Learning Rate	Batch Size	Learning Rate	Batch Size
3D U-Net	0.005	1	0.01	1
VNetLight	0.01	1	0.01	1
HighResNet	0.005	1	0.001	1

(Table 3, bottom), and use scikit-learn to fit a linear regression in order to predict the semantic features and report the R^2 on the entire subvolume of interest. Furthermore, we provide visualizations of the representations and the corresponding semantic features of the 4 interest cubes using a BYOL encoder trained under two different dataset conditions (ROI-C1 and ROI-C3) in Figure A4.

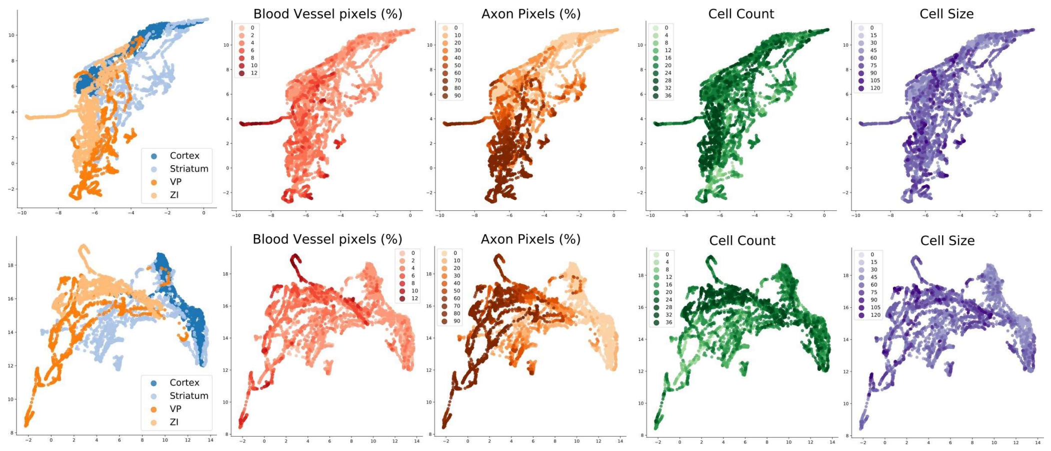

Figure A4: *Visualizations of semantic features overlaid on two dimensional learned representations.* Here, the first row shows 2D U-map projections of learned BYOL (ROI-C1) embeddings and overlay the different semantic attributes on the latents . From left to right, we color the embeddings by brain area (class), % blood vessels, % axons, cell count, and cell size. The second row shows the 2D projections of learned BYOL (ROI-C3) embedding which too are overlaid with different semantic attributes.

6 Maintenance, Licensing, and Ethical Concerns

The dataset, data access API, and dataloaders are maintained by the BossDB team (bossdb.org), which is tasked as the BRAIN Initiative archive of record for nanoscale connectomics datasets. The team is developing standards in accordance with FAIR principles (`https://www.go-fair.org/fair-principles/`) to ensure permanent identifiers and broad accessibility. The data are available under the CC-BY-4.0 license. The baseline code is available open source hosted in a github organization, and community forks and pull requests will be welcome, to be reviewed by the repository maintainers. As improvements are made to the baseline codebase for future challenges, changes will be pushed to the MTNeuro repository as a new versioned release.

The dataset used includes animal data which was collected under an approved IACUC protocol, as detailed in the original paper. There is no human derived data in this dataset. We emphasize this dataset is for development of algorithms for fundamental analysis of structural neuroscience data, but it is possible future efforts could use these data inappropriately for the development of clinically relevant algorithms which could result in negative societal impacts. This use is discouraged due to the unknown generalizations of high-resolution X-ray Microtomography to clinical modalities.