Supplementary Materials for "Towards Explainable Automated Neuroanatomy"

Anonymous

Anonymous Organizations
@**.***

Details about OpenCV procedure For cell segmentation from brain section images, we used the adaptive thresholding technique provided by OpenCV. This process, specifically through the `cv.adaptiveThreshold` function, computes the threshold for a pixel based on a small region around it. The parameters used in this function were as follows:

```
thresholdType: cv2.THRESH_BINARY,
adaptiveMethod: cv.ADAPTIVE_THRESH_GAUSSIAN_C,
blockSize: 101 (chosen based on the typical cell size),
C: -12 (selected manually based on segmentation quality).
```

Details about KMeans We applied the Kmeans clustering algorithm on an extensive dataset of ten million cell patches extracted from a single brain's segmentation to find around one thousand representative cell patches. The Kmeans procedure included two key steps: initialization with Kmeans++ and subsequent refinement. Kmeans++ was used to select 2,000 initial cluster centroids. Then these centroids were used to aggregate similar cell patches into clusters. The mean of each cluster was computed as the final set of representative cell patches. Clusters containing fewer than 5 samples were excluded. Figure 1 shows samples of the original cell images and of representative cell patches selected using K-means after normalizing rotation.

Details about Diffusion map We used the public implementation of DM [1]. The parameters used in this function were as follows:

```
n_evecs - Number of diffusion map eigenvectors: 100,
k - Number of nearest neighbors to construct the kernel: 100,
epsilon: 5000 (chosen based on the typical cell size),
alpha: 1.0, neighbor_params: {'n_jobs': -1, 'algorithm': 'ball_tree'}.
```

Details about RMS-based optimization The mathematical formulation of finding a linear transformation between two different diffusion maps can be described as follows: Given a set of vector pairs $(a_1, b_1), \ldots, (a_n, b_n)$ where each of the vectors a_i, b_i are in a d dimensional space R^d, find an offset vector $\mu \in r^d$

[1] `https://github.com/DiffusionMapsAcademics/pyDiffMap`

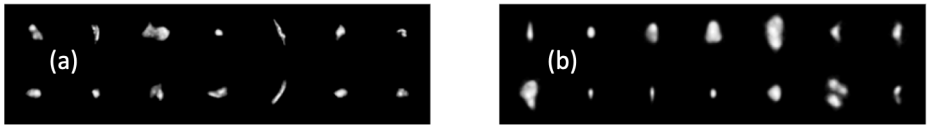

Fig. 1. (a) Original cell patches, **(b)** Cell patches selected using K-means after normalizing rotation.

and a linear transformation M which is a $d \times d$ matrix so that the following cost function is minimized:

$$cost(\mu, M) = \frac{1}{n} \sum_{i=1}^{n} ||\mu + Ma_i - b_i||_2^2 \tag{1}$$

We first compute the hessian matrix of 1 with respect to μ and M to see if the problem can be directly solved.

$$\frac{\partial^2 cost(\mu, M)}{\partial \mu \partial \mu^T} = 2I \rightarrow p.d \tag{2}$$

$$\frac{\partial^2 cost(\mu, M)}{\partial M \partial M^T} = \frac{2}{n} \sum_{i=1}^{n} a_i a_i^T \rightarrow p.s.d \tag{3}$$

The results show that both hessian matrices are Positive Semi-definite matrices and thus we can find the solution by setting derivatives to 0. We have:

$$(\frac{1}{n} \sum_{i=1}^{n} a_i a_i^T - \frac{1}{n} \sum_{i=1}^{n} a_i \times \frac{1}{n} \sum_{i=1}^{n} a_i^T) M^T = \frac{1}{n} \sum_{i=1}^{n} a_i b_i^T - \frac{1}{n} \sum_{i=1}^{n} a_i \times \frac{1}{n} \sum_{i=1}^{n} b_i^T \tag{4}$$

Details about 10 manually designed features The 10 manually designed features encompass the following: width, height, and area of a cell; rotation angle along with its confidence level; mean and standard deviation of pixel intensities within the cell image; the size of the cell patch, and standard deviation of horizontal and vertical coordinates of all cell pixels in the cell patch.

Details about XGBoost classifiers The parameters used to train our XGBoost classifiers were as follows:

```
params: {'max_depth':3, 'eta': 0.2,
        'objective':'binary:logistic', 'num_class':1},
num_boost_round: 100.
```