Supplementary Material HistoCartography: A Toolkit for Graph Analytics in Digital Pathology

Guillaume Jaume GJA@ZURICH.IBM.COM
*IBM Research, Zurich – EPFL, Lausanne, *equal contribution*

Pushpak Pati* PUS@ZURICH.IBM.COM
IBM Research, Zurich – ETH, Zurich

Valentin Anklin ANKLINV@STUDENT.ETHZ.CH
ETH, Zurich

Antonio Foncubierta FRA@ZURICH.IBM.COM
IBM Research, Zurich

Maria Gabrani MGA@ZURICH.IBM.COM
IBM Research, Zurich

HistoCartography Ecosystem

HistoCartography core functionalities can be tested using a set of examples available at `https://github.com/histocartography/histocartography/blob/main/examples/`. Examples include stain normalization, cell- and tissue-graph generation, cell-graph explanation, and feature cube extraction. Additionally, a Jupyter Notebook presenting the library interpretability and explainability capabilities can be found at `https://github.com/maragraziani/interpretAI_DigiPath/tree/main/hands-on-session-2`. Individual functions are thoroughly unit tested (88% unit test coverage), and can be accessed at `https://github.com/histocartography/histocartography/tree/main/test`. The code documentation, which provides a user-friendly approach to understanding HistoCartography architecture and modules can be accessed at `https://histocartography.github.io/histocartography/`. Finally, papers using HistoCartography can be found at `https://github.com/histocartography`.

HistoCartography Syntax

In this section, we introduce the syntax to implement the functionalities of HistoCartography. Figure 1 presents code snippets to implement Vahadane stain normalization and tissue mask detection. Figure 2 shows the syntax for building cell- and tissue-graphs. Noticeably, these functionalities require only ten lines of code by using HistoCartography, which could have otherwise required a few hundred lines. In Figure 3, we present the syntax to declare and run a cell- and tissue-graph model. All the model parameters, *e.g.*, Graph Neural Network (GNN) type, number of GNN layers, can be adapted and fine-tuned using

a configuration file. Finally, Figure 4 shows code snippets to use the graph explainability modules. All explainers follow a similar syntax with the same input and output types, making implementation and integration straightforward.

Stain Normalization Code

```python
from PIL import Image
import numpy as np
from histocartography.preprocessing import (
  VahadaneStainNormalizer
)

# define Vahadane stain normalizer
normalizer = VahadaneStainNormalizer(
  target_path='target.png'
)

# process image
img = np.array(Image.open('17B0031061.png'))
norm_image = normalizer.process(img)
```

Tissue Mask Detection Code

```python
from PIL import Image
from histocartography.preprocessing import (
  GaussianTissueMask
)

# define tissue detector
tissue_detector = GaussianTissueMask(
  kernel_size=20,
  sigma=10
)

# process image
img = np.array(Image.open('17B0031061.png'))
mask = tissue_detector.process(img)
```

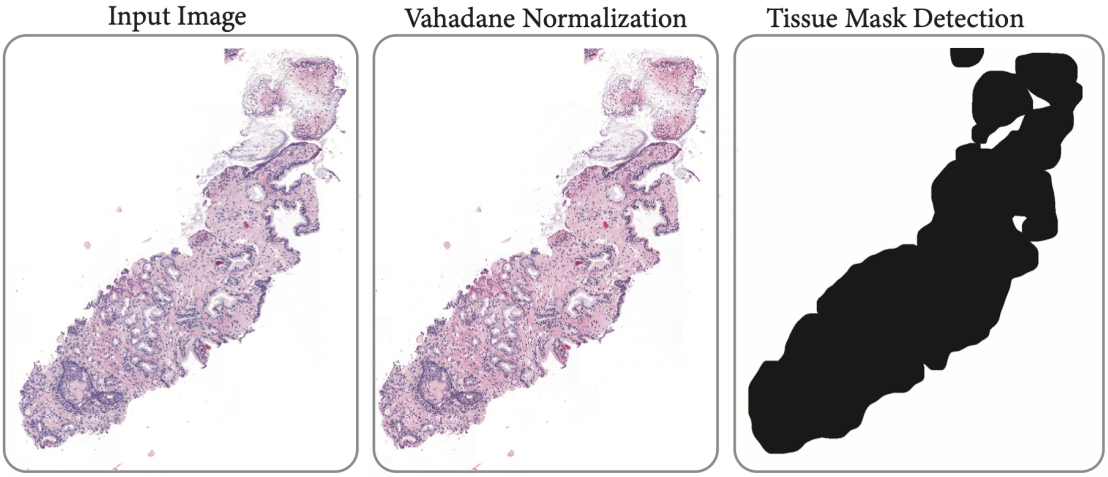

Figure 1: Implementation of Vahadane stain normalization (left) and tissue mask detection (right) with the *Preprocessing* functionalities in the HistoCartography Application Programming Interface (API).

Handcrafted Feature Extraction

In this section, we provide a comprehensive list of morphological and topological features which can be extracted per-entity by HistoCartography. Morphological features include shape, size and texture properties, namely, entity area, convex area, eccentricity, equivalent diameter, euler number, length of the major and minor axis, orientation, perimeter, solidity, convex hull perimeter, roughness, shape factor, ellipticity, roudness. Texture properties are based on gray-level co-occurrence matrices (GLCM). Specifically, we extract the GLCM contrast, dissimilarity, homogeneity, energy, angular speed moment and dispersion. The topological features are based on the entity density computed as the mean and variance of entity crowdedness. These features can be computed for the most important set of

Cell Graph Generation Code

```python
from PIL import Image
from histocartography.preprocessing import (
  NucleiExtractor,
  DeepFeatureExtractor,
  KNNGraphBuilder
)

# define nuclei
nuclei_detector = NucleiExtractor()

# define feature extractor
feats_extractor = DeepFeatureExtractor(
    architecture='resnet34',
    patch_size=72,
    resize_size=224
)

# define graph builder
knn_graph_builder = KNNGraphBuilder(
  k=5,
  thresh=50
)

# process image
img = np.array(Image.open('image.png'))
nuclei, _ = nuclei_detector.process(img)
feats = feats_extractor.process(img, nuclei)
graph = knn_graph_builder.process(nuclei, feats)
```

Tissue Graph Generation Code

```python
from PIL import Image
from histocartography.preprocessing import (
    ColorMergedSuperpixelExtractor,
    DeepFeatureExtractor,
    RAGGraphBuilder
)

# define super-pixels
superpx_detector = ColorMergedSuperpixelExtractor(
    nr_superpixels=400,
    downsampling_factor=4
)

# define feature extractor
feats_extractor = DeepFeatureExtractor(
    patch_size=144,
    resize_size=224
)

# define graph builder
rag_graph_builder = RAGGraphBuilder()

# process image
img = np.array(Image.open('image.png'))
superpxs, _ = superpx_detector.process(img)
feats = feats_extractor.process(img, superpxs)
graph = rag_graph_builder.process(superpxs, feats)
```

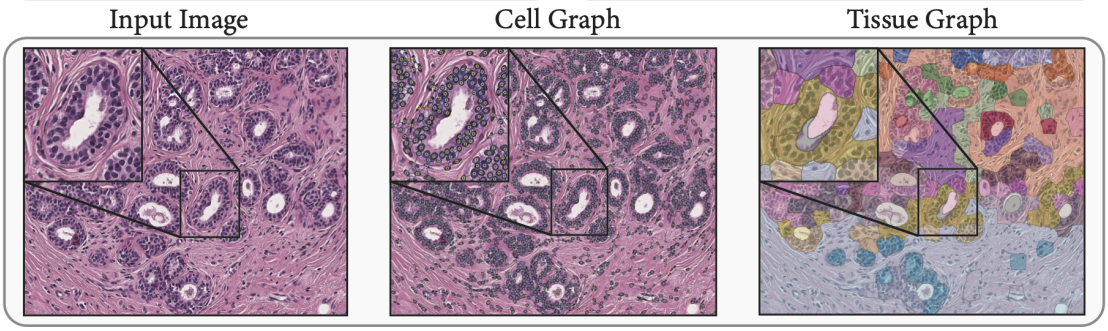

Figure 2: Implementation of cell-graph (left) and tissue-graph (right) generation using the graph builders in HistoCartography.
entities highlighted by the graph explainability techniques, and utilized along with prior pathological knowledge to interpret the trained entity-graph models.

Future of HistoCartography

HistoCartography development is only in its infancy, bugs will be fixed as people use it, new modules will be developed as the community develops novel graph-based methods and algorithms. Nevertheless, HistoCartography can already be used for developing new projects. Thanks to its modularity, pipelines can be developed by only partially using HistoCartography, *e.g.*, only for building tissue-graphs, while novel components that require more flexibility and control can be developed on the side, *e.g.*, for developing new models.

Cell Graph Model

```python
import yaml
from dgl.data.utils import (
  load_graphs
)
from histocartography.ml import (
  CellGraphModel
)

# load model configurations
cfg = yaml.safe_load(open('cg_cfg.yml', 'r'))

# define cell graph model
model = CellGraphModel(
    gnn_params=cfg['gnn_params'],
    classification_params=cfg['cls_params'],
    node_dim=512,
    num_classes=3
)

# load cell graph
cg, _ = load_graphs('cg.bin')

# forward pass
logits = model(cg)
```

Tissue Graph Model

```python
import yaml
from dgl.data.utils import (
  load_graphs
)
from histocartography.ml import (
  TissueGraphModel
)

# load model configurations
cfg = yaml.safe_load(open('tg_cfg.yml', 'r'))

# define tissue graph model
model = TissueGraphModel(
    gnn_params=cfg['gnn_params'],
    classification_params=cfg['cls_params'],
    node_dim=512,
    num_classes=3
)

# load tissue graph
tg, _ = load_graphs('tg.bin')

# forward pass
logits = model(tg)
```

Figure 3: Implementation of the cell- (left) and tissue- graph (right) model by using the Machine Learning (ML) modules in the HistoCartography API

Cell Graph Explainer Code

```python
from histocartography.interpretability import (
  GraphGradCAMExplainer,
  GraphGradCAMPPExplainer,
  GraphPruningExplainer,
  GraphLRPExplainer
)

# load pretrained model
model = CellGraphModel(config['gnn_params'], config['cls_params'], 512, pretrained=True)

# load cell graph
graph, _ = load_graphs('291_dcis_18.bin')

# define graph explainers
grad_cam_explainer = GraphGradCAMExplainer(model=model)
grad_campp_explainer = GraphGradCAMPPExplainer(model=model)
gnn_explainer = GraphPruningExplainer(model=model)
graph_lrp_explainer = GraphLRPExplainer(model=model)

# explain cell graph
grad_cam_scores, _ = grad_cam_explainer.process(graph)
grad_campp_scores, _ = grad_campp_explainer.process(graph)
gnn_explainer_scores, _ = gnn_explainer.process(graph)
graph_lrp_scores, _ = graph_lrp_explainer.process(graph)
```

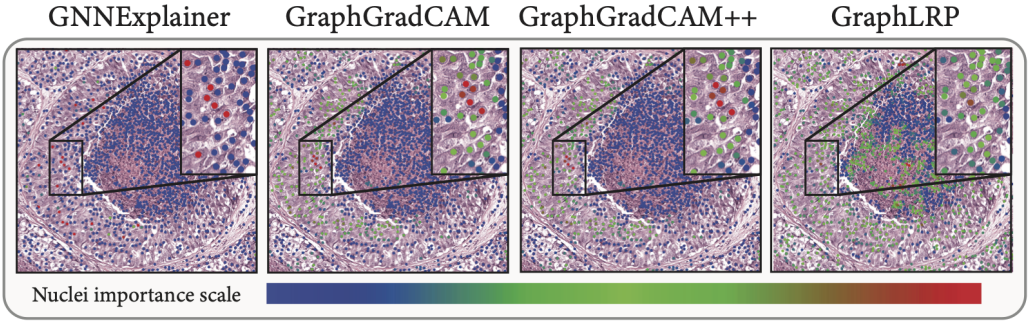

Figure 4: Implementation of graph explainers in HistoCartography. The most important nodes are marked in red and the least important ones in blue.

Table 1: Extended version of Table 2. Reported time to run HistoCartography core functionalities. CPU-only experiments were run on a single-core POWER8 processor, and GPU-compatible experiments were run on an NVIDIA P100 GPU. Time is reported in seconds.

Mod.		Function / Image type	Tumor RoI			Whole Slide Image (WSI)		
			1000^2	2500^2	5000^2	5000^2	7500^2	11000^2
Preprocessing	Standard	Vahadane Normalization	1.77	6.46	29.03	30.67	68.27	186.10
		Macenko Normalization	0.80	2.86	11.19	15.98	32.37	81.72
		Tissue Mast Detection	-	-	-	1.04	2.11	8.09
		Feature Cube Extraction	0.24	1.61	5.92	6.27	11.97	29.79
	CG	Nuclei Detection	3.03	12.93	47.66	-	-	-
		Nuclei Concept Extraction	2.95	6.52	27.94	-	-	-
		Deep Nuclei Feature Extraction	0.10	0.30	1.28	-	-	-
		k-Nearest Neighbors (k-NN) Graph Building	0.06	0.20	1.35	-	-	-
	TG	Super-pixel Detection	3.32	17.84	68.99	31.50	68.99	183.54
		Deep Tissue Feature Extraction	0.56	2.99	8.40	4.17	9.96	20.54
		RAG Graph Building	0.12	2.04	25.6	6.33	19.98	85.73
ML		Cell-Graph Model	0.028	0.033	0.040	-	-	-
		Tissue-Graph Model	0.011	0.015	0.026	0.039	0.056	0.069
		HACT Model	0.034	0.041	0.057	-	-	-
Explainers	CG	GnnExplainer	12.00	13.09	35.33	-	-	-
		GraphGrad-CAM	0.011	0.022	0.035	-	-	-
		GraphGrad-CAM++	0.011	0.023	0.035	-	-	-
		GraphLRP	0.020	0.024	0.90	-	-	-
	TG	GnnExplainer	11.23	11.28	11.38	-	-	-
		GraphGrad-CAM	0.011	0.012	0.018	0.025	0.030	0.033
		GraphGrad-CAM++	0.011	0.013	0.018	0.026	0.030	0.033
		GraphLRP	0.011	0.014	0.016	0.079	0.085	0.089