SSAM analysis of mouse VISp, imaged by multiplexed smFISH

- Author: Jeongbin Park
- Date: 2019-10-12

Set plot parametes / define helper functions

[1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib_scalebar.scalebar import ScaleBar

from sklearn import preprocessing
import pickle
```

[2]:
```
# post-filtering parameter for cell-type map
filter_method = "local"
filter_params = {
    "block_size": 151,
    "method": "mean",
    "mode": "constant",
    "offset": 0.2
}
```

[3]:
```
# Helper function to load precomputed tSNE
def load_tsne(tsne_id):
    with open("zenodo/multiplexed_smFISH/tsne/%s.pkl"%tsne_id, "rb") as f:
        ds.tsne = pickle.load(f)
```

Load data

Load mRNA spot locations

[4]:
```
import numpy as np
from collections import defaultdict

pos_dic = defaultdict(lambda: [])

um_per_pixel = 0.1
xmin, ymin, zmin = 10000000, 10000000, 10000000
xmax, ymax, zmax = -10000000, -10000000, -10000000
with open("zenodo/multiplexed_smFISH/raw_data/smFISH_MCT_CZI_Panel_0_spot_table.
 ↪csv") as f:
    f.readline()
    for line in f:
        e = line.strip().split(',')
        x, y, z, g = e[1], e[2], e[3], e[-1]
        x, y, z = [float(e) for e in [x, y, z]]
```

```
        if x > xmax:
            xmax = x
        if y > ymax:
            ymax = y
        if z > zmax:
            zmax = z
        if x < xmin:
            xmin = x
        if y < ymin:
            ymin = y
        if z < zmin:
            zmin = z

with open("zenodo/multiplexed_smFISH/raw_data/smFISH_MCT_CZI_Panel_0_spot_table.
 ↪csv") as f:
    f.readline()
    for line in f:
        e = line.strip().split(',')
        x, y, z, g = e[1], e[2], e[3], e[-1]
        x, y, z = [float(e) for e in [x, y, z]]
        x -= xmin
        y -= ymin
        z -= zmin
        x, y = [e*um_per_pixel + 10 for e in [x, y]]
        z = z * um_per_pixel
        pos_dic[g].append([x, y])

for g in pos_dic:
    pos_dic[g] = np.array(pos_dic[g])
```

SSAM analysis

Run KDE and select representative vectors

Initilize SSAM and run KDE

[5]:
```
import ssam
```

[6]:
```
width = (xmax - xmin) * um_per_pixel + 10
height = (ymax - ymin) * um_per_pixel + 10
ncores = 10 # Number of cores used for kernel density estimation
```

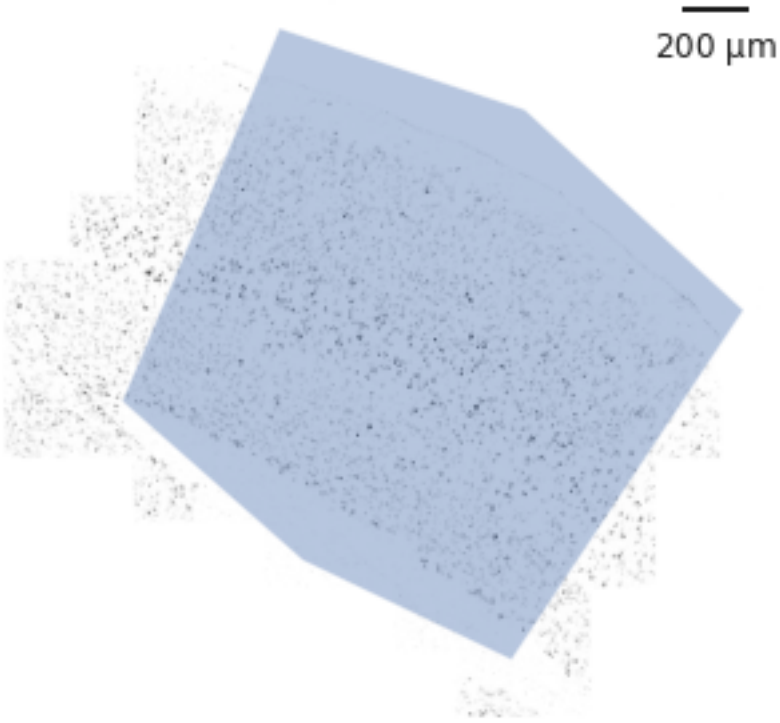
[7]:
```
all_genes = list(pos_dic.keys())
ds = ssam.SSAMDataset(all_genes, [pos_dic[gene] for gene in all_genes], width,␣
 ↪height)
```

```
analysis = ssam.SSAMAnalysis(ds, ncores=10, save_dir="zenodo/multiplexed_smFISH/
↪kde", verbose=True)
```

[8]:
```
analysis.run_kde(bandwidth=2.5, use_mmap=False)
```

Select VISp area

[9]:
```
plt.figure(figsize=[5, 5])
ds.plot_l1norm(rotate=1, cmap="Greys")
xy = np.array([[1535,  90],
               [ 795,  335],
               [ 135,  940],
               [ 835, 1995],
               [1465, 1695],
               [2010, 1215]]) # VISp area manually curated

from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection

patch = Polygon(xy, True)
p = PatchCollection([patch], alpha=0.4)
plt.gca().add_collection(p)

scalebar = ScaleBar(1, 'um') # 1 pixel = 1um
plt.gca().add_artist(scalebar)
plt.tight_layout()

plt.axis('off')
pass
```



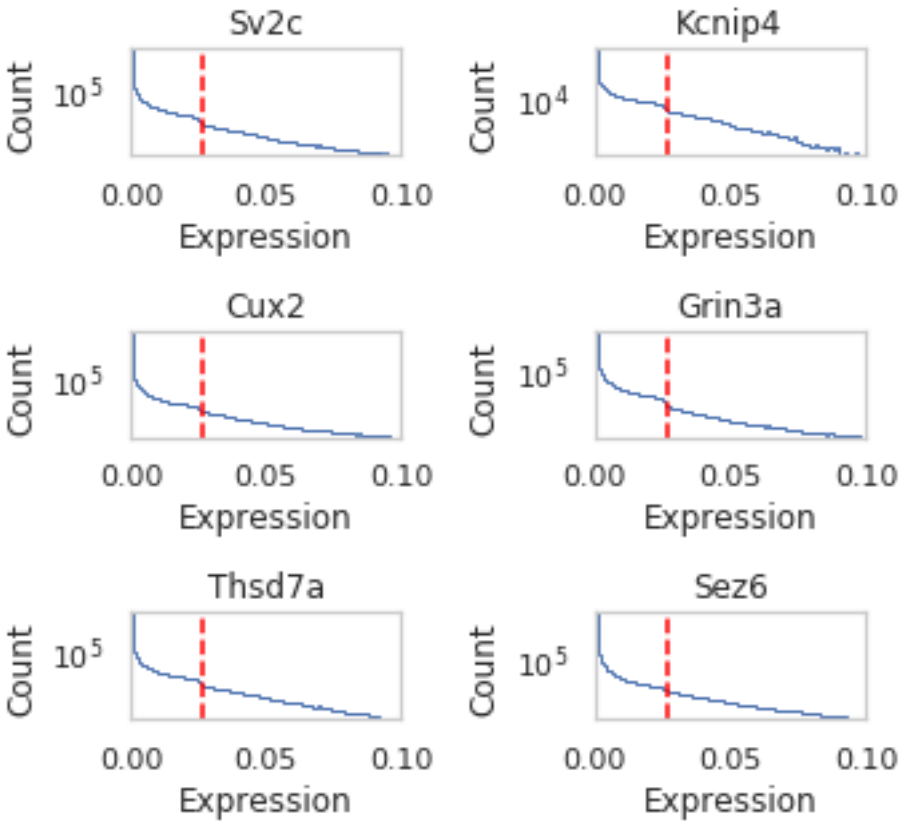
Make input/output mask for VISp region

[10]:
```
from matplotlib.path import Path

x, y = np.meshgrid(np.arange(ds.vf.shape[0]), np.arange(ds.vf.shape[1]))
x, y = x.flatten(), y.flatten()
points = np.vstack((x,y)).T

path = Path(xy)
input_mask = path.contains_points(points)
output_mask = input_mask = input_mask.reshape((ds.vf.shape[1], ds.vf.shape[0],␣
↪1)).swapaxes(0, 1)
```

Select local maxima of gene expression in the vector field

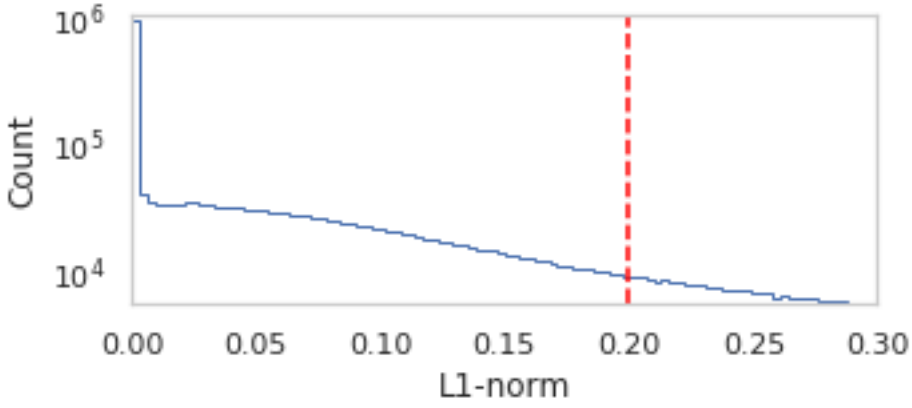
[11]:
```
exp_thres = 0.027
viewport = 0.1
gindices = np.arange(len(ds.genes))
np.random.shuffle(gindices)
plt.figure(figsize=[5, 7])
for i, gidx in enumerate(gindices[:6], start=1):
    ax = plt.subplot(5, 2, i)
    n, bins, patches = ax.hist(ds.vf[..., gidx][np.logical_and(ds.vf[..., gidx]␣
↪> 0, ds.vf[..., gidx] < viewport)], bins=100, log=True, histtype=u'step')
```

```
    ax.set_xlim([0, viewport])
    ax.set_ylim([n[0], n[-1]])
    ax.axvline(exp_thres, c='red', ls='--')
    ax.set_title(ds.genes[gidx])
    ax.set_xlabel("Expression")
    ax.set_ylabel("Count")
plt.tight_layout()
pass
```



[12]:
```
norm_thres = 0.2
gidx = 0
plt.figure(figsize=[5, 2])
#plt.hist(ds.vf[..., gidx][ds.vf[..., gidx] > 0], bins=100, log=True)
n, _, _ = plt.hist(ds.vf_norm[np.logical_and(ds.vf_norm > 0, ds.vf_norm < 0.
↪3)], bins=100, log=True, histtype='step')
ax = plt.gca()
ax.axvline(norm_thres, c='red', ls='--')
ax.set_xlabel("L1-norm")
ax.set_ylabel("Count")
```

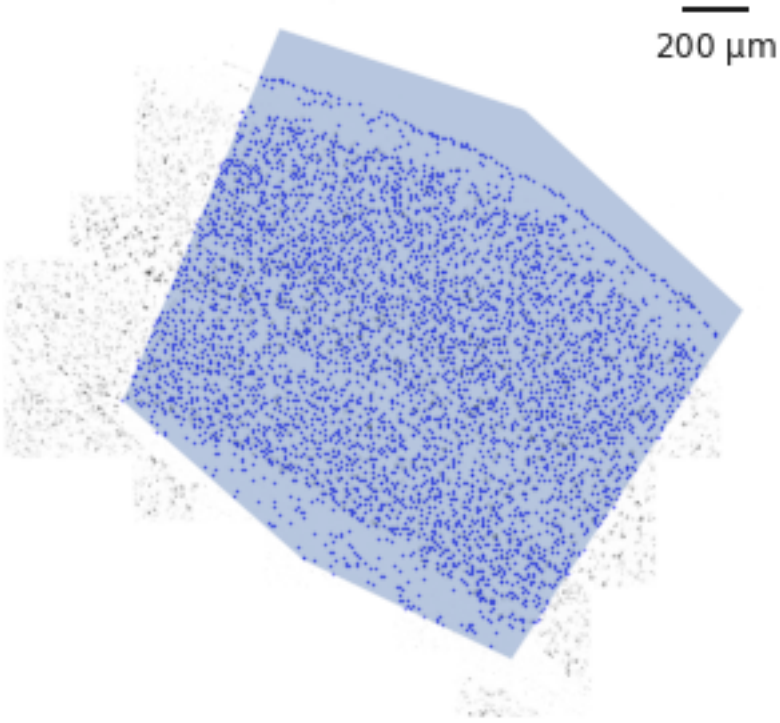
```
plt.xlim([0, 0.3])
plt.ylim([np.min(n), np.max(n) + 100000])
pass
```



[13]:
```
analysis.find_localmax(search_size=3, min_norm=norm_thres,␣
↪min_expression=exp_thres, mask=input_mask)
```

```
Found 4586 local max vectors.
```

[14]:
```
plt.figure(figsize=[5, 5])
ds.plot_l1norm(cmap="Greys", rotate=1)
ds.plot_localmax(c="Blue", rotate=1, s=0.1)

patch = Polygon(xy, facecolor="black", edgecolor="red", linewidth=10, ls="-")
p = PatchCollection([patch], alpha=0.4)
plt.gca().add_collection(p)

scalebar = ScaleBar(1, 'um') # 1 pixel = 1um
plt.gca().add_artist(scalebar)
#plt.show()
plt.tight_layout()

plt.axis('off')

plt.show()
```



Normalize local maxima vectors and vector field

[15]:
```
# this requires local R installation with packages 'sctransform' and 'feather'
analysis.normalize_vectors_sctransform()
```

SSAM guided mode: using scRNA-seq data (Tasic *et al.* 2018)

Load scRNA-seq data

[16]:
```
import pandas as pd
scrna_cl = pd.read_feather("zenodo/multiplexed_smFISH/raw_data/
 ↪scrna_data_tasic_2018/cl.feather")
scrna_cl_df = pd.read_feather("zenodo/multiplexed_smFISH/raw_data/
 ↪scrna_data_tasic_2018/cl_df.feather")
scrna_genes = pd.read_feather("zenodo/multiplexed_smFISH/raw_data/
 ↪scrna_data_tasic_2018/genes.feather")
scrna_counts = pd.read_feather("zenodo/multiplexed_smFISH/raw_data/
 ↪scrna_data_tasic_2018/counts.feather")
```

[17]:
```
dendrogram_order = [
    'L2/3 IT VISp Rrad',
    'L2/3 IT VISp Adamts2',
```

```
    'L2/3 IT VISp Agmat',
    'L4 IT VISp Rspo1',
    'L5 IT VISp Hsd11b1 Endou',
    'L5 IT VISp Whrn Tox2',
    'L5 IT VISp Batf3',
    'L5 IT VISp Col6a1 Fezf2',
    'L5 IT VISp Col27a1',
    'L6 IT VISp Penk Col27a1',
    'L6 IT VISp Penk Fst',
    'L6 IT VISp Col23a1 Adamts2',
    'L6 IT VISp Col18a1',
    'L6 IT VISp Car3',
    'L5 PT VISp Chrna6',
    'L5 PT VISp Lgr5',
    'L5 PT VISp C1ql2 Ptgfr',
    'L5 PT VISp C1ql2 Cdh13',
    'L5 PT VISp Krt80',
    'L5 NP VISp Trhr Cpne7',
    'L5 NP VISp Trhr Met',
    'L6 CT Nxph2 Sla',
    'L6 CT VISp Krt80 Sla',
    'L6 CT VISp Nxph2 Wls',
    'L6 CT VISp Ctxn3 Brinp3',
    'L6 CT VISp Ctxn3 Sla',
    'L6 CT VISp Gpr139',
    'L6b Col8a1 Rprm',
    'L6b VISp Mup5',
    'L6b VISp Col8a1 Rxfp1',
    'L6b P2ry12',
    'L6b VISp Crh',
    'L6b Hsd17b2',
    'Lamp5 Krt73',
    'Lamp5 Fam19a1 Pax6',
    'Lamp5 Fam19a1 Tmem182',
    'Lamp5 Ntn1 Npy2r',
    'Lamp5 Plch2 Dock5',
    'Lamp5 Lsp1',
    'Lamp5 Lhx6',
    'Sncg Slc17a8',
    'Sncg Vip Nptx2',
    'Sncg Gpr50',
    'Sncg Vip Itih5',
    'Serpinf1 Clrn1',
    'Serpinf1 Aqp5 Vip',
    'Vip Igfbp6 Car10',
    'Vip Igfbp6 Pltp',
    'Vip Lmo1 Fam159b',
```

```
    'Vip Lmo1 Myl1',
    'Vip Igfbp4 Mab21l1',
    'Vip Arhgap36 Hmcn1',
    'Vip Gpc3 Slc18a3',
    'Vip Ptprt Pkp2',
    'Vip Rspo4 Rxfp1 Chat',
    'Vip Lect1 Oxtr',
    'Vip Rspo1 Itga4',
    'Vip Chat Htr1f',
    'Vip Pygm C1ql1',
    'Vip Crispld2 Htr2c',
    'Vip Crispld2 Kcne4',
    'Vip Col15a1 Pde1a',
    'Sst Chodl',
    'Sst Mme Fam114a1',
    'Sst Tac1 Htr1d',
    'Sst Tac1 Tacr3',
    'Sst Calb2 Necab1',
    'Sst Calb2 Pdlim5',
    'Sst Nr2f2 Necab1',
    'Sst Myh8 Etv1',
    'Sst Chrna2 Glra3',
    'Sst Myh8 Fibin',
    'Sst Chrna2 Ptgdr',
    'Sst Tac2 Myh4',
    'Sst Hpse Sema3c',
    'Sst Hpse Cbln4',
    'Sst Crhr2 Efemp1',
    'Sst Crh 4930553C11Rik',
    'Sst Esm1',
    'Sst Tac2 Tacstd2',
    'Sst Rxfp1 Eya1',
    'Sst Rxfp1 Prdm8',
    'Sst Nts',
    'Pvalb Gabrg1',
    'Pvalb Th Sst',
    'Pvalb Calb1 Sst',
    'Pvalb Akr1c18 Ntf3',
    'Pvalb Sema3e Kank4',
    'Pvalb Gpr149 Islr',
    'Pvalb Reln Itm2a',
    'Pvalb Reln Tac1',
    'Pvalb Tpbg',
    'Pvalb Vipr2',
    'Meis2 Adamts19',
    'CR Lhx5',
    'Astro Aqp4',
```

```
    'OPC Pdgfra Grm5',
    'OPC Pdgfra Ccnb1',
    'Oligo Rassf10',
    'Oligo Serpinb1a',
    'Oligo Synpr',
    'VLMC Osr1 Cd74',
    'VLMC Spp1 Hs3st6',
    'VLMC Osr1 Mc5r',
    'VLMC Spp1 Col15a1',
    'Peri Kcnj8',
    'SMC Acta2',
    'Endo Ctla2a',
    'Endo Cytl1',
    'PVM Mrc1',
    'Microglia Siglech'
]
```

[18]:
```
scrna_clusters = scrna_cl['cluster_id']
```

[19]:
```
scrna_cl_dic = dict(zip(scrna_cl['cell_id'], scrna_cl['cluster_id']))
scrna_cl_metadata_dic = dict(zip(
    scrna_cl_df['cluster_id'],
    zip(scrna_cl_df['cluster_label'],
        scrna_cl_df['cluster_color'], )
))
```

[20]:
```
qc_gene_indices = np.sum(scrna_counts > 0, axis=1) > 5
scrna_genes_qc = np.array(scrna_genes)[qc_gene_indices]
```

[21]:
```
scrna_counts_qc = np.array(scrna_counts).T[:, qc_gene_indices]
```

[22]:
```
# Normalize it with sctransform
scrna_data_normalized = np.array(ssam.run_sctransform(scrna_counts_qc)[0])
```

[23]:
```
selected_genes_idx = [list(scrna_genes_qc).index(g) for g in ds.genes]
scrna_uniq_clusters = np.unique(scrna_clusters)
scrna_centroids = []
for cl in scrna_uniq_clusters:
    scrna_centroids.append(np.mean(scrna_data_normalized[:,␣
 ↪selected_genes_idx][scrna_clusters == cl], axis=0))
```

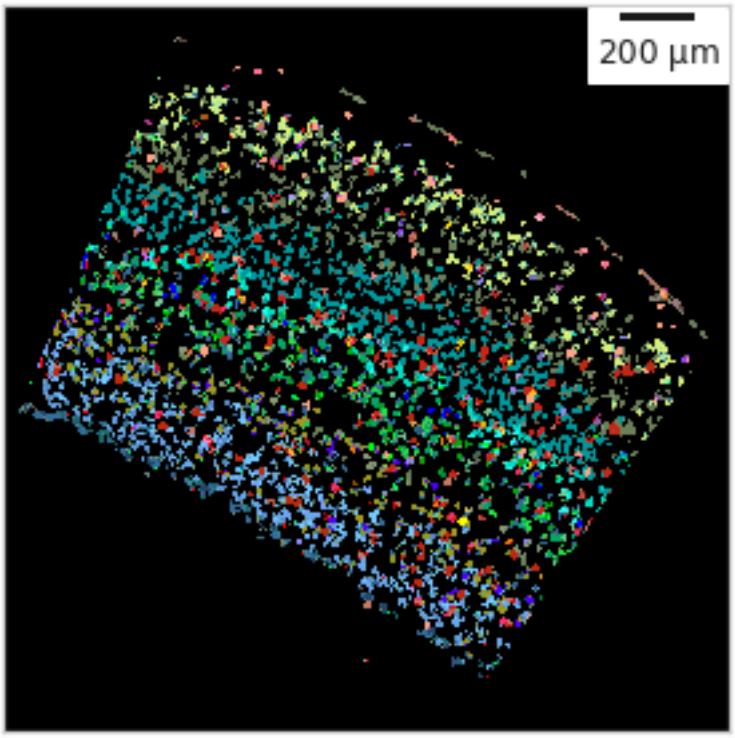
Map it to the vector field

[24]:
```
analysis.map_celltypes(scrna_centroids)
analysis.filter_celltypemaps(min_norm=filter_method,␣
 ↪filter_params=filter_params, min_r=0.3, output_mask=output_mask) #␣
 ↪post-filter cell-type map to remove spurious pixels
```

[25]:
```
scrna_uniq_labels = [scrna_cl_metadata_dic[i][0] for i in scrna_uniq_clusters]
scrna_colors = [scrna_cl_metadata_dic[i][1] for i in scrna_uniq_clusters]
```

[26]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltypes_map(rotate=1, colors=scrna_colors, set_alpha=False)
plt.xlim([2050, 150])
plt.ylim([2050, 150])
plt.gca().get_xaxis().set_visible(False)
plt.gca().get_yaxis().set_visible(False)
scalebar = ScaleBar(1, 'um') # 1 pixel = 1um
plt.gca().add_artist(scalebar)
```

[26]:
```
<matplotlib_scalebar.scalebar.ScaleBar at 0x2aafd4471dd0>
```



SSAM *de novo* mode

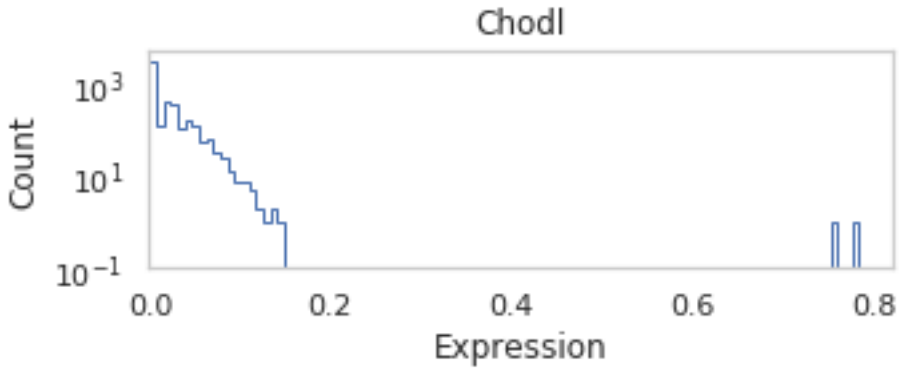
Cluster vectors

[27]:
```
analysis.cluster_vectors(min_cluster_size=0, pca_dims=22, resolution=0.15,␣
↪metric='correlation')
```

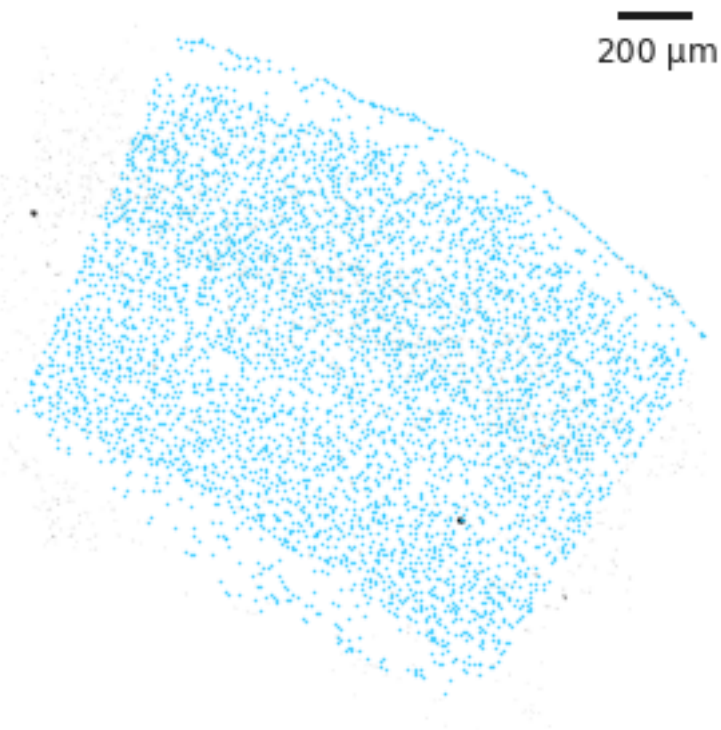
```
Found 30 clusters
```

Rescue cluster expressing high Chodl

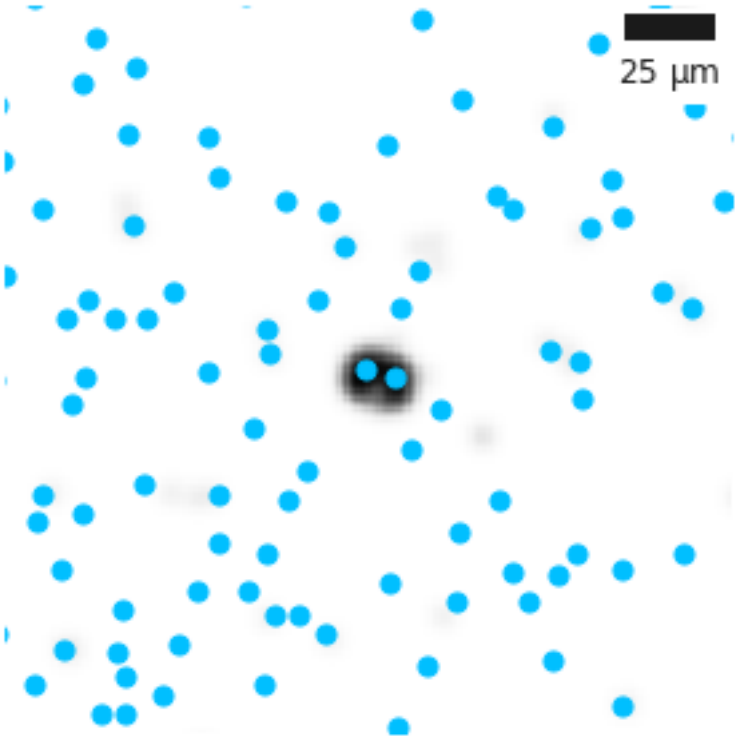
[28]:
```
# check vectors highly expressing Chodl
lm_vectors = ds.vf[ds.local_maxs[0], ds.local_maxs[1]].reshape([-1, len(ds.
 ↪genes)])
#for g in ds.genes:
g = 'Chodl'
plt.figure(figsize=[5, 1.5])
n, _, _ = plt.hist(lm_vectors[:, ds.genes.index(g)], bins=100, log=True,␣
 ↪histtype='step')
ax = plt.gca()
ax.set_xlabel("Expression")
ax.set_ylabel("Count")
plt.title(g)
plt.xlim(left=0)
pass
```



[29]:
```
# check whether the vectors are clustered properly
plt.figure(figsize=[5, 5])
plt.imshow(np.log(ds.vf[..., 0, ds.genes.index('Chodl')].T + 0.1), cmap='Greys')
plt.scatter(ds.local_maxs[0], ds.local_maxs[1], s=0.1, color="deepskyblue")
scalebar = ScaleBar(1, 'um') # 1 pixel = 1um
plt.gca().add_artist(scalebar)

plt.xlim([2050, 150])
plt.ylim([2050, 150])
plt.axis('off')
pass
```



[30]:
```
# check whether the vectors are clustered properly
plt.figure(figsize=[5, 5])
plt.imshow(np.log(ds.vf[..., 0, ds.genes.index('Chodl')].T + 0.1), cmap='Greys')
plt.scatter(ds.local_maxs[0], ds.local_maxs[1], s=50, color="deepskyblue")
scalebar = ScaleBar(1, 'um', height_fraction=0.035) # 1 pixel = 1um
plt.gca().add_artist(scalebar)

plt.xlim([950, 750])
plt.ylim([1600, 1400])
plt.axis('off')
pass
```

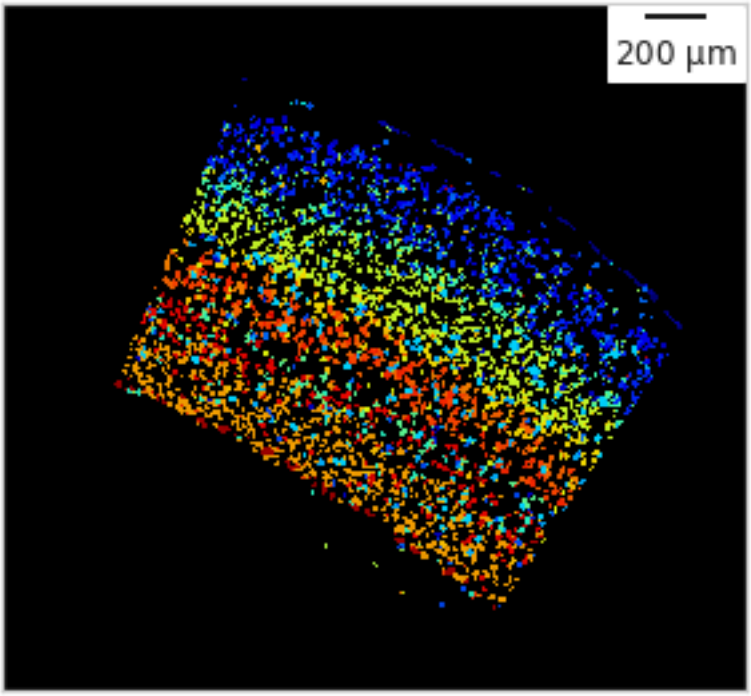


[31]:
```
analysis.rescue_cluster(['Chodl'], [0.5])
```

Generate cell type maps

[32]:
```
analysis.map_celltypes()
analysis.filter_celltypemaps(min_norm=filter_method,␣
↪filter_params=filter_params, min_r=0.6, output_mask=output_mask)
```

[33]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltypes_map(rotate=1)
plt.gca().get_xaxis().set_visible(False)
plt.gca().get_yaxis().set_visible(False)
scalebar = ScaleBar(1, 'um') # 1 pixel = 1um
plt.gca().add_artist(scalebar)
plt.show()
```

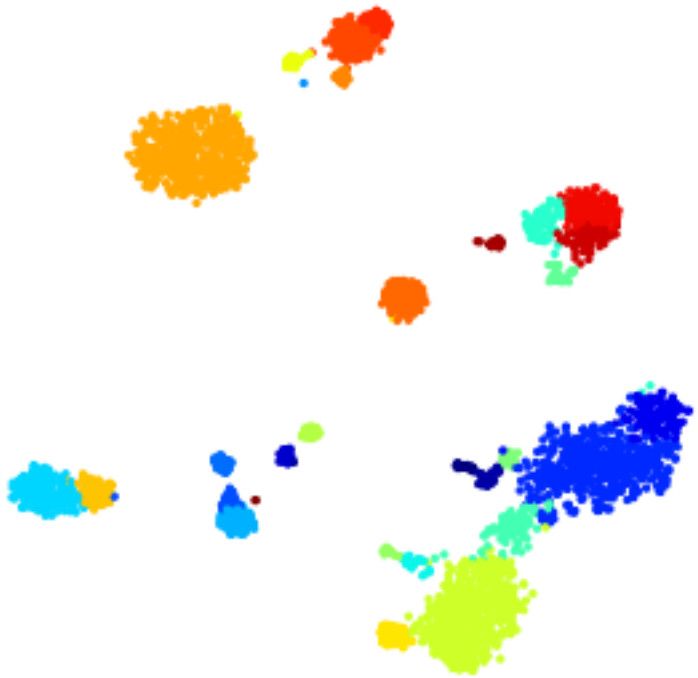


Draw embedding

[34]:
```
# Load precomputed tSNE
load_tsne("all_excluded_2d")
```

[35]:
```
plt.figure(figsize=[5, 5])
ds.plot_tsne(pca_dims=22, metric="correlation", s=5, run_tsne=False)
plt.gca().get_xaxis().set_visible(False)
plt.gca().get_yaxis().set_visible(False)
plt.axis('off')
```

[35]:
```
(-61.09699339223529, 43.75521177602435, -52.1520332185149, 73.0648864594817)
```



Draw diagnostic plots

[36]:
```
from scipy.stats import pearsonr, spearmanr

for idx in range(len(ds.centroids)):
    plt.figure(figsize=[50, 15])
    ds.plot_diagnostic_plot(idx, known_signatures=[
        ("scRNA-seq", scrna_uniq_labels, scrna_centroids, scrna_colors),
    ], correlation_methods=[
        ("r", pearsonr),
        ("rho", spearmanr)
    ])
    plt.tight_layout()
    plt.savefig('diagplots_multiplexed_smFISH/diagplot_centroid_%d.png'%idx)
    plt.close()
```

Merge/remove clusters

[37]:
```
denovo_labels = [
    "N/A",
    "VLMC",
    "Vip Arhgap36 Hmcn1 / Vip Igfbp4 Map2l1",
    "L2/3 IT Rrad",
```

```
    "N/A",
    "L2/3 IT Adamts2",
    "Sst Nts / Sst Rxfp1 Eya1",
    "Lamp5 Lsp1",
    "N/A",
    "Sst Crhr2 Efemp1 / Sst Esm1",

    "Pvalb Calb1 Sst / Pvalb Reln Tac1",
    "Astro Aqp4",
    "L6 IT Penk Fst",
    "L4 IT Superficial",
    "L5 IT Col27a1",
    "L2/3 IT Adamts2",
    "OPC",
    "Oligo",
    "L4 IT Rspo1",
    "L5 NP Trhr Met",

    "L5 IT Hsd11b1 Endou",
    "Pvalb Th Sst / Pvalb Reln Tac1",
    "L6 CT Ctxn3 Brinp3 / L6 CT Gpr139",
    "L5 PT Chrna6",
    "L5 IT Batf3",
    "L5 PT C1ql2 Cdh13",
    "L5 PT Krt80",
    "L6 IT Penk Col27a1",
    "L6 IT Penk Col27a1",
    "L6b Crh",

    "Sst Chodl",
]
```

[38]:
```
denovo_labels_final = []
exclude_indices = []
merge_indices = []

for idx, cl in enumerate(denovo_labels):
    if cl == 'N/A':
        exclude_indices.append(idx)
        continue
    if cl in denovo_labels_final:
        continue
    denovo_labels_final.append(cl)

for cl in np.unique(denovo_labels):
    if cl == 'N/A':
        continue
```

```
    mask = [cl == e for e in denovo_labels]
    if np.sum(mask) > 1:
        merge_indices.append(np.where(mask)[0])
```

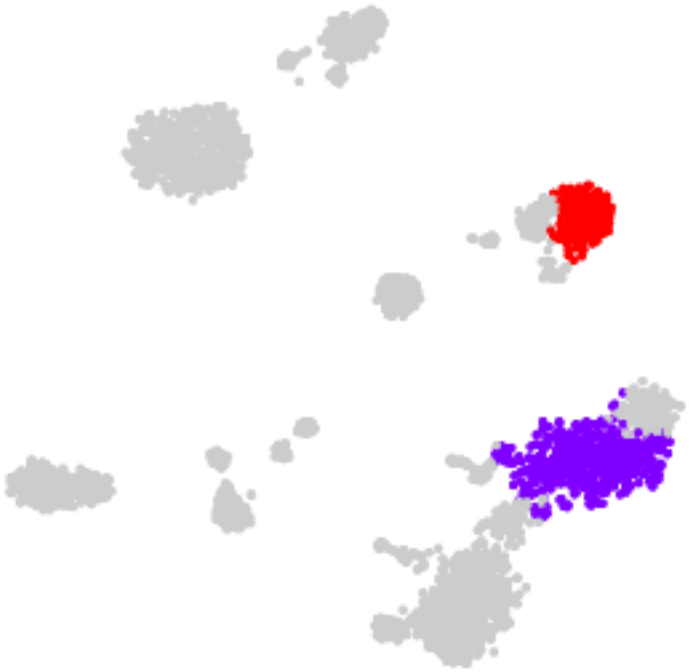
[39]:
```
# Plot removed clusters in tSNE
cmap = plt.get_cmap('jet')
jet_colors = cmap(np.array(list(range(len(ds.centroids)))) / (len(ds.centroids)␣
↪- 1))
tsne_colors = np.zeros_like(jet_colors)
tsne_colors[..., :] = [0.8, 0.8, 0.8, 1]
tsne_colors[exclude_indices] = [0, 0, 0, 1] #jet_colors[exclude_indices]
import matplotlib.patheffects as PathEffects
plt.figure(figsize=[5, 5])
ds.plot_tsne(pca_dims=33, metric="correlation", s=5, run_tsne=False,␣
↪colors=tsne_colors)
plt.axis('off')
```

[39]:
```
(-61.09699339223529, 43.75521177602435, -52.1520332185149, 73.0648864594817)
```

[40]:
```
# Plot merged clusters in tSNE
cmap = plt.get_cmap('rainbow')
jet_colors = cmap(np.array(list(range(len(merge_indices)))) /␣
↪(len(merge_indices) - 1))
```

```
plt.figure(figsize=[5, 5])
tsne_colors = np.zeros([len(ds.centroids), 4])
tsne_colors[..., :] = [0.8, 0.8, 0.8, 1]
for idx, mi in enumerate(merge_indices):
    tsne_colors[mi] = jet_colors[idx]
    ds.plot_tsne(pca_dims=33, metric="correlation", s=5, run_tsne=False,␣
 ↪colors=tsne_colors)
plt.axis('off')
```

[40]: (-61.55477591805327, 44.21299430184233, -52.71408845028678, 73.6269416912536)



[41]:
```
analysis.exclude_and_merge_clusters(exclude_indices, merge_indices,␣
 ↪centroid_correction_threshold=0.6)
```

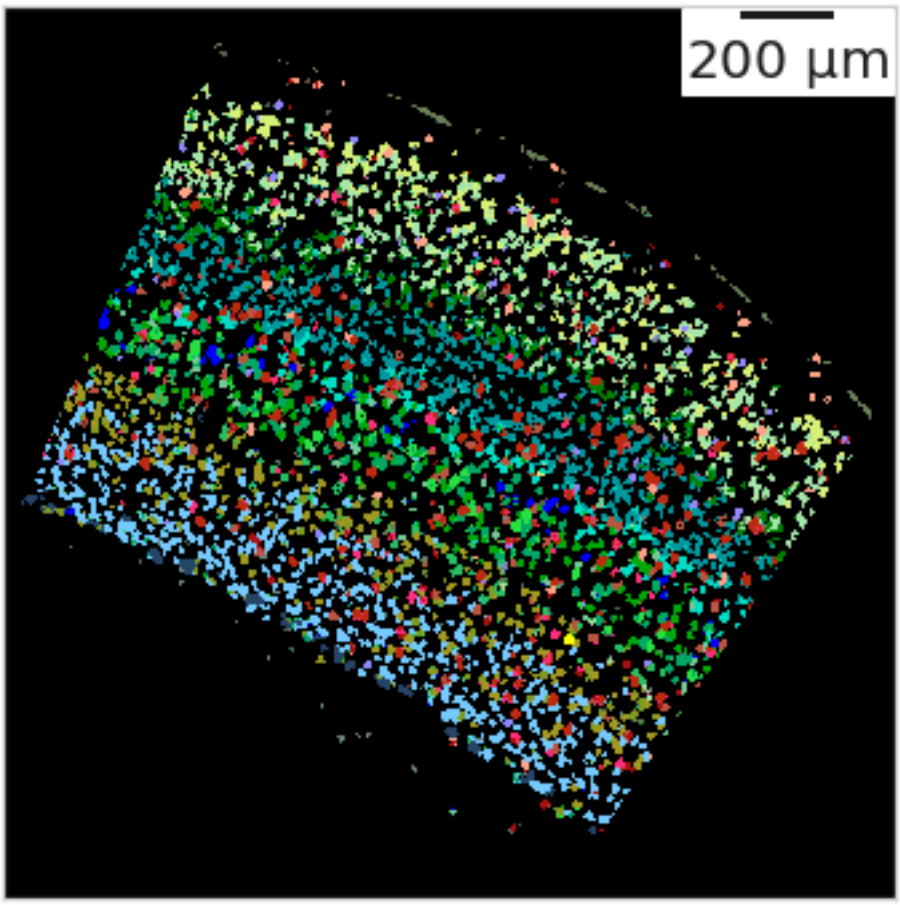
Draw cell-type map

[42]:
```
# Borrow clutster colors from scRNA-seq data
import matplotlib
denovo_celltype_colors = []
cluster_col_dic = dict(scrna_cl_metadata_dic.values())
for cl in denovo_labels_final:
    if ' / ' in cl:
        cl = cl.split(' / ')[0].rstrip()
```

```
    if cl == 'VLMC':
        cl = 'VLMC Spp1 Hs3st6'
    elif cl == 'OPC':
        cl = 'OPC Pdgfra Grm5'
    elif cl == 'Oligo':
        cl = 'Oligo Serpinb1a'
    elif cl == 'L6b Crh':
        cl = 'L6b VISp Crh'
    if ' IT ' in cl:
        cl = cl.replace(' IT ', ' IT VISp ')
    elif ' NP ' in cl:
        cl = cl.replace(' NP ', ' NP VISp ')
    elif ' CT ' in cl:
        cl = cl.replace(' CT ', ' CT VISp ')
    elif ' PT ' in cl:
        cl = cl.replace(' PT ', ' PT VISp ')
    if cl == "L4 IT VISp Superficial":
        col = '#008000'
    else:
        col = cluster_col_dic[cl]
    denovo_celltype_colors.append(col)
```

[43]:
```
analysis.map_celltypes()
analysis.filter_celltypemaps(min_norm=filter_method,␣
↪filter_params=filter_params, min_r=0.6, fill_blobs=True, min_blob_area=50,␣
↪output_mask=output_mask)
```

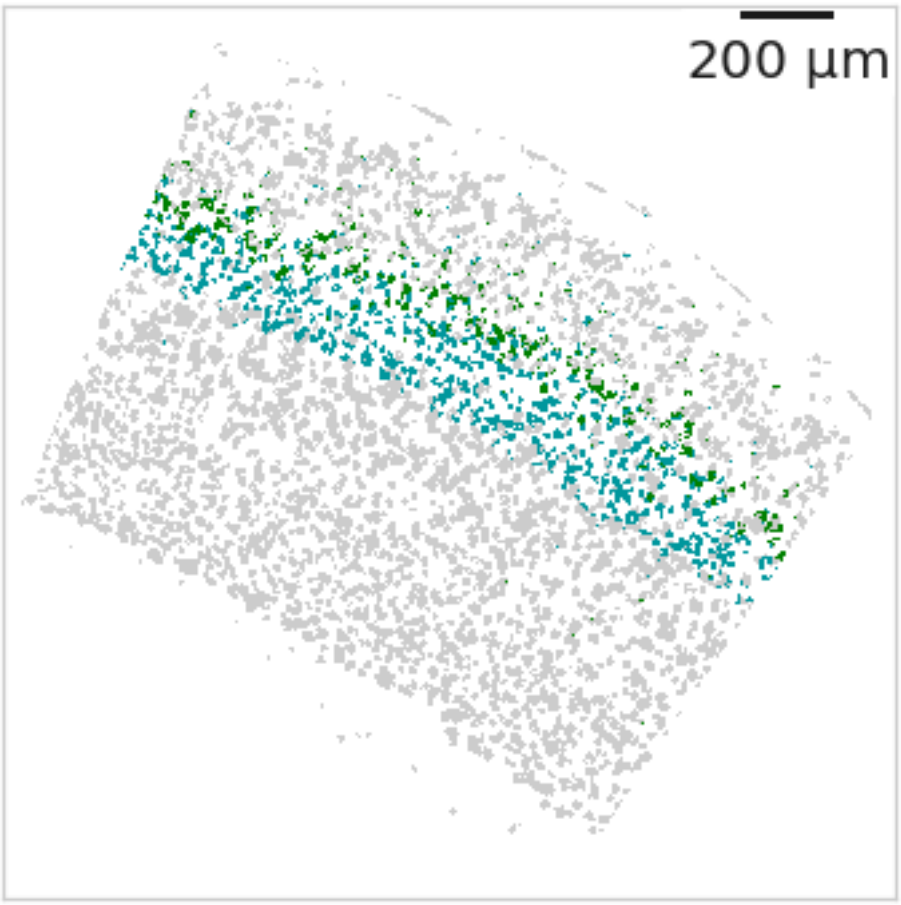
[44]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltypes_map(colors=denovo_celltype_colors, rotate=1, set_alpha=False)
scalebar = ScaleBar(1, 'um', pad=0.1, font_properties={"size": 20})
plt.gca().add_artist(scalebar)
plt.gca().get_xaxis().set_visible(False)
plt.gca().get_yaxis().set_visible(False)
plt.xlim([2050, 150])
plt.ylim([2050, 150])
plt.tight_layout()
```



[45]:
```
dc_cols = np.zeros_like(denovo_celltype_colors)
dc_cols[:] = "#cccccc"
dc_cols[10] = denovo_celltype_colors[10]
dc_cols[14] = denovo_celltype_colors[14]
```

Heterogeneity in L4

[46]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltypes_map(background='white', colors=dc_cols, rotate=1,␣
↪set_alpha=False)
scalebar = ScaleBar(1, 'um', pad=0.1, font_properties={"size": 20})
plt.gca().add_artist(scalebar)
plt.gca().get_xaxis().set_visible(False)
plt.gca().get_yaxis().set_visible(False)
plt.xlim([2050, 150])
plt.ylim([2050, 150])
plt.tight_layout()
```



Draw vector vs genes heatmap

[47]:
```
# Reorder clusters according to dendrogram order of Tasic et al. (2018)
import matplotlib
heatmap_clusters_dend_index = []
for cl in denovo_labels_final:
    if ' / ' in cl:
        cl = cl.split(' / ')[0].rstrip()
    if cl == 'VLMC':
        cl = 'VLMC Spp1 Hs3st6'
    elif cl == 'OPC':
        cl = 'OPC Pdgfra Grm5'
    elif cl == 'Oligo':
        cl = 'Oligo Serpinb1a'
    elif cl == 'L6b Crh':
        cl = 'L6b VISp Crh'
    if ' IT ' in cl:
        cl = cl.replace(' IT ', ' IT VISp ')
    elif ' NP ' in cl:
```

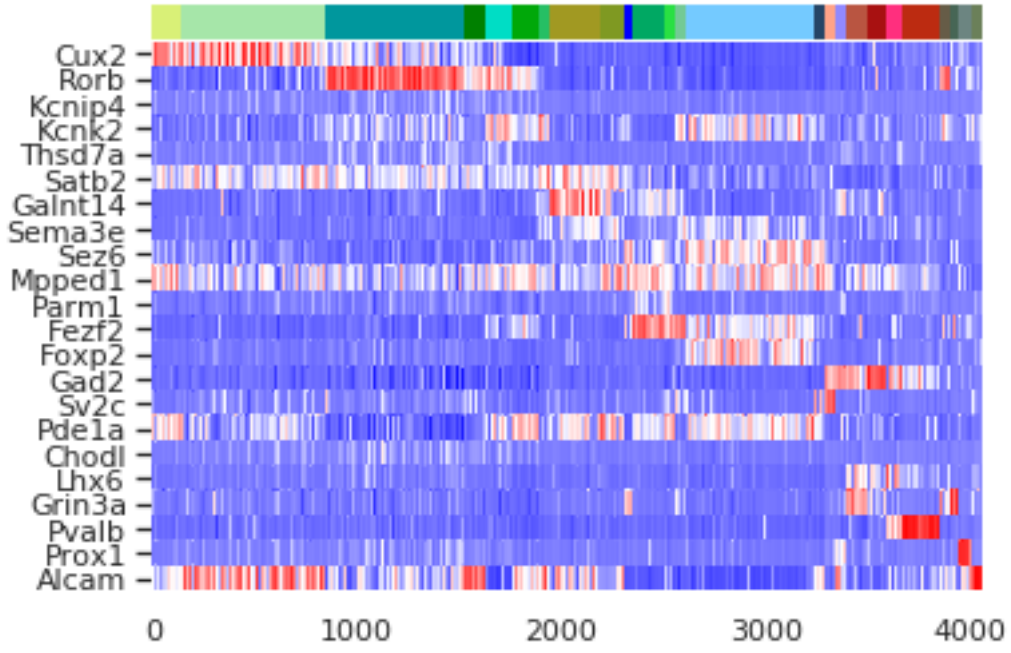
```
        cl = cl.replace(' NP ', ' NP VISp ')
    elif ' CT ' in cl:
        cl = cl.replace(' CT ', ' CT VISp ')
    elif ' PT ' in cl:
        cl = cl.replace(' PT ', ' PT VISp ')
    if cl == "L4 IT VISp Superficial":
        cl = 'L4 IT VISp Rspo1'
    heatmap_clusters_dend_index.append(dendrogram_order.index(cl))
heatmap_clusters_index = np.argsort(heatmap_clusters_dend_index)
heatmap_clusters_ordered = np.array(denovo_labels_final)[heatmap_clusters_index]
```

[48]:
```
# Shorten the cluster labels
denovo_labels_final_short = [
    'VLMC',
    'Vip',
    'L2/3 IT 1',
    'L2/3 IT 2',
    'Sst 3',
    'Lamp5',
    'Sst 2',
    'Pvalb 2',
    'Astro',
    'L6 IT 2',
    'L4 IT 2',
    'L5 IT 3',
    'OPC',
    'Oligo',
    'L4 IT 1',
    'L5 NP',
    'L5 IT 1',
    'Pvalb 1',
    'L6 CT',
    'L5 PT 1',
    'L5 IT 2',
    'L5 PT 2',
    'L5 PT 3',
    'L6 IT 1',
    'L6b',
    'Sst 1'
]
to_short = dict(zip(denovo_labels_final, denovo_labels_final_short))
heatmap_clusters_ordered_short = [to_short[cl] for cl in␣
 ↪heatmap_clusters_ordered]
```

[49]:
```
# Sort genes for heatmap
from matplotlib.colors import to_rgba, to_hex
```

```
heatmap_vectors = np.zeros([np.sum(ds.filtered_cluster_labels != -1), len(ds.
 ↪genes)], dtype=float)
col_colors = np.zeros(np.sum(ds.filtered_cluster_labels != -1), dtype='<U7')
acc_idx = 0
for cl_idx in heatmap_clusters_index:
    cl_vecs = ds.normalized_vectors[ds.filtered_cluster_labels == cl_idx]
    col = denovo_celltype_colors[cl_idx]
    heatmap_vectors[acc_idx:acc_idx+cl_vecs.shape[0], :] = cl_vecs
    col_colors[acc_idx:acc_idx+cl_vecs.shape[0]] = to_hex(col)
    acc_idx += cl_vecs.shape[0]

heatmap_genes_index = []
heatmap_genes_ordered = []
_, i = np.unique(col_colors, return_index=True)
uc = col_colors[sorted(i)]
mean_genes = np.zeros([len(uc), len(ds.genes)])
for i, c in enumerate(uc):
    mean_genes[i, :] = np.mean(heatmap_vectors[col_colors == c], axis=0)

max_exp_indices = np.argmax(mean_genes, axis=0)
for i in range(len(uc)):
    cl_gene_indices = np.where(max_exp_indices == i)[0]
    heatmap_genes_index += list(cl_gene_indices)
    heatmap_genes_ordered += list(np.array(ds.genes)[cl_gene_indices])
heatmap_vectors = heatmap_vectors[:, heatmap_genes_index]
```

[50]:
```
gene_exp_heatmap = heatmap_vectors.T
gene_exp_heatmap = preprocessing.scale(gene_exp_heatmap)
g = sns.clustermap(gene_exp_heatmap, figsize=[7, 5],␣
 ↪yticklabels=heatmap_genes_ordered,
                   cmap='bwr', row_cluster=False, col_cluster=False,
                   col_colors=col_colors, xticklabels = 1000)
g.cax.set_visible(False)
g.ax_heatmap.tick_params(labelright=False, labelleft=True, right=False,␣
 ↪left=True)
```

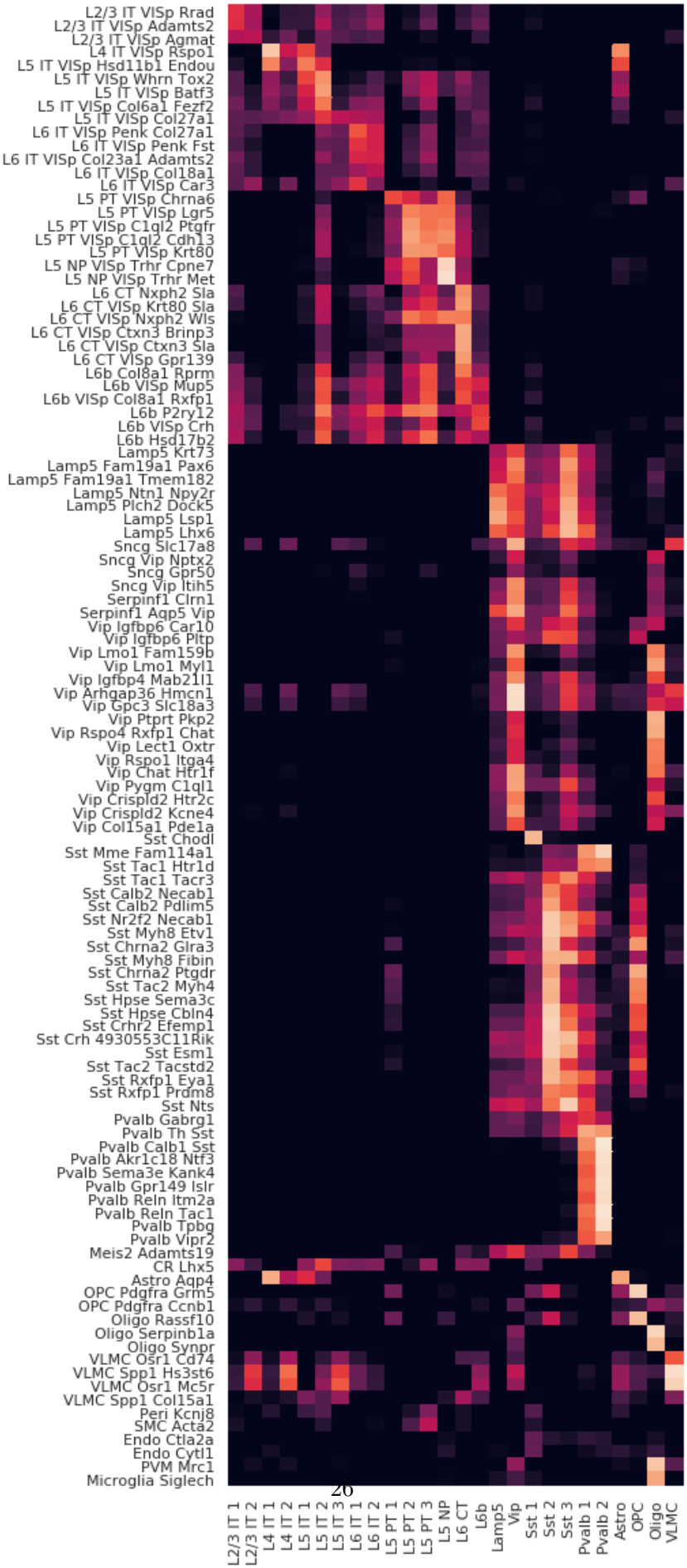


Draw correlation plot, SSAM vs scRNA-seq

[51]:
```
scrna_uniq_cluster_eng = [scrna_cl_metadata_dic[cl][0].strip() for cl in␣
↪scrna_uniq_clusters]
heatmap_scrna_clusters_index = [scrna_uniq_cluster_eng.index(cl) for cl in␣
↪dendrogram_order]
```

[52]:
```
ssam_scrna_pcorrs_final = np.zeros((len(scrna_centroids), len(ds.centroids)))
for i, scrna_centroid in enumerate(scrna_centroids):
    for j, centroid in enumerate(ds.centroids):
        ssam_scrna_pcorrs_final[i, j] = ssam.utils.corr(scrna_centroid,␣
↪centroid)

hm = ssam_scrna_pcorrs_final[:,␣
↪heatmap_clusters_index][heatmap_scrna_clusters_index, :]

plt.figure(figsize=[6, 20])
sns.heatmap(hm, vmin=0, vmax=1, yticklabels=dendrogram_order,␣
↪xticklabels=heatmap_clusters_ordered_short, cbar=False)
```

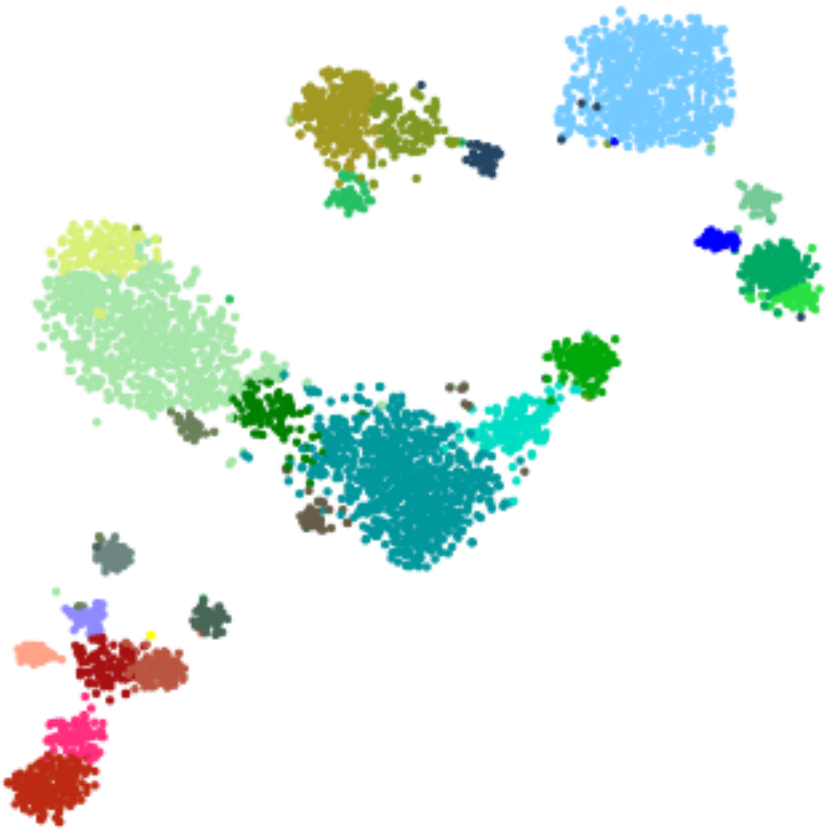
[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2aaf6f28be10>
```


L2/3 IT VISp Rrad
L2/3 IT VISp Adamts2
L2/3 IT VISp Agmat
L4 IT VISp Rspo1
L5 IT VISp Hsd11b1 Endou
L5 IT VISp Whrn Tox2
L5 IT VISp Batf3
L5 IT VISp Col6a1 Fezf2
L5 IT VISp Col27a1
L6 IT VISp Penk Col27a1
L6 IT VISp Penk Fst
L6 IT VISp Col23a1 Adamts2
L6 IT VISp Col18a1
L6 IT VISp Car3
L5 PT VISp Chrna6
L5 PT VISp Lgr5
L5 PT VISp C1ql2 Ptgfr
L5 PT VISp C1ql2 Cdh13
L5 PT VISp Krt80
L5 NP VISp Trhr Cpne7
L5 NP VISp Trhr Met
L6 CT Nxph2 Sla
L6 CT VISp Krt80 Sla
L6 CT VISp Nxph2 Wls
L6 CT VISp Ctxn3 Brinp3
L6 CT VISp Ctxn3 Sla
L6 CT VISp Gpr139
L6b Col8a1 Rprm
L6b VISp Mup5
L6b VISp Col8a1 Rxfp1
L6b P2ry12
L6b VISp Crh
L6b Hsd17b2
Lamp5 Krt73
Lamp5 Fam19a1 Pax6
Lamp5 Fam19a1 Tmem182
Lamp5 Ntn1 Npy2r
Lamp5 Plch2 Dock5
Lamp5 Lsp1
Lamp5 Lhx6
Sncg Slc17a8
Sncg Vip Nptx2
Sncg Gpr50
Sncg Vip Itih5
Serpinf1 Clrn1
Serpinf1 Aqp5 Vip
Vip Igfbp6 Car10
Vip Igfbp6 Pltp
Vip Lmo1 Fam159b
Vip Lmo1 Myl1
Vip Igfbp4 Mab21l1
Vip Arhgap36 Hmcn1
Vip Gpc3 Slc18a3
Vip Ptprt Pkp2
Vip Rspo4 Rxfp1 Chat
Vip Lect1 Oxtr
Vip Rspo1 Itga4
Vip Chat Htr1f
Vip Pygm C1ql1
Vip Crispld2 Htr2c
Vip Crispld2 Kcne4
Vip Col15a1 Pde1a
Sst Chodl
Sst Mme Fam114a1
Sst Tac1 Htr1d
Sst Tac1 Tacr3
Sst Calb2 Necab1
Sst Calb2 Pdlim5
Sst Nr2f2 Necab1
Sst Myh8 Etv1
Sst Chrna2 Glra3
Sst Myh8 Fibin
Sst Chrna2 Ptgdr
Sst Tac2 Myh4
Sst Hpse Sema3c
Sst Hpse Cbln4
Sst Crhr2 Efemp1
Sst Crh 4930553C11Rik
Sst Esm1
Sst Tac2 Tacstd2
Sst Rxfp1 Eya1
Sst Rxfp1 Prdm8
Sst Nts
Pvalb Gabrg1
Pvalb Th Sst
Pvalb Calb1 Sst
Pvalb Akr1c18 Ntf3
Pvalb Sema3e Kank4
Pvalb Gpr149 Islr
Pvalb Reln Itm2a
Pvalb Reln Tac1
Pvalb Tpbg
Pvalb Vipr2
Meis2 Adamts19
CR Lhx5
Astro Aqp4
OPC Pdgfra Grm5
OPC Pdgfra Ccnb1
Oligo Rassf10
Oligo Serpinb1a
Oligo Synpr
VLMC Osr1 Cd74
VLMC Spp1 Hs3st6
VLMC Osr1 Mc5r
VLMC Spp1 Col15a1
Peri Kcnj8
SMC Acta2
Endo Ctla2a
Endo Cytl1
PVM Mrc1
Microglia Siglech
L2/3 IT 1
L2/3 IT 2
L4 IT 1
L4 IT 2
L5 IT 1
L5 IT 2
L5 IT 3
L6 IT 1
L6 IT 2
L5 PT 1
L5 PT 2
L5 PT 3
L5 NP
L6 CT
L6b
Lamp5
Vip
Sst 1
Sst 2
Sst 3
Pvalb 1
Pvalb 2
Astro
OPC
Oligo
VLMC

Draw tSNE

[53]:
```
load_tsne("final_excluded_2d")
```

[54]:
```
plt.figure(figsize=[5, 5])
ds.plot_tsne(pca_dims=22, metric="correlation", s=5, run_tsne=False,␣
↪colors=denovo_celltype_colors)
plt.gca().get_xaxis().set_visible(False)
plt.gca().get_yaxis().set_visible(False)
plt.gca().axis('off')
plt.tight_layout()
```



Plot diagnostic plot with the final result

[55]:
```
for idx in range(len(ds.centroids)):
    plt.figure(figsize=[50, 15])
    ds.plot_diagnostic_plot(idx,
```

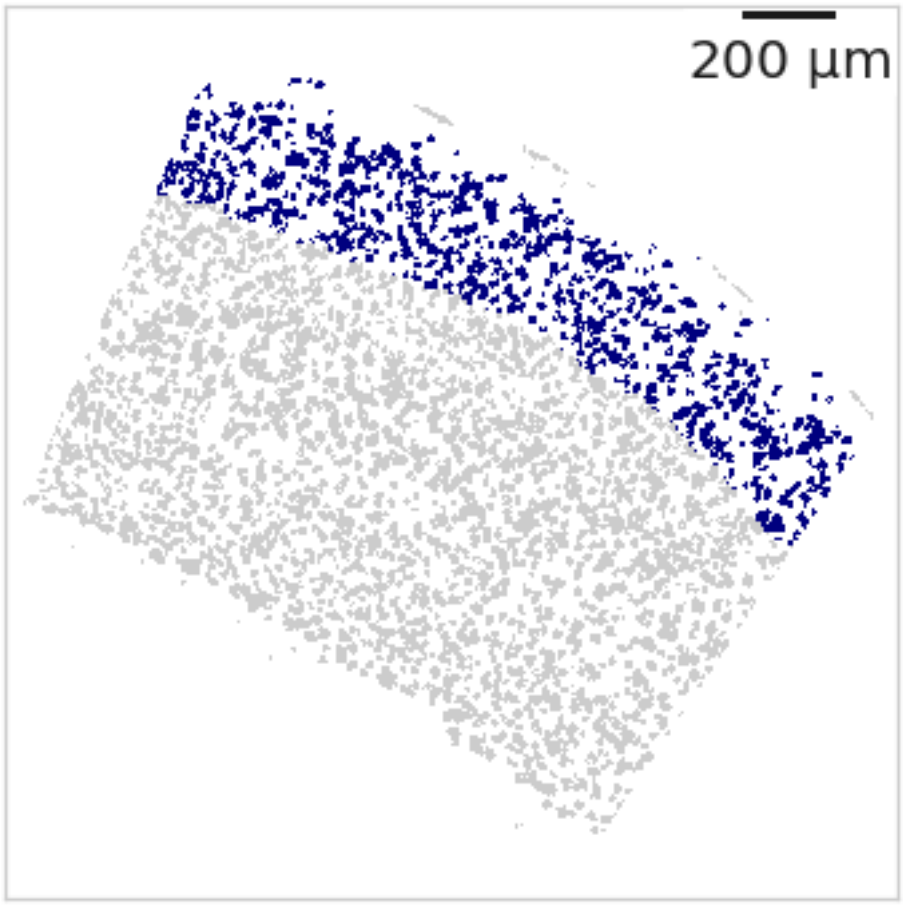
```
                              cluster_name=denovo_labels_final_short[idx],
                              cluster_color=denovo_celltype_colors[idx])
    plt.tight_layout()
    plt.savefig('diagplots_multiplexed_smFISH/
↪diagplot_centroid_%d_after_merging_removing.png'%idx)
    plt.close()
```

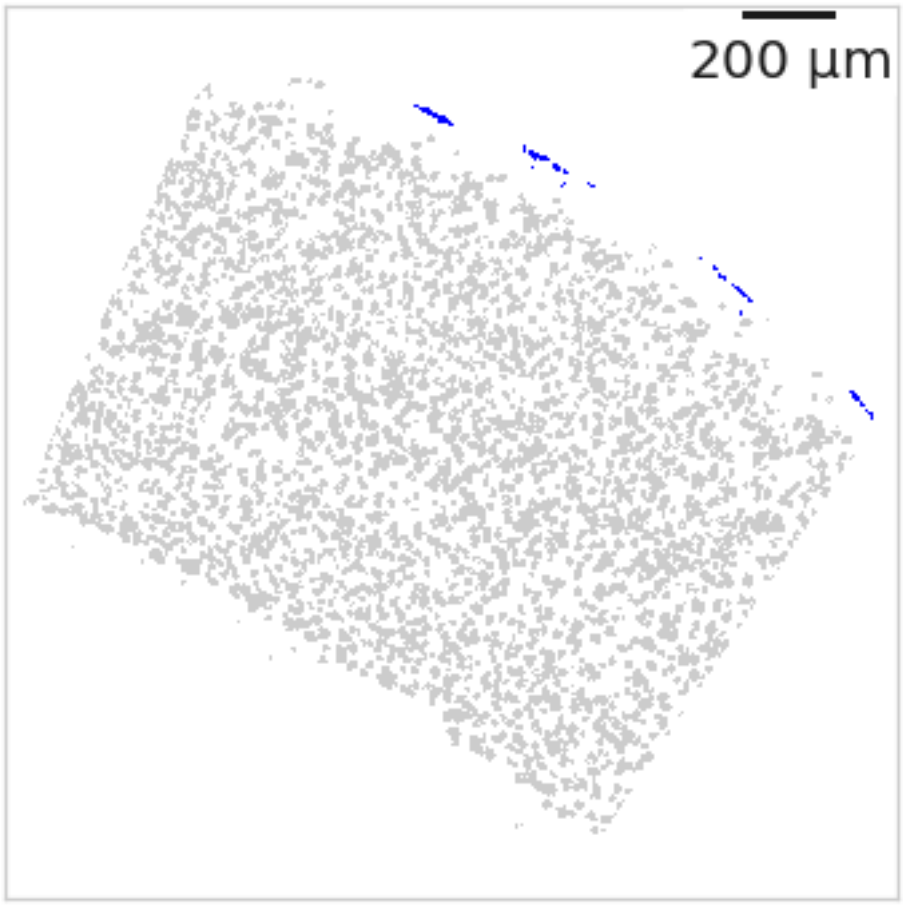
Infer domains in tissue

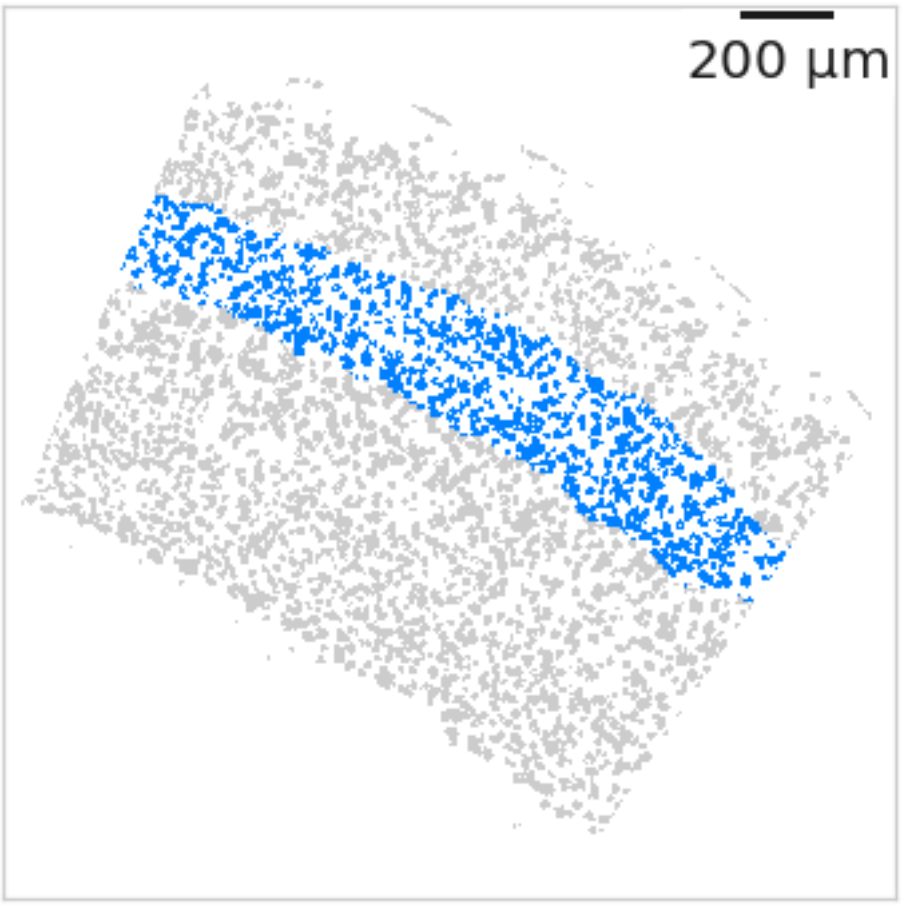
[56]:
```
# Sweep circular window
analysis.bin_celltypemaps(step=10, radius=100)
```

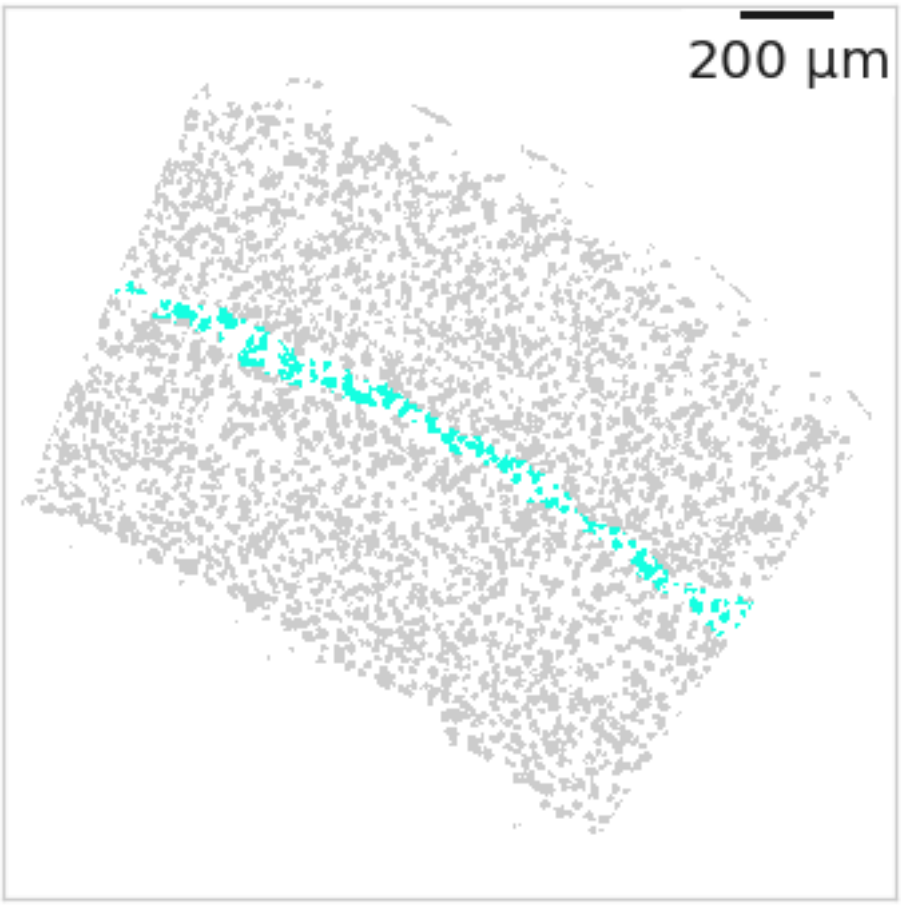
[57]:
```
analysis.find_domains(n_clusters=20, merge_remote=True, merge_thres=0.7,␣
↪norm_thres=1500)
```

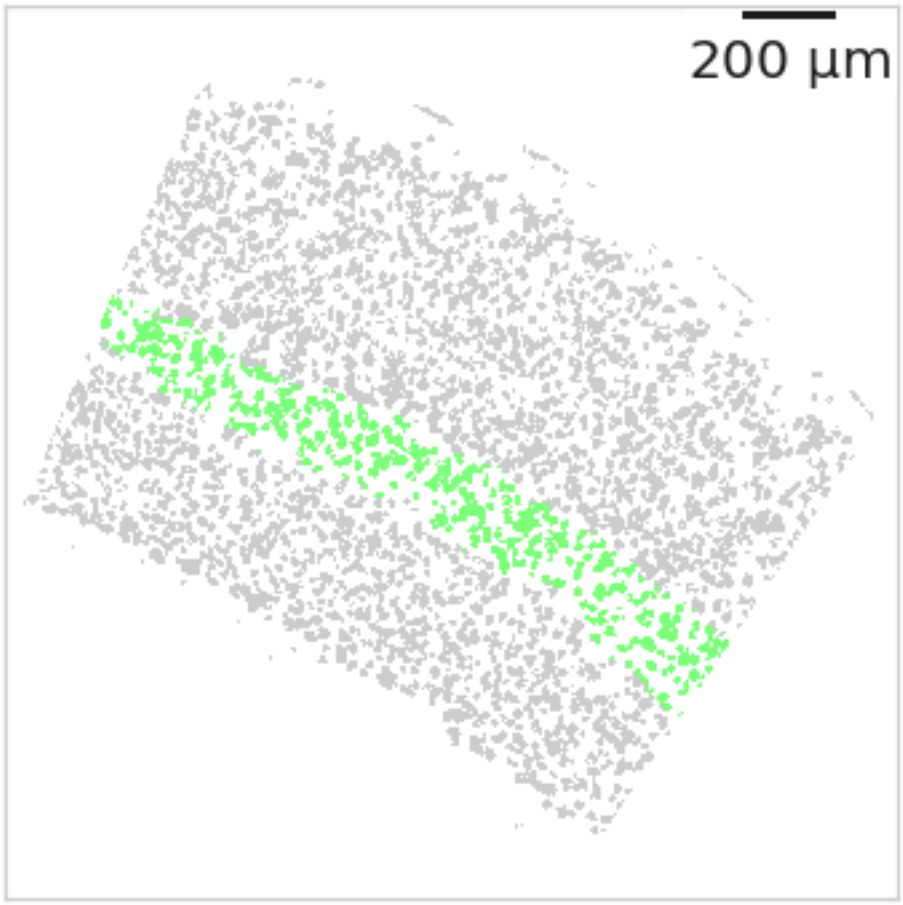
[58]:
```
# Check found domains
from matplotlib.colors import ListedColormap
cmap_jet = plt.get_cmap('jet')
num_domains = np.max(ds.inferred_domains_cells) + 1
for domain_idx in range(num_domains):
    plt.figure(figsize=[5, 5])
    cmap = ListedColormap([cmap_jet(lbl_idx / num_domains) if domain_idx ==␣
↪lbl_idx else "#cccccc" for lbl_idx in range(num_domains)])
    ds.plot_domains(rotate=1, cmap=cmap)
    scalebar = ScaleBar(1, 'um', pad=0.1, font_properties={"size": 20})
    plt.gca().add_artist(scalebar)
    plt.gca().get_xaxis().set_visible(False)
    plt.gca().get_yaxis().set_visible(False)
    plt.xlim([2050, 150])
    plt.ylim([2050, 150])
    plt.tight_layout()
```

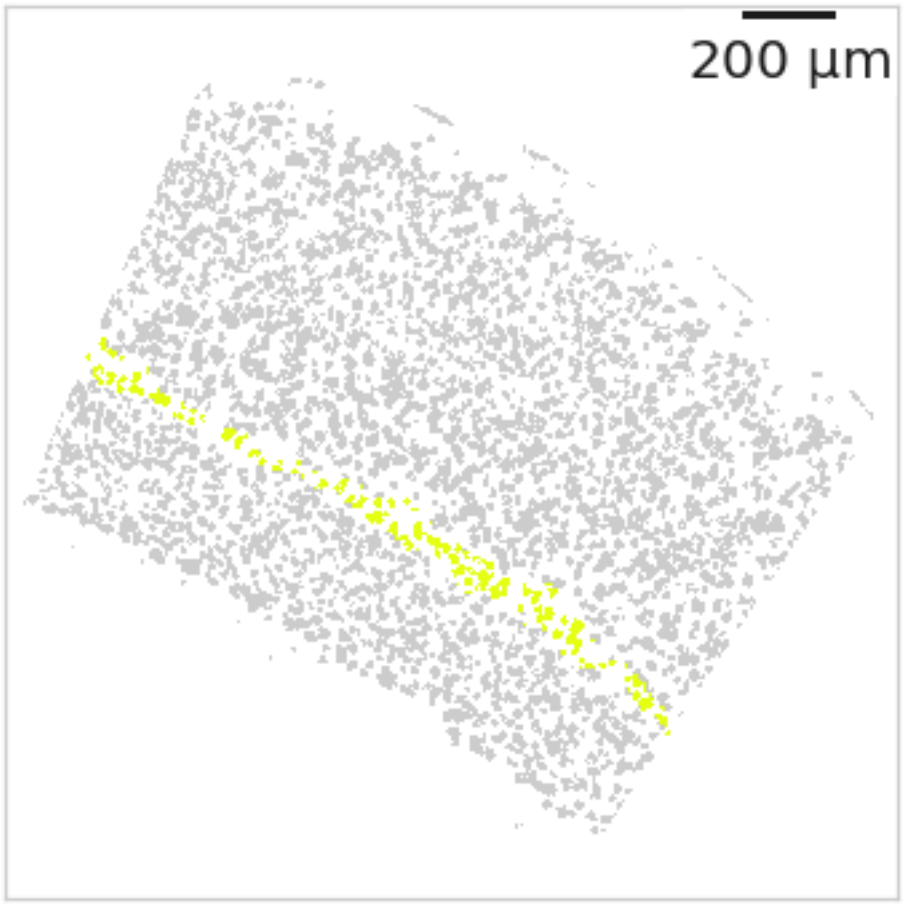

200 µm

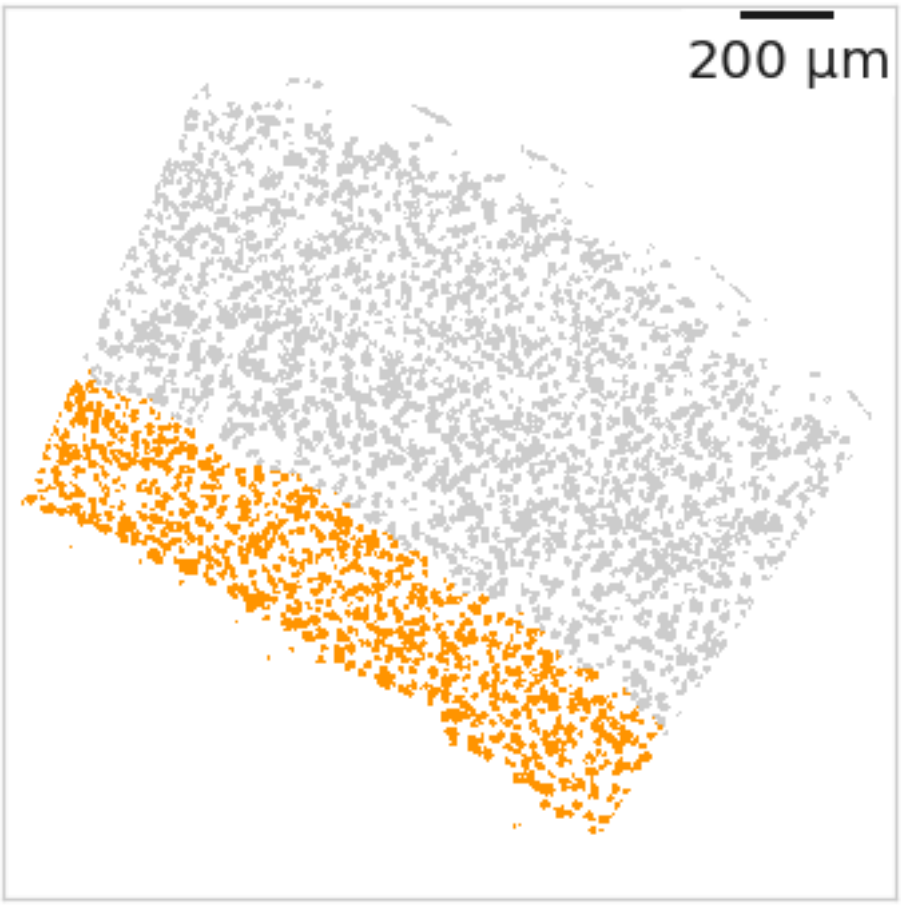

200 µm

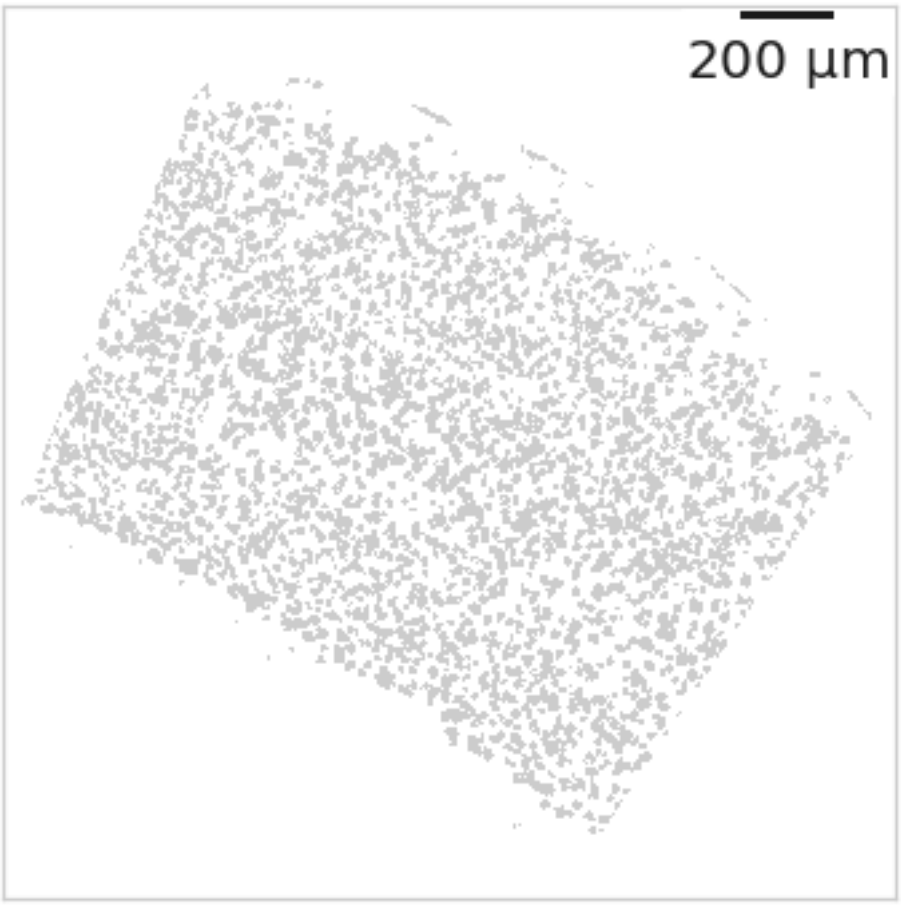

200 µm


200 µm


200 µm


200 µm


200 µm



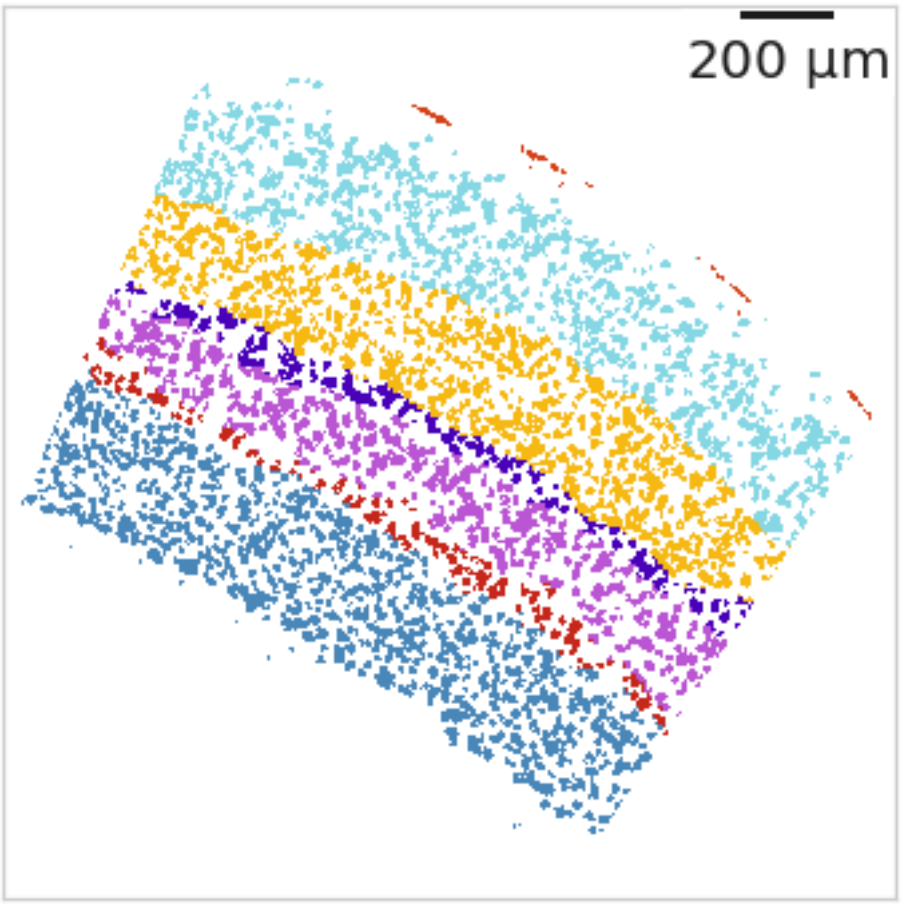
[59]:
```
#excluded_domain_indices = []
#merged_domain_indices = [[6, 7], ]
excluded_domain_indices = [7]
merged_domain_indices = []
```

[60]:
```
analysis.exclude_and_merge_domains(excluded_domain_indices,␣
↪merged_domain_indices)
```

[61]:
```
# Define domain colors
domain_colors = {
    'Pia': '#D44218',
    'Layer 1/2/3': '#85D7E4',
    'Layer 4': '#F6B813',
    'Layer 4/5': '#4900B9',
    'Layer 5a': '#BA55D3',
    'Layer 5b': '#C6271B',
    'Layer 6': '#4987B9',
}
```

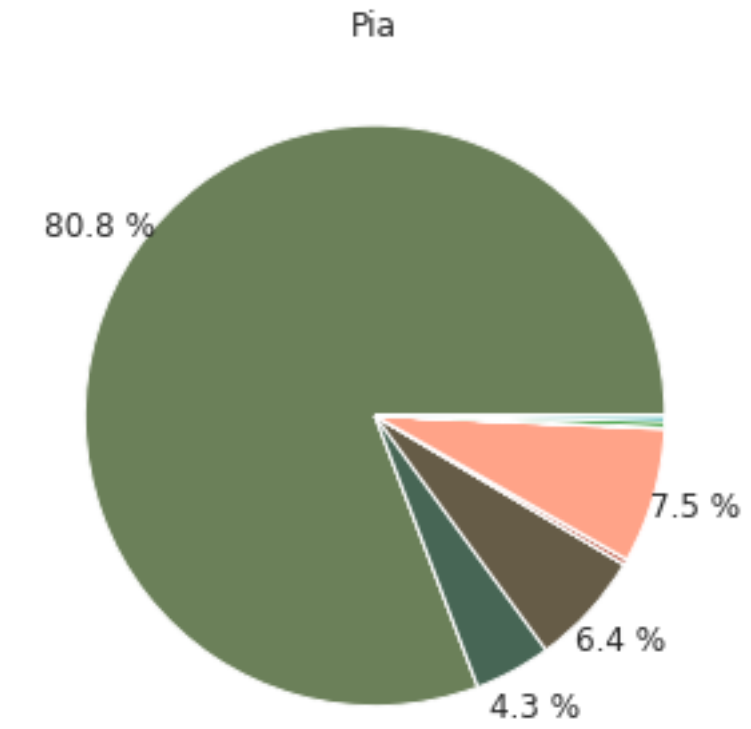
[62]:
```
# Define domain labels
domain_labels = [
    'Layer 1/2/3',
    'Pia',
    'Layer 4',
    'Layer 4/5',
    'Layer 5a',
    'Layer 5b',
    'Layer 6',
]
```

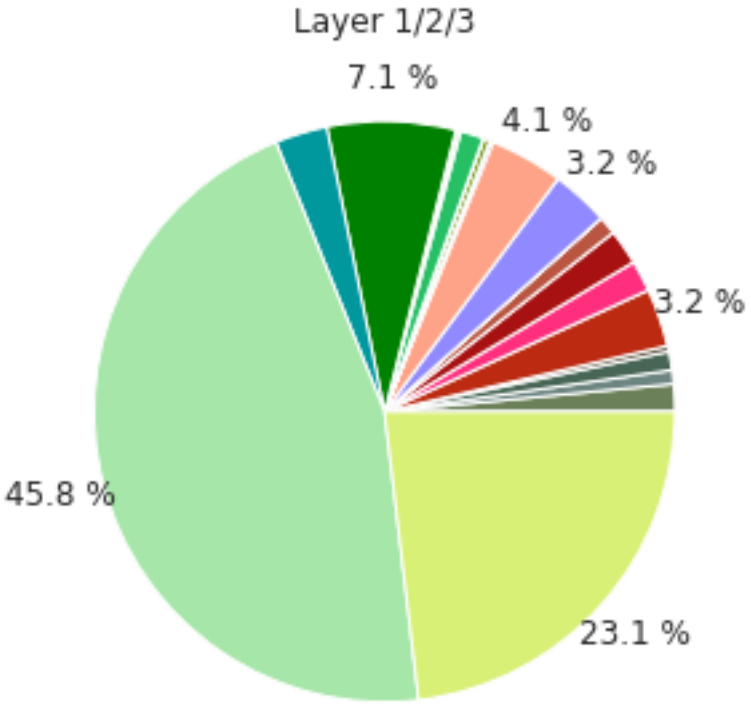
[63]:
```
from matplotlib.colors import ListedColormap
plt.figure(figsize=[5, 5])
cmap = ListedColormap([domain_colors[lbl] for lbl in domain_labels])
ds.plot_domains(rotate=1, cmap=cmap)
scalebar = ScaleBar(1, 'um', pad=0.1, font_properties={"size": 20})
plt.gca().add_artist(scalebar)
plt.gca().get_xaxis().set_visible(False)
plt.gca().get_yaxis().set_visible(False)
plt.xlim([2050, 150])
plt.ylim([2050, 150])
plt.tight_layout()
```

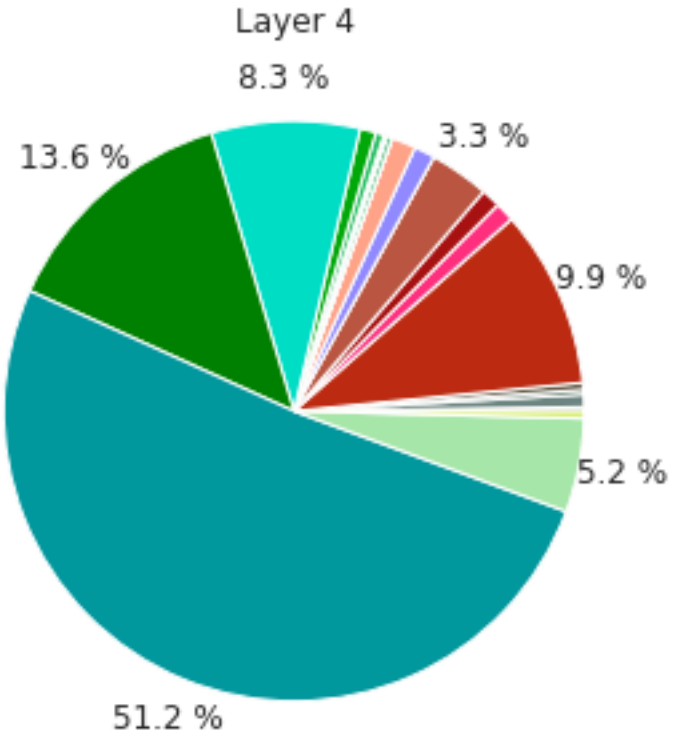


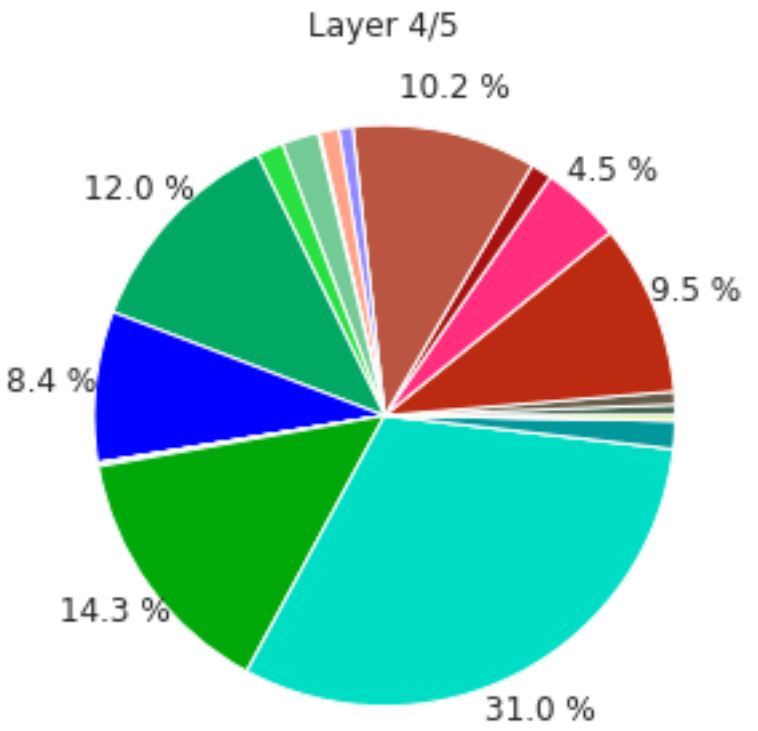
[64]:
```
analysis.calc_cell_type_compositions()
```

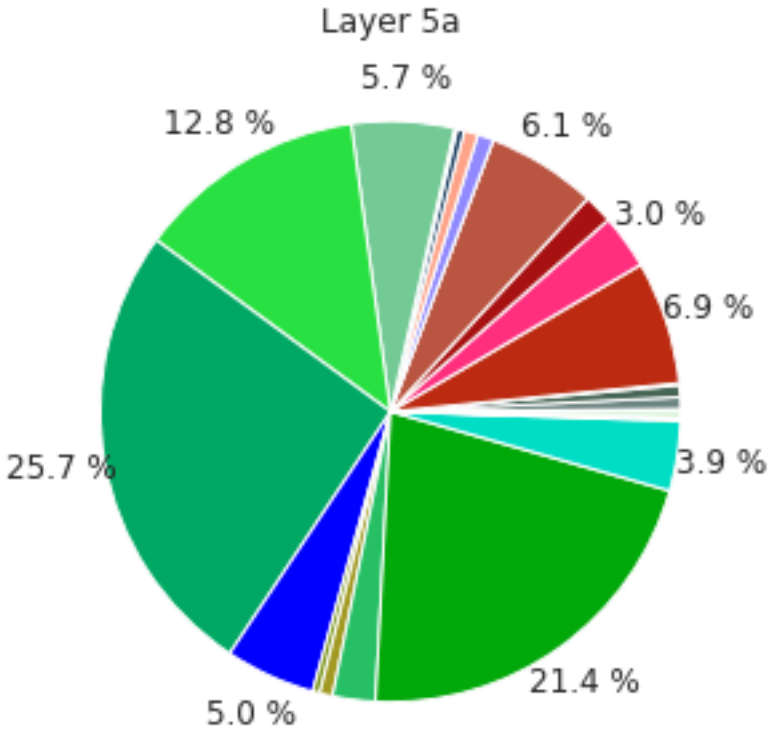
[65]:
```
for domain_idx in [1, 0, 2, 3, 4, 5, 6]:
    plt.figure(figsize=[5, 5])
    ds.plot_celltype_composition(domain_idx,
                                 cell_type_colors=denovo_celltype_colors,
                                 cell_type_orders=heatmap_clusters_index[::-1],
                                 label_cutoff=0.03)
    plt.title(domain_labels[domain_idx])
```

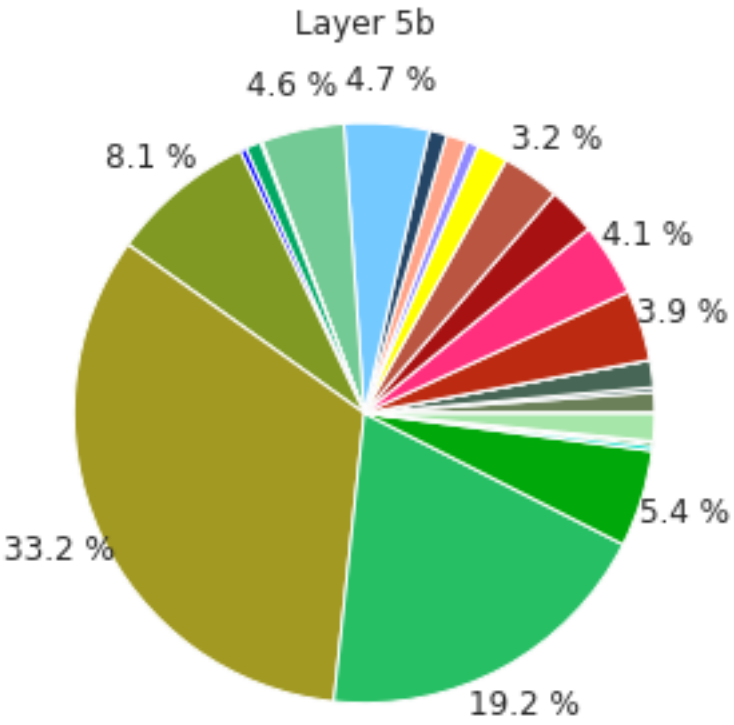

Pia
80.8 %
7.5 %
6.4 %
4.3 %

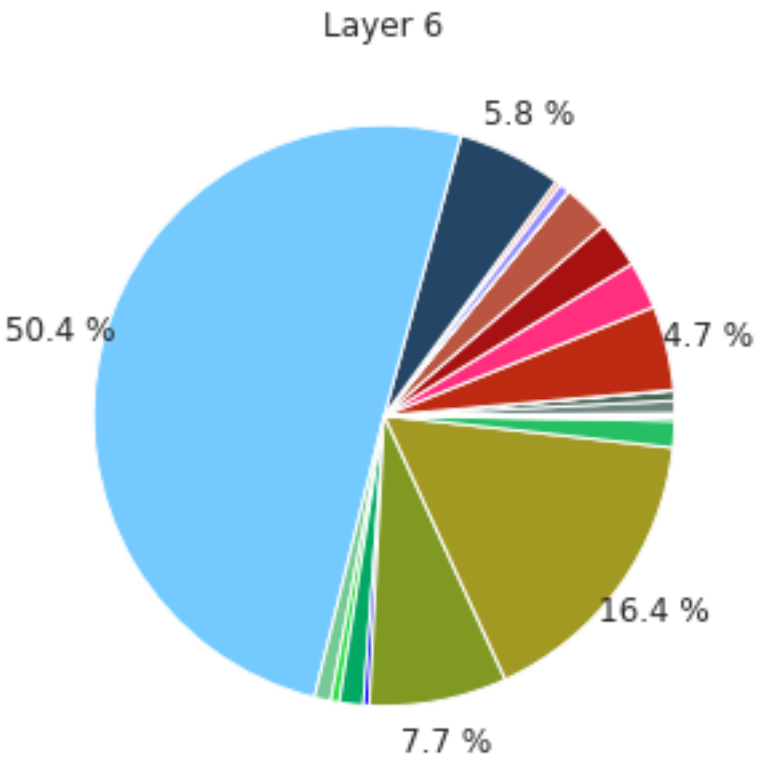

Layer 1/2/3
7.1 %
4.1 %
3.2 %
3.2 %
45.8 %
23.1 %


Layer 4
8.3 %
3.3 %
13.6 %
9.9 %
5.2 %
51.2 %


Layer 4/5
10.2 %
4.5 %
12.0 %
9.5 %
8.4 %
14.3 %
31.0 %

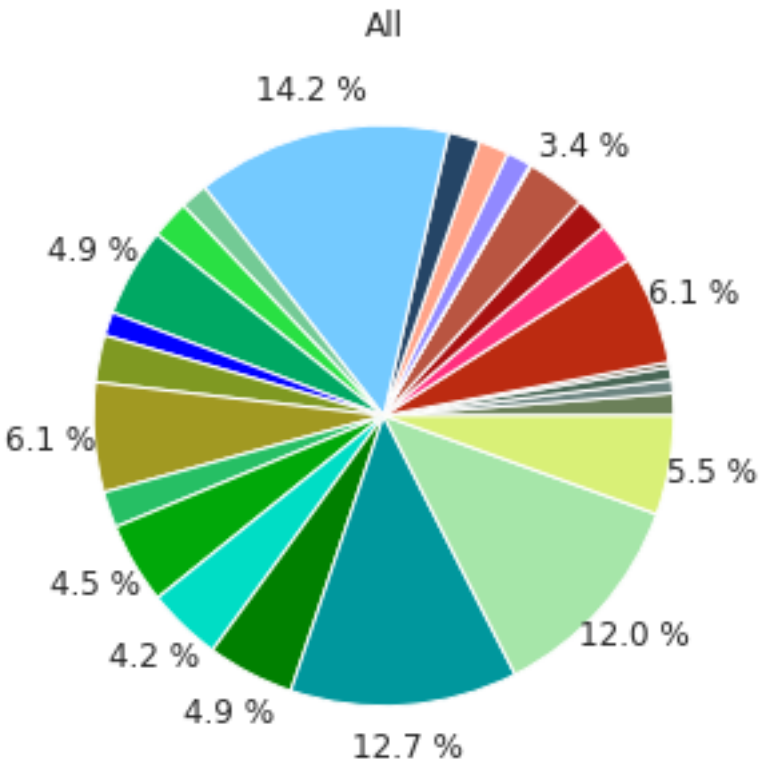

Layer 5a
5.7 %
12.8 %
6.1 %
3.0 %
6.9 %
25.7 %
3.9 %
21.4 %
5.0 %


Layer 5b
4.6 %
4.7 %
3.2 %
8.1 %
4.1 %
3.9 %
5.4 %
33.2 %
19.2 %



[66]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltype_composition(domain_index=7,
                             cell_type_colors=denovo_celltype_colors,
                             cell_type_orders=heatmap_clusters_index[::-1],
                             label_cutoff=0.03)
plt.title('All')
```

[66]:
```
Text(0.5, 1.0, 'All')
```


All
14.2 %
3.4 %
4.9 %
6.1 %
6.1 %
5.5 %
4.5 %
12.0 %
4.2 %
4.9 %
12.7 %