SSAM analysis of mouse POA, imaged by MERFISH

- Author: Jeongbin Park
- Date: 2019-10-13

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]:
```
## Parameters for post-filtering of cell type maps
filter_method = "local"
filter_params = {
    "block_size": 151,
    "method": "mean",
    "mode": "constant",
    "offset": 0.1
}
```

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

[4]:
```
from matplotlib.colors import to_rgba, to_hex

# Helper function to create shades of given color
def generate_shades(col, maxlen, percentage):
    def mix_color(c1, c2, percentage):
        w = percentage / 100.0
        return c1 * (1 - w) + c2 * w

    isstr = False
    if type(col) is str:
        col = to_rgba(col)
        isstr = True

    col = np.array(col)
    white = np.array([1, 1, 1, 1])
    black = np.array([0, 0, 0, 1])

    len_darks = int(np.floor(maxlen / 2))
    len_brights = len_darks + (0 if maxlen % 2 == 0 else 1)
```

```
    shades = []
    for i in range(len_brights):
        if percentage * i > 100.0:
            break
        shades.append(mix_color(col, white, percentage * i))
    shades = shades[::-1]
    for i in range(1, len_darks + 1):
        if percentage * i > 100.0:
            break
        shades.append(mix_color(col, black, percentage * i))

    shades = tuple(shades)
    if isstr:
        shades = [to_hex(e) for e in shades]

    return shades
```

[5]:
```
# Helper function to shorten cluster labels
def shorten_labels(lbls):
    l = []
    for lbl in lbls:
        l.append(lbl.
                 replace("Inhibitory", "Inhib.").
                 replace("Excitatory", "Excit.")
                )
    return l
```

Load data

Load mRNA spot locations

[6]:
```
import numpy as np
from collections import defaultdict

pos_dic = defaultdict(lambda: [])

with open("zenodo/MERFISH/raw_data/merfish_barcodes_example.csv") as f:
    f.readline()
    for line in f:
        e = line.rstrip().split(',')
        g, x, y, z = e[0], e[6], e[7], e[8]
        if g in ['Blank-1', 'Blank-2', 'Blank-3', 'Blank-4', 'Blank-5']:
            continue
        x, y, z = float(x), float(y), float(z)
        pos_dic[g].append([x, y, z])
```

```
all_genes = list(pos_dic.keys())
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

[7]:
```
import ssam
```

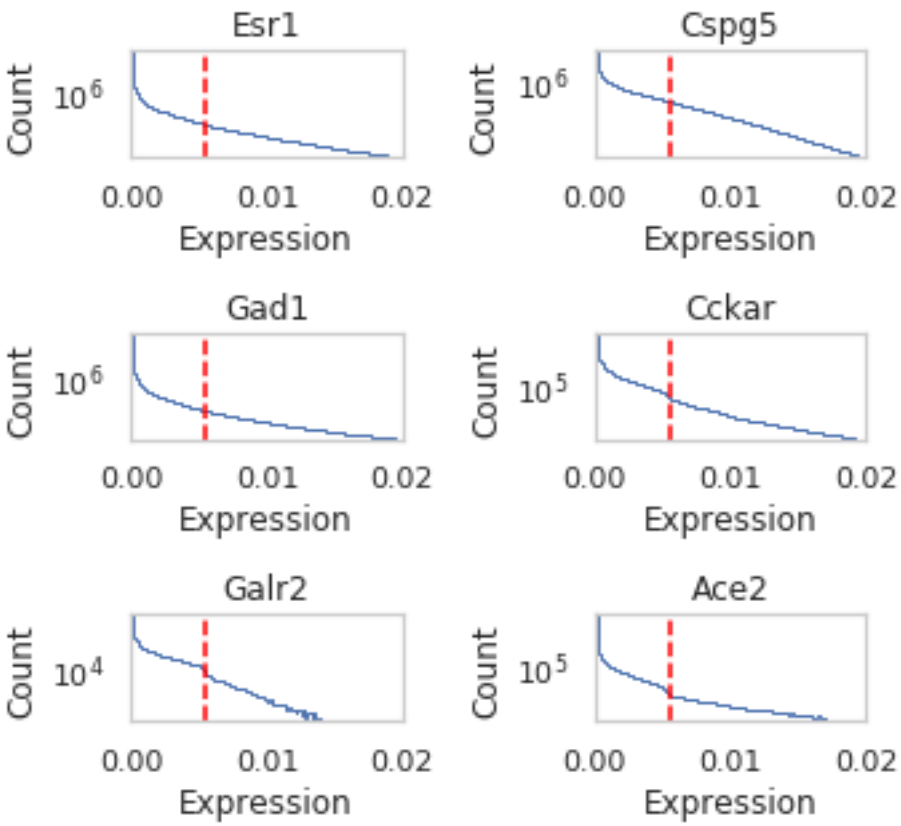
[8]:
```
width = 1790
height = 1790
depth = 9
```

[9]:
```
all_genes = list(pos_dic.keys())
ds = ssam.SSAMDataset(all_genes, [pos_dic[gene] for gene in all_genes], width,␣
 ↪height, depth)
analysis = ssam.SSAMAnalysis(ds, ncores=10, save_dir="zenodo/MERFISH/kde",␣
 ↪verbose=True)
```

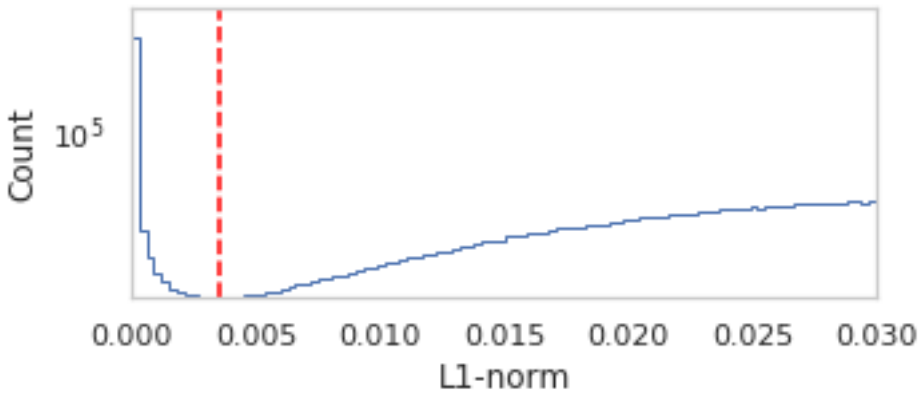
[10]:
```
analysis.run_kde(bandwidth=2.5, use_mmap=False)
```

Select local maxima of gene expression in the vector field

[11]:
```
exp_thres = 0.0055
viewport = 0.02
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
```



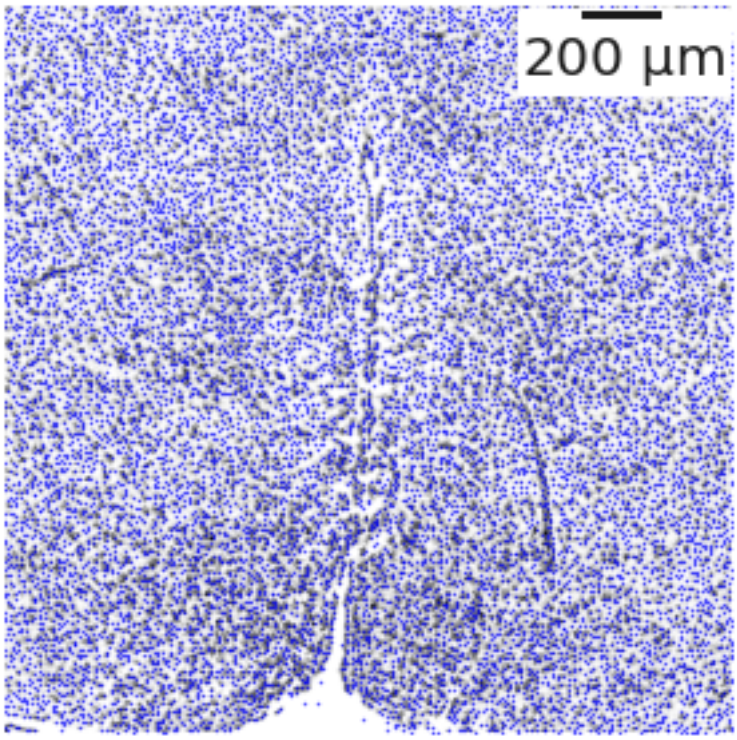
[12]:
```
gidx = 0
norm_thres = 0.0035
plt.figure(figsize=[5, 2])
n, _, _ = plt.hist(ds.vf_norm[np.logical_and(ds.vf_norm > 0, ds.vf_norm < 0.
↪03)], bins=100, log=True, histtype='step')
ax = plt.gca()
ax.axvline(norm_thres, c='red', ls='--')
ax.set_xlabel("L1-norm")
ax.set_ylabel("Count")

plt.xlim([0, 0.03])
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)
```

```
Found 13666 local max vectors.
```

[14]:
```
plt.figure(figsize=[5, 5])
ds.plot_l1norm(cmap="Greys", rotate=3)
ds.plot_localmax(c="blue", rotate=3, s=0.1)
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.gca().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 MERFISH segmentation-based centroids and scRNA-seq data (Moffit *et al.*)

1) SSAM guided by segmentation-based cluster centroids

Load MERFISH segementation-based expression data

[16]:
```
# Load slice at -0.24 (the most posterior slice)
selected_z = "-0.24"
cid_dic = {}
merfish_gene_expressions = defaultdict(lambda: [])
with open("zenodo/MERFISH/raw_data/
 ↪Moffitt_and_Bambah-Mukku_et_al_merfish_all_cells.csv") as f:
    line = f.readline()
    entries = line.strip().split(',')
    merfish_genes = entries[9:]
    for line in f:
```

```
        entries = line.strip().split(',')
        cid, aid, z, cell_type, neuron_cell_type = entries[0], entries[1],␣
↪entries[4], entries[7], entries[8]
        if aid != '1':
            continue
        elif cell_type == "Ambiguous":
            continue
        if z == selected_z:
            cid_dic[cid] = cell_type
            if cell_type == "Inhibitory" or cell_type == "Excitatory":
                cell_type = neuron_cell_type
            merfish_gene_expressions[cell_type].append([float(e) for e in␣
↪entries[9:]])
```

[17]:
```
merfish_gene_indices = [merfish_genes.index(gene) for gene in ds.genes]
merfish_genes = np.array(merfish_genes)[merfish_gene_indices]
for cl in merfish_gene_expressions:
    merfish_gene_expressions[cl] = np.array(merfish_gene_expressions[cl])[:,␣
↪merfish_gene_indices]
```

[18]:
```
merfish_uniq_labels = np.array(sorted(merfish_gene_expressions.keys()))
merfish_centroids = np.zeros([len(merfish_uniq_labels), len(ds.genes)])

for cl_idx, cl in enumerate(merfish_uniq_labels):
    merfish_centroids[cl_idx, :] = np.mean(merfish_gene_expressions[cl], axis=0)
```

Map it to vector field

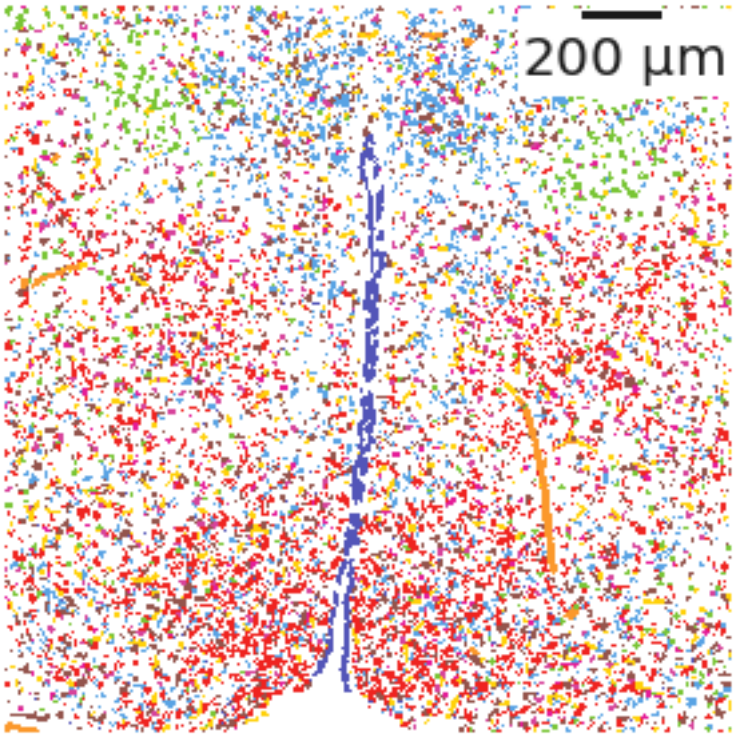
[19]:
```
analysis.map_celltypes(merfish_centroids)
analysis.filter_celltypemaps(min_norm=filter_method,␣
↪filter_params=filter_params, min_r=0.5)
```

[20]:
```
# The colors below is extracted from Figure 3 of Moffit et al. (2018)
cell_class_colors = {
    'Inhibitory': '#ef2420',
    'Excitatory': '#59a3e4',
    'Mature OD': '#79c535',
    'Immature OD': '#d7439b',
    'Astrocyte': '#94544c',
    'Microglia': '#e51c98',
    'Ependymal': '#5355b8',
    'Endothelial': '#ffd109',
    'Mural': '#fb982b'
}
```

[21]:
```
# Define cluster colors based on the cell-class colors
merfish_colors_dic = {}
```

```
for cl in merfish_uniq_labels:
    if cl[:2] == "E-":
        merfish_colors_dic[cl] = cell_class_colors['Excitatory']
    elif cl[:2] == "I-" or cl[:2] == "H-":
        merfish_colors_dic[cl] = cell_class_colors['Inhibitory']
    elif 'Mature' in cl:
        merfish_colors_dic[cl] = cell_class_colors['Mature OD']
    elif 'Immature' in cl:
        merfish_colors_dic[cl] = cell_class_colors['Immature OD']
    elif 'Endothelial' in cl:
        merfish_colors_dic[cl] = cell_class_colors['Endothelial']
    elif cl == 'Pericytes':
        merfish_colors_dic[cl] = cell_class_colors['Mural']
    else:
        merfish_colors_dic[cl] = cell_class_colors[cl]
merfish_colors = [merfish_colors_dic[cl] for cl in merfish_uniq_labels]
```

[22]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltypes_map(rotate=3, background='white', colors=merfish_colors,␣
 ↪set_alpha=False)
from matplotlib_scalebar.scalebar import ScaleBar
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.gca().axis('off')
plt.show()
```



2) SSAM guided by scRNA-seq cluster centroids

Load scRNA-seq data

[23]:
```
import gzip
scrna_genes = []
with gzip.open("zenodo/MERFISH/raw_data/GSE113576_genes.tsv.gz", "rt") as f:
    for line in f:
        ensid, gene = line.rstrip().split('\t')
        # Correct gene name
        if gene == '4732456N10Rik':
            gene = "Krt90"
        scrna_genes.append(gene)

scrna_barcodes = []
with gzip.open("zenodo/MERFISH/raw_data/GSE113576_barcodes.tsv.gz", "rt") as f:
    for line in f:
        scrna_barcodes.append(line.rstrip())
```

[24]:
```
from scipy.io import mmread
with gzip.open("zenodo/MERFISH/raw_data/GSE113576_matrix.mtx.gz") as f:
    scrna_expmat = mmread(f)
```

[25]:
```
# Normalize it with sctransform
norm_scrna_expmat = ssam.run_sctransform(scrna_expmat.T.todense(),␣
↪min_cells=1)[0]
```

[26]:
```
key_to_gene = {'%d'%i: g for i, g in enumerate(scrna_genes)}
filtered_scrna_genes = [key_to_gene[i] for i in norm_scrna_expmat.keys()]
gidx = [g in ds.genes for g in filtered_scrna_genes]
scrna_expmat_reduced = np.array(norm_scrna_expmat)[:, np.where(gidx)[0]]
scrna_genes_reduced = list(np.array(filtered_scrna_genes)[gidx])
sorted_gidx = [scrna_genes_reduced.index(g) for g in ds.genes]
scrna_expmat_final = scrna_expmat_reduced[:, sorted_gidx]
```

[27]:
```
scrna_cluster_dic = {}
scrna_cluster_dic2 = {}

# Downloaded from supp. material of Moffit et al, table S1
# This file contains cluster metadata
table_s1_df = pd.read_excel("zenodo/MERFISH/raw_data/aau5324_Moffitt_Table-S1.
↪xlsx", skiprows=[0,1], header=None)
for _, (bc, _, _, c1, c2, c3) in table_s1_df.iterrows():
    if c2 == '' and c3 == '':
        c2 = c1 # Ambiguous and Unstable
    elif c2 == '':
        c2 = c1 + ' (' + c3 + ')'

    scrna_cluster_dic[bc] = c1
    scrna_cluster_dic2[bc] = c2
```

[28]:
```
scrna_cluster_labels = [scrna_cluster_dic2[e] for e in scrna_barcodes]
scrna_uniq_labels = sorted(filter(lambda e: not e in ['Ambiguous', 'Unstable'],␣
↪set(scrna_cluster_labels)))
```

[29]:
```
scrna_centroids = []
for c in scrna_uniq_labels:
    scrna_centroids.append(np.mean(scrna_expmat_final[np.
↪array(scrna_cluster_labels) == c, :], axis=0))
```

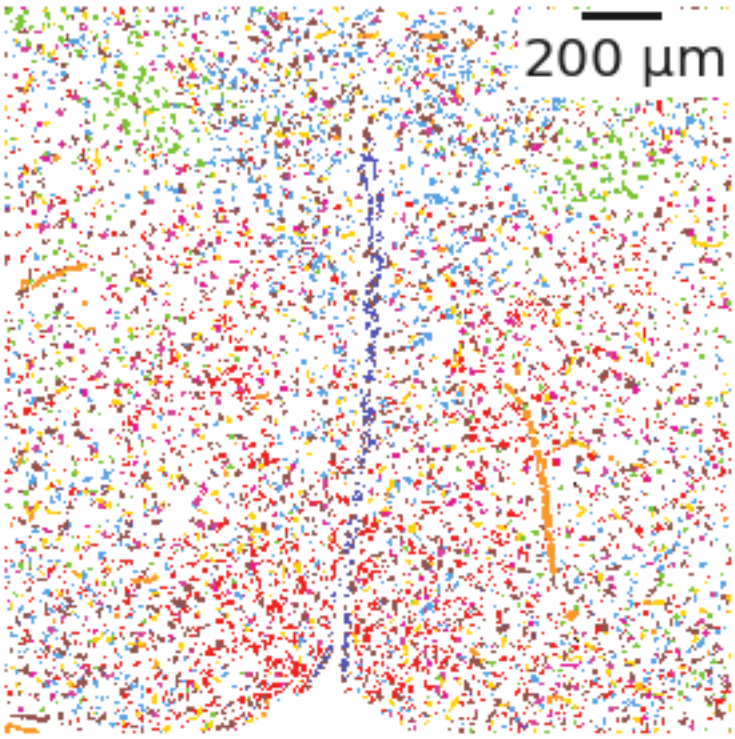
Map it to the vector field

[30]:
```
analysis.map_celltypes(scrna_centroids)
analysis.filter_celltypemaps(min_norm=filter_method,␣
↪filter_params=filter_params, min_r=0.5)
```

[31]:
```
# Define color of scRNAseq data
scrna_colors_dic = {}
for cl in scrna_uniq_labels:
    if cl == "Fibroblast 1":
```

```
        scrna_colors_dic[cl] = "black"
        continue
    elif cl == "Macrophage 1":
        scrna_colors_dic[cl] = "grey"
        continue
    r_cl = cl.replace("oligodendrocyte", 'OD').replace('Newly formed',␣
↪'Immature')
    for cate in cell_class_colors:
        if cate in r_cl:
            scrna_colors_dic[cl] = cell_class_colors[cate]
            break
scrna_colors = [scrna_colors_dic[cl] for cl in scrna_uniq_labels]
```

[32]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltypes_map(rotate=3, background='white', colors=scrna_colors,␣
↪set_alpha=False)
from matplotlib_scalebar.scalebar import ScaleBar
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.gca().axis('off')
plt.show()
```



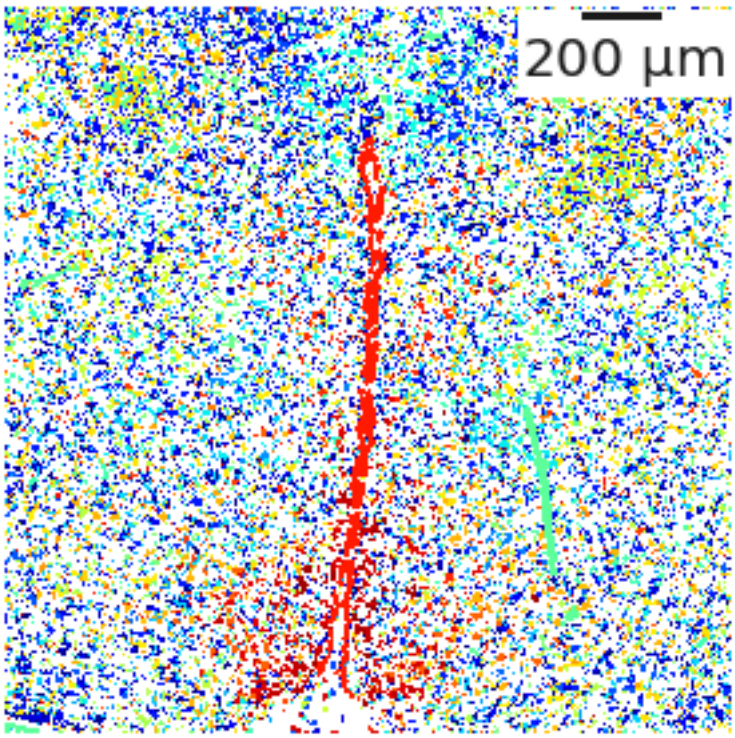
SSAM *de novo* mode

[33]:
```
analysis.cluster_vectors(pca_dims=60, resolution=0.15, max_correlation=1.0,␣
↪metric="correlation", centroid_correction_threshold=0.6)
```

```
Found 68 clusters
```

[34]:
```
filter_params['offset'] = 0
analysis.map_celltypes()
analysis.filter_celltypemaps(min_norm=filter_method,␣
↪filter_params=filter_params, min_r=0.6, min_blob_area=7, fill_blobs=True)
```

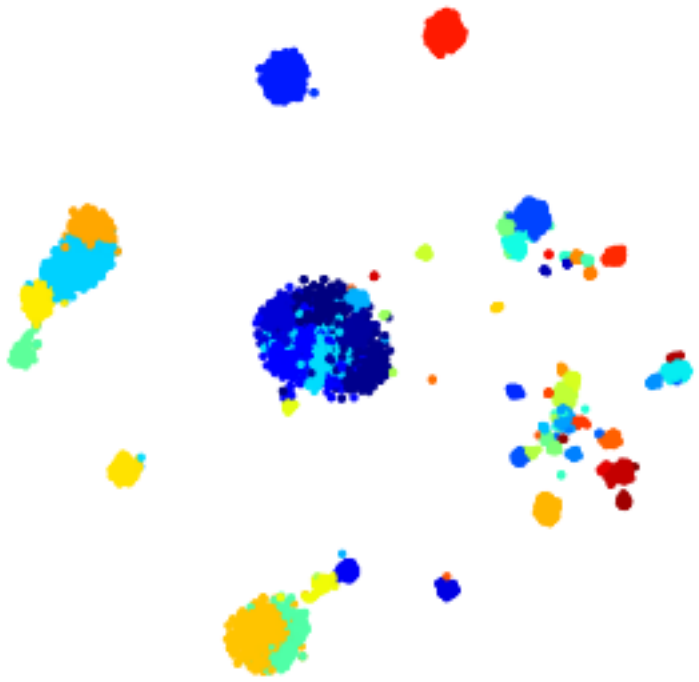
[35]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltypes_map(rotate=3, background='white', set_alpha=False)

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.gca().axis('off')
plt.show()
```



[36]:
```
load_tsne('excluded')
```

[37]:
```
import matplotlib.patheffects as PathEffects
plt.figure(figsize=[5, 5])
ds.plot_tsne(pca_dims=60, metric="correlation", s=5, run_tsne=False)
plt.axis('off')
```

[37]: (-79.68918890309955, 82.88242620778705, -75.24307382978199, 79.1792310563445)



[38]:
```
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=[
        ("segmentation", merfish_uniq_labels, merfish_centroids,␣
↪merfish_colors),
        ("scRNA-seq", scrna_uniq_labels, scrna_centroids, scrna_colors),
    ], correlation_methods=[
        ("r", pearsonr),
    ])
    plt.tight_layout()
```

```
    plt.savefig('diagplots_MERFISH/diagplot_centroid_%d.png'%idx)
    plt.close()
```

[39]:
```
denovo_labels = [
    "Astrocyte 2",
    "Astrocyte 2",
    "Astrocyte 1",
    "Excitatory Cbln2",
    "Excitatory Nos1",
    "Astrocyte 1",
    "Inhibitory Sytl4",
    "Mature OD 3 (Lpar1)",
    "Astrocyte 1",
    "N/A", # Too few vectors

    "Immature OD",
    "Inhibitory Syt2,Ramp3",
    "N/A", # Too few vectors
    "Excitatory Necab1",
    "Inhibitory Amigo2,Sema3c",
    "Inhibitory Dgkk",
    "N/A",
    "Inhibitory Esr1",
    "Inhibitory Arhgap36,Gda",
    "Inhibitory Gad1",

    "Astrocyte 2",
    "Inhibitory Gda",
    "Endothelial 2",
    "Astrocyte 1",
    "Inhibitory Arhgap36",
    "Excitatory Syt4",
    "Inhibitory Synpr",
    "Excitatory Fezf1",
    "Inhibitory Tacr1",
    "Excitatory Cbln1,Cbln2 (e19)",

    "Mature OD 2",
    "Mural",
    "Inhibitory Gpr165,Gda",
    "Excitatory Necab1,Gda",
    "Inhibitory Gpr165",
    "N/A", # only one vector, not mapped to anywhere
    "N/A", # Astrocyte + Sox4 contamination
    "N/A", # Contamination?
    "Inhibitory Amigo2",
    "Inhibitory Col25a1",
```

```
    "N/A",
    "Inhibitory Cxcl14",
    "N/A", # Astrocyte + Sox2 contamination
    "Mature OD 1",
    "Endothelial 3",
    "Microglia",
    "Excitatory Ebf3",
    "Mature OD 2",
    "Inhibitory Calcr",
    "Endothelial 1",

    "Excitatory Col25a1",
    "Excitatory Cbln2,Nos1",
    "Excitatory Irs4",
    "N/A", # Too few vectors
    "Inhibitory Coch",
    "N/A",
    "Inhibitory Tacr3",
    "Inhibitory Prlr",
    "Excitatory Omp",
    "Ependymal",

    "Excitatory Pcdh11x",
    "Inhibitory Irs4",
    "N/A",
    "Inhibitory Mixed Irs4",
    "Inhibitory Arhgap36,Mlc1",
    "Inhibitory Oprd1",
    "Inhibitory Fbxw13",
    "N/A",
]
```

[40]:
```
denovo_labels_final = []
exclude_indices = []
merge_indices = []
import re

p = re.compile(r' \(.*\)')

denovo_labels_short = [p.sub("", cl) for cl in denovo_labels]
for idx, cl in enumerate(denovo_labels_short):
    if cl == 'N/A':
        exclude_indices.append(idx)
        continue
    if cl in denovo_labels_final:
        continue
```

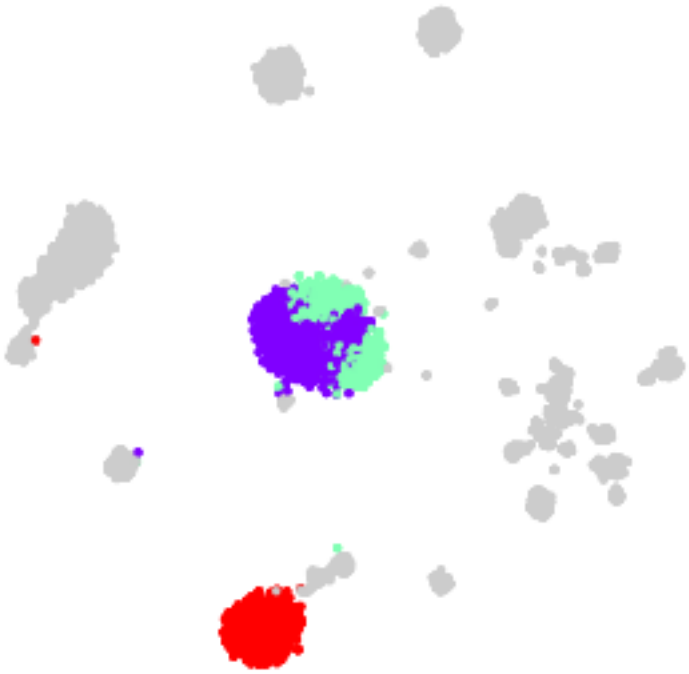
```
    denovo_labels_final.append(cl)
for cl in np.unique(denovo_labels_short):
    if cl == 'N/A':
        continue
    mask = [cl == e for e in denovo_labels_short]
    if np.sum(mask) > 1:
        merge_indices.append(np.where(mask)[0])
```

[41]:
```
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')
```

[41]: (-79.68918890309955, 82.88242620778705, -75.24307382978199, 79.1792310563445)

[42]:
```
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')
```

[42]:
```
(-80.4076785804484, 83.60091588513596, -75.94355948562917, 79.87971671219165)
```



[43]:
```
analysis.exclude_and_merge_clusters(exclude_indices, merge_indices,␣
 ↪centroid_correction_threshold=0.6)
```

[44]:
```
# Set cluster colors the same as category colors
denovo_celltype_colors = []
for cl in denovo_labels_final:
    for c_cl in cell_class_colors:
        if c_cl in cl:
            col = cell_class_colors.get(c_cl)
```

```
            break
    denovo_celltype_colors.append(col)
```

[45]:
```
# Generate unique cluster colors as shades of category colors
denovo_celltype_colors_shades = np.array(denovo_celltype_colors, copy=True)
for col in set(denovo_celltype_colors):
    same_color_indices = np.where(np.array(denovo_celltype_colors) == col)[0]
    if len(same_color_indices) > 3:
        percentage = 5
    else:
        percentage = 30
    shades = generate_shades(col, len(same_color_indices), percentage)
    denovo_celltype_colors_shades[same_color_indices] = shades
```

[46]:
```
load_tsne('merged')
```

[47]:
```
import matplotlib.patheffects as PathEffects
plt.figure(figsize=[5, 5])
ds.plot_tsne(pca_dims=60, metric="correlation", s=5, run_tsne=False,␣
 ↪colors=denovo_celltype_colors_shades)
plt.gca().axis('off')
plt.tight_layout()
```

[48]:
```
heatmap_cell_class_ordered = [
    "Inhibitory",
    "Excitatory",
    "Mature OD",
    "Immature OD",
    "Astrocyte",
    "Microglia",
    "Ependymal",
    "Endothelial",
    "Mural",
]
```

[49]:
```
heatmap_clusters_ordered = []
for i, cate in enumerate(heatmap_cell_class_ordered):
    for j, cl in enumerate(denovo_labels_final):
        if cate in cl:
            heatmap_clusters_ordered.append(cl)
```

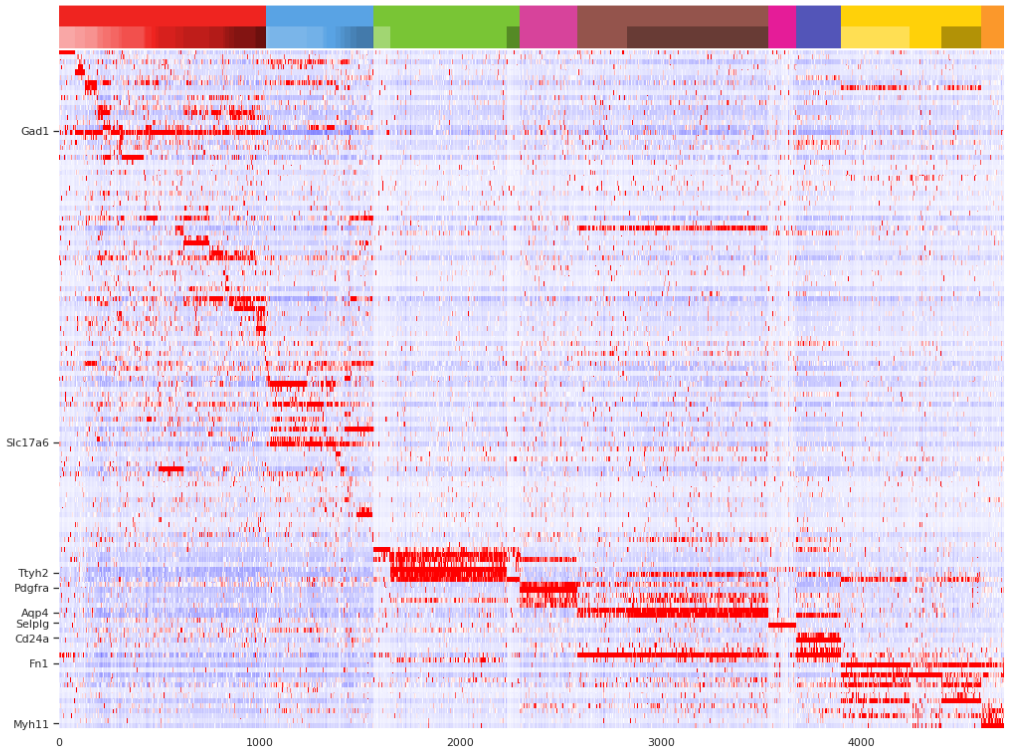
```
heatmap_clusters_index = [denovo_labels_final.index(cl) for cl in␣
↪heatmap_clusters_ordered]
```

[50]:
```
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_cellclass = np.zeros(np.sum(ds.filtered_cluster_labels != -1),␣
↪dtype='<U7')
col_colors_celltype = 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]
    heatmap_vectors[acc_idx:acc_idx+cl_vecs.shape[0], :] = cl_vecs

    col = denovo_celltype_colors[cl_idx]
    col_colors_cellclass[acc_idx:acc_idx+cl_vecs.shape[0]] = to_hex(col)

    col = denovo_celltype_colors_shades[cl_idx]
    col_colors_celltype[acc_idx:acc_idx+cl_vecs.shape[0]] = to_hex(col)

    acc_idx += cl_vecs.shape[0]
```

[51]:
```
heatmap_genes_index = []
heatmap_genes_ordered = []
_, i = np.unique(col_colors_celltype, return_index=True)
uc = col_colors_celltype[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_celltype == 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]
```

[52]:
```
import seaborn as sns
gene_exp_heatmap = heatmap_vectors.T
gene_exp_heatmap = preprocessing.scale(gene_exp_heatmap)
heatmap_genes_selected = ["Gad1", "Slc17a6", "Ttyh2", "Pdgfra", "Aqp4",␣
↪"Selplg", "Cd24a", "Fn1", "Myh11"]
```

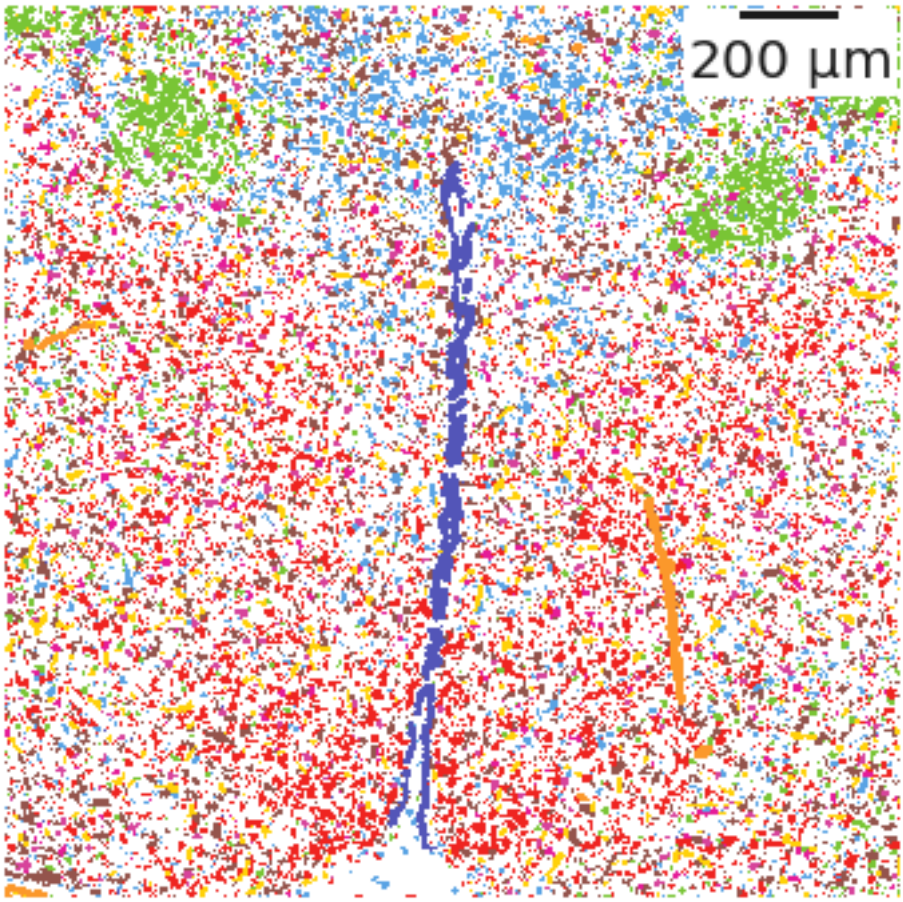
```
heatmap_genes_selected_indices = [heatmap_genes_ordered.index(gene) for gene in␣
↪heatmap_genes_selected]
g = sns.clustermap(gene_exp_heatmap, figsize=[20, 15],␣
↪yticklabels=heatmap_genes_selected,
                   cmap='bwr', row_cluster=False, col_cluster=False,
                   col_colors=[col_colors_cellclass, col_colors_celltype],␣
↪xticklabels = 1000, vmin=-2.5, vmax=2.5)
g.cax.set_visible(False)
g.ax_heatmap.set_yticks(heatmap_genes_selected_indices)
g.ax_heatmap.tick_params(labelright=False, labelleft=True, right=False,␣
↪left=True)
```



[53]:
```
analysis.map_celltypes()
analysis.filter_celltypemaps(min_norm=filter_method,␣
↪filter_params=filter_params, min_r=0.6, min_blob_area=7, fill_blobs=True)
```

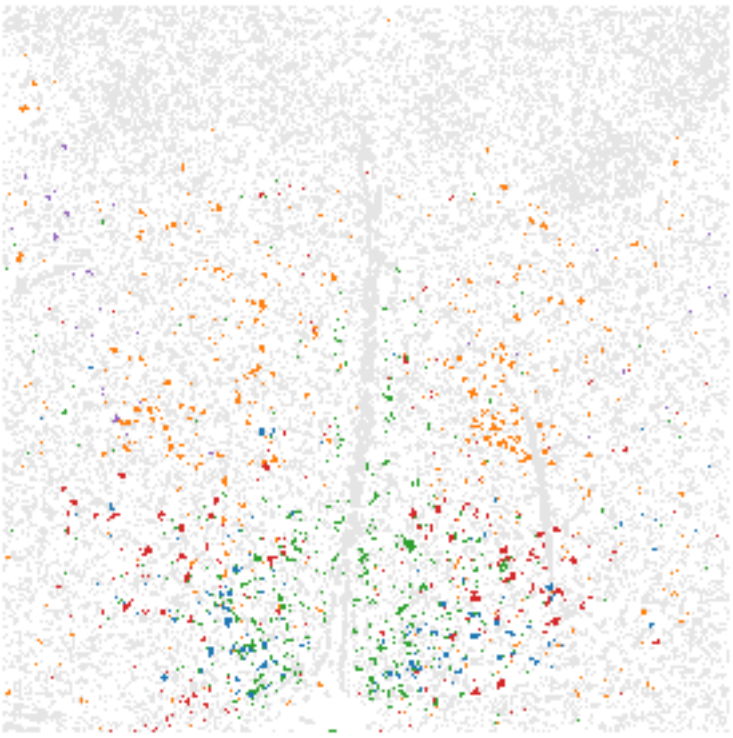
[54]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltypes_map(rotate=3, background='white',␣
↪colors=denovo_celltype_colors, set_alpha=False, z=4)
from matplotlib_scalebar.scalebar import ScaleBar
```

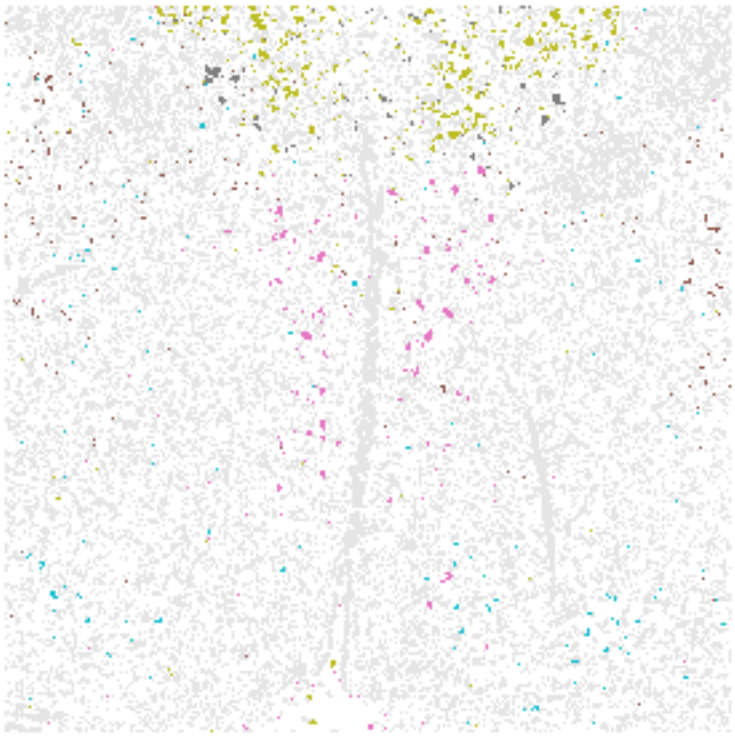
```
scalebar = ScaleBar(1, 'um', pad=0.1, font_properties={"size": 20})
plt.gca().add_artist(scalebar)
plt.gca().axis('off')
plt.tight_layout()
```



[55]:
```
indices = [[39, 16, 45, 34, 14], [19, 42, 25, 8, 36]]
colors = plt.get_cmap('tab10')(range(10))
#np.random.shuffle(colors)
i = 0
for subindices in indices:
    plt.figure(figsize=[5, 5])
    ctmap = np.zeros([ds.filtered_celltype_maps.shape[1], ds.
 ↪filtered_celltype_maps.shape[0], 4])
    ctmap[ds.filtered_celltype_maps[..., 0].T > -1] = [0.9, 0.9, 0.9, 1]
    for idx in subindices:
        cluster_color = to_rgba(denovo_celltype_colors[idx])
        ctmap[ds.filtered_celltype_maps[..., 0].T == idx] = colors[i]
        plt.imshow(ctmap)
```

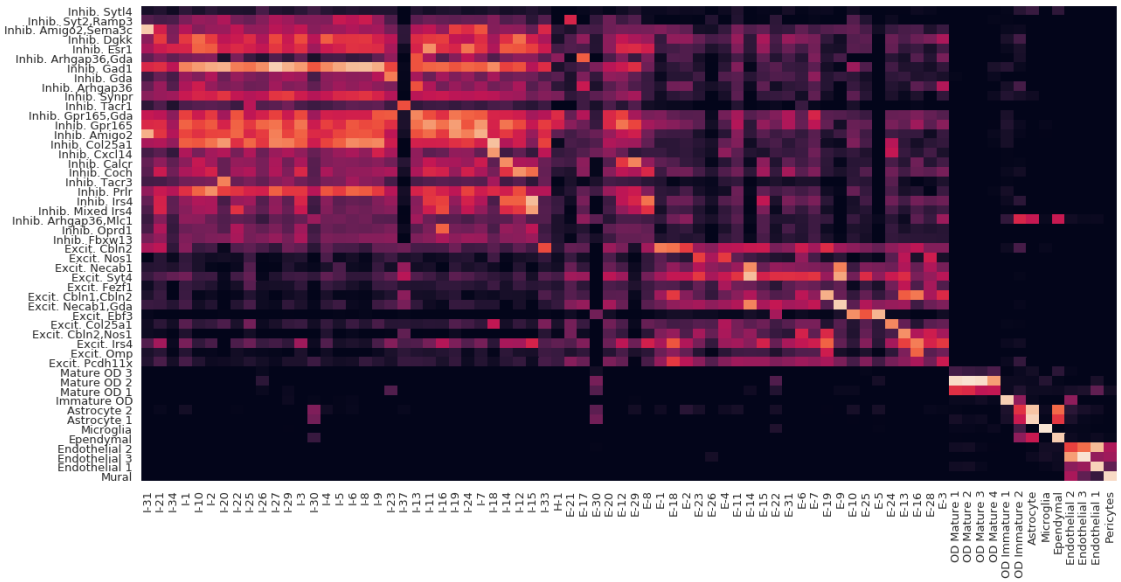
```
        plt.xlim([0, ds.celltype_maps.shape[0]])
        plt.ylim([0, ds.celltype_maps.shape[1]])
        plt.axis('off')
        i += 1
```





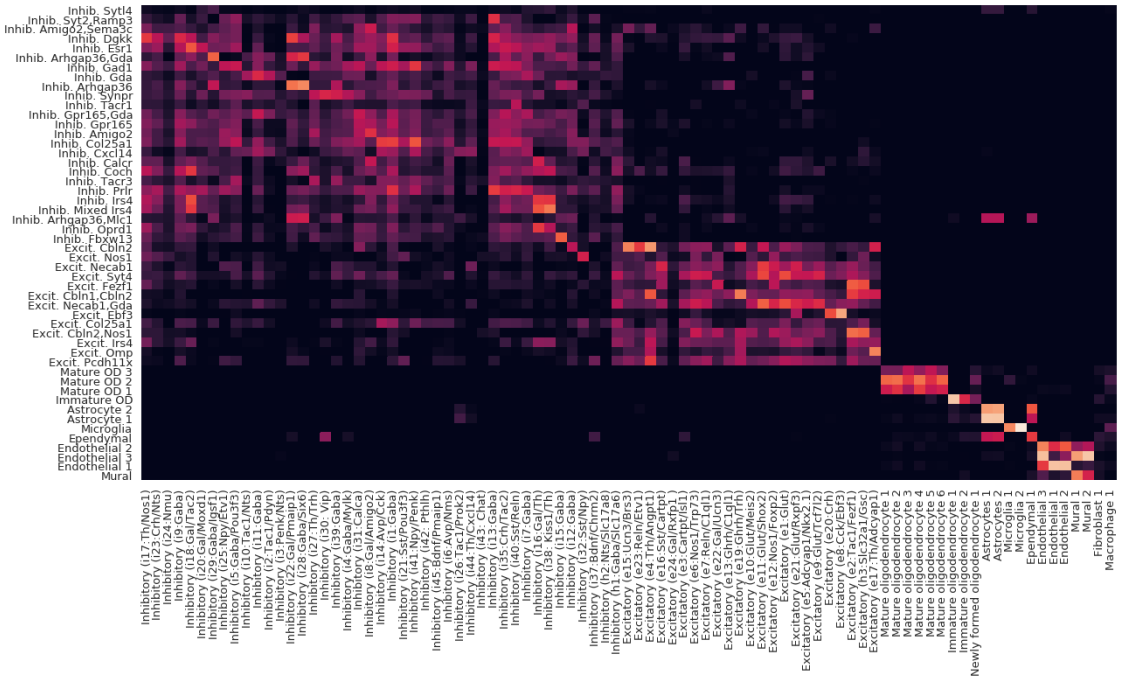
[56]:
```
centroid_corrs = np.zeros([len(merfish_centroids), len(ds.centroids)])

for i, mc in enumerate(merfish_centroids):
    for j, dc in enumerate(ds.centroids):
        centroid_corrs[i, j] = ssam.utils.corr(mc, dc)
centroid_corrs = centroid_corrs[:, heatmap_clusters_index]

heatmap_cls_index = []
heatmap_cls_ordered = []
cur_pos = 0
for cate in ['I-', 'H-', 'E-', 'OD Mature', 'OD Immature', 'Astrocyte',
             'Microglia', 'Ependymal', 'Endothelial', 'Pericytes']:
    cate_indices = np.where([cate in e for e in merfish_uniq_labels])[0]
    max_corr_indices = np.argmax(centroid_corrs[cate_indices, :], axis=1)
    for i in np.unique(max_corr_indices):
        cl_cl_indices = cate_indices[np.where(max_corr_indices == i)[0]]
        heatmap_cls_index += list(cl_cl_indices)
        heatmap_cls_ordered += list(np.
 ↪array(merfish_uniq_labels)[cl_cl_indices])
centroid_corrs = centroid_corrs[heatmap_cls_index, :]

plt.figure(figsize=[20, 10])
ax_heatmap = sns.heatmap(centroid_corrs.T,
```

```
                          xticklabels=merfish_uniq_labels[heatmap_cls_index],
                          yticklabels=heatmap_clusters_ordered,
                          vmin=0, vmax=1, cbar=False)
ax_heatmap.set_xticklabels(merfish_uniq_labels[heatmap_cls_index], fontsize =␣
↪13)
ax_heatmap.set_yticklabels(shorten_labels(heatmap_clusters_ordered), fontsize =␣
↪13)
pass
```



[57]:
```
centroid_corrs = np.zeros([len(scrna_centroids), len(ds.centroids)])

for i, mc in enumerate(scrna_centroids):
    for j, dc in enumerate(ds.centroids):
        centroid_corrs[i, j] = ssam.utils.corr(mc, dc)
centroid_corrs = centroid_corrs[:, heatmap_clusters_index]

heatmap_cls_index = []
heatmap_cls_ordered = []
cur_pos = 0
for cate in ['Inhibitory (i', 'Inhibitory (h', 'Excitatory', 'Mature',
             'Immature', 'Newly formed', 'Astrocytes', 'Microglia',␣
↪'Ependymal', 'Endothelial',
             'Mural', 'Fibroblast', 'Macrophage']:
    cate_indices = np.where([cate in e for e in scrna_uniq_labels])[0]
    max_corr_indices = np.argmax(centroid_corrs[cate_indices, :], axis=1)
    for i in np.unique(max_corr_indices):
        cl_cl_indices = cate_indices[np.where(max_corr_indices == i)[0]]
```

```
        heatmap_cls_index += list(cl_cl_indices)
        heatmap_cls_ordered += list(np.array(scrna_uniq_labels)[cl_cl_indices])
centroid_corrs = centroid_corrs[heatmap_cls_index, :]

plt.figure(figsize=[20, 10])
ax_heatmap = sns.heatmap(centroid_corrs.T,␣
 ↪yticklabels=heatmap_clusters_ordered, xticklabels=np.
 ↪array(scrna_uniq_labels)[heatmap_cls_index], vmin = 0, vmax = 1, cbar =␣
 ↪False)
ax_heatmap.set_yticklabels(shorten_labels(heatmap_clusters_ordered), fontsize =␣
 ↪13)
ax_heatmap.set_xticklabels(np.array(scrna_uniq_labels)[heatmap_cls_index],␣
 ↪fontsize = 13)
pass
```



[58]:
```
for idx in range(len(ds.centroids)):
    plt.figure(figsize=[50, 15])
    ds.plot_diagnostic_plot(idx,
                            cluster_name=denovo_labels_short[idx],
                            cluster_color=denovo_celltype_colors[idx])
    plt.tight_layout()
    plt.savefig('diagplots_MERFISH/diagplot_centroid_%d_after_merging_removing.
 ↪png'%idx)
    plt.close()
```

Find cell-class domains

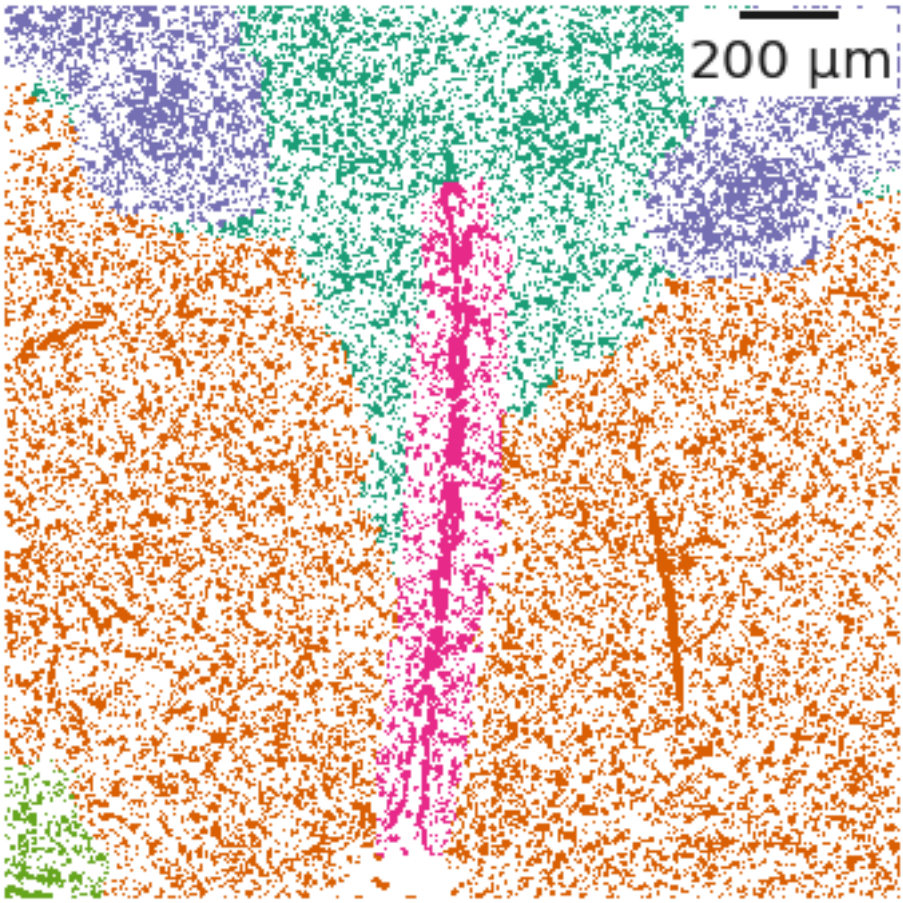
[59]:
```
# Save the current cell-type map
filtered_celltype_maps_backup = np.array(ds.filtered_celltype_maps, copy=True)
```

[60]:
```
# Convert cell-type map to cell-class map
ds.filtered_celltype_maps = np.zeros_like(filtered_celltype_maps_backup)
ds.filtered_celltype_maps[:] = -1
for cate_idx, cate in enumerate(heatmap_cell_class_ordered):
    cl_indices = np.where(np.array(denovo_celltype_colors) ==␣
 ↪cell_class_colors[cate])[0]
    for cl_idx in cl_indices:
        ds.filtered_celltype_maps[filtered_celltype_maps_backup == cl_idx] =␣
 ↪cate_idx
```

[61]:
```
analysis.bin_celltypemaps(step=10, radius=100)
```

[62]:
```
analysis.find_domains(n_clusters=20, merge_remote=True, merge_thres=0.8,␣
 ↪norm_thres=4000)
```

[63]:
```
from matplotlib.colors import ListedColormap
plt.figure(figsize=[5, 5])
cmap = ListedColormap([
    '#66a61e',
    '#d95f02',
    '#e7298a',
    '#1b9e77',
    '#7570b3',
])
ds.plot_domains(rotate=3, cmap=cmap, z=0)
plt.axis('off')
scalebar = ScaleBar(1, 'um', pad=0.1, font_properties={"size": 20})
plt.gca().add_artist(scalebar)
plt.tight_layout()
```

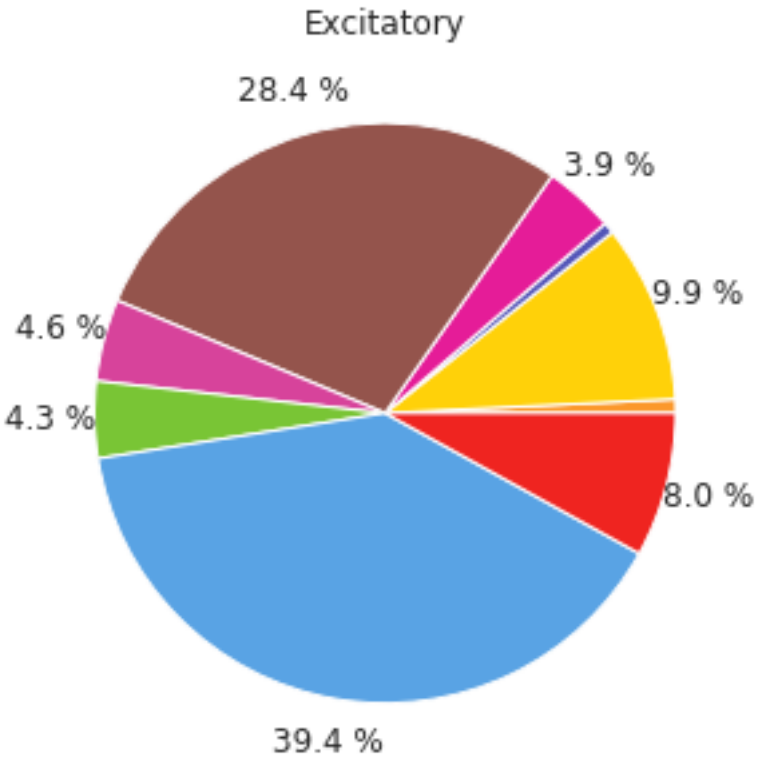


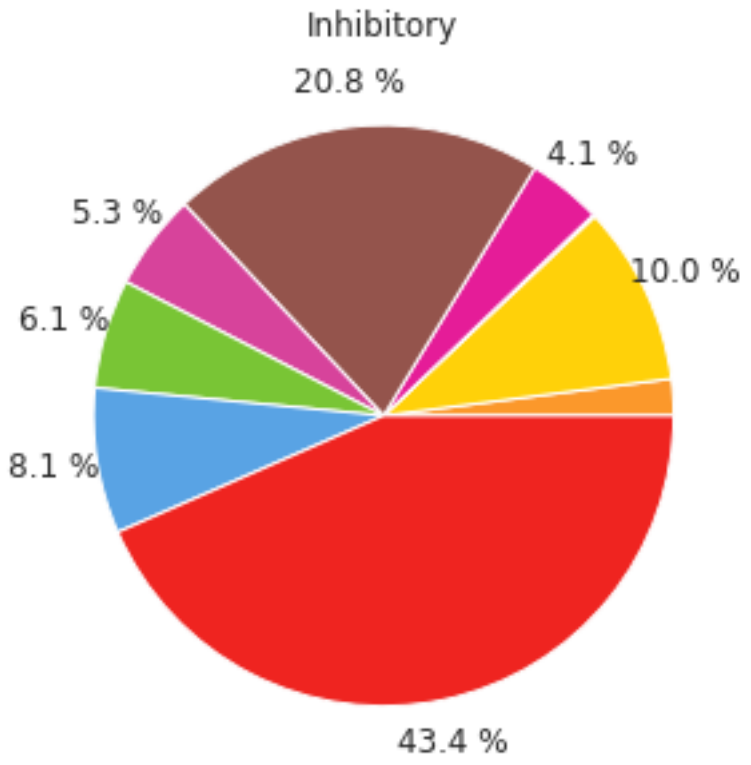
[64]:
```
analysis.calc_cell_type_compositions()
```

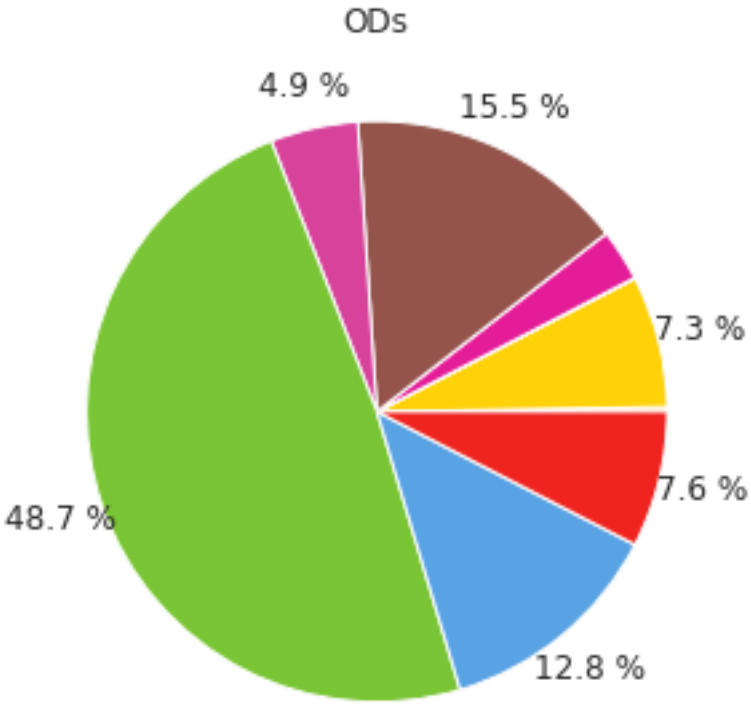
[65]:
```
domain_labels = [
    'Boundary',
    'Inhibitory',
    'Ventricle',
    'Excitatory',
    'ODs',
]
```

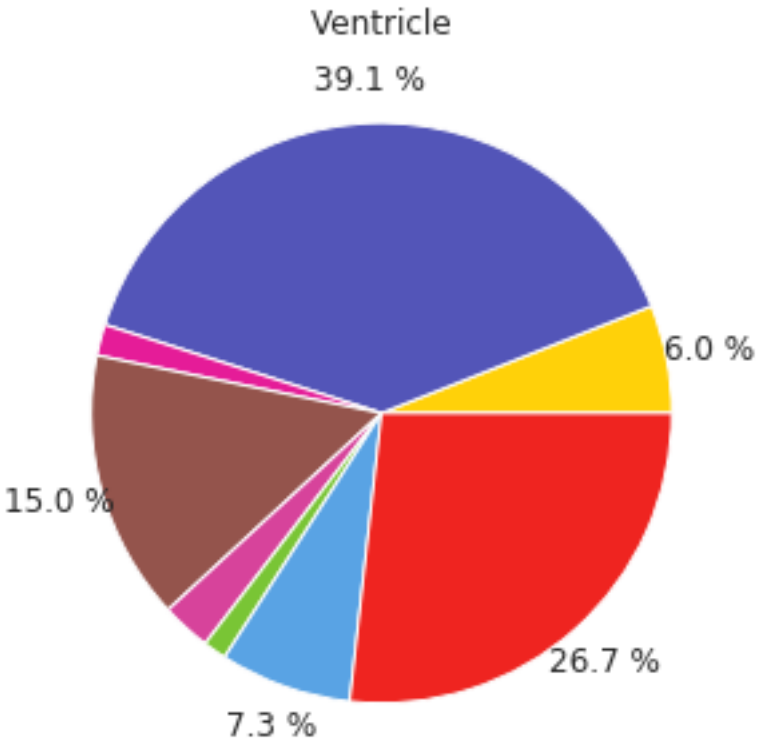
[66]:
```
cols = [cell_class_colors[cl] for cl in heatmap_cell_class_ordered]
cell_class_orders = range(len(heatmap_cell_class_ordered))[::-1]
for domain_idx in [3, 1, 4, 2, 0]:
    plt.figure(figsize=[5, 5])
    ds.plot_celltype_composition(domain_idx,
                                 cell_type_colors=cols,
                                 cell_type_orders=cell_class_orders,
                                 label_cutoff=0.03)
```

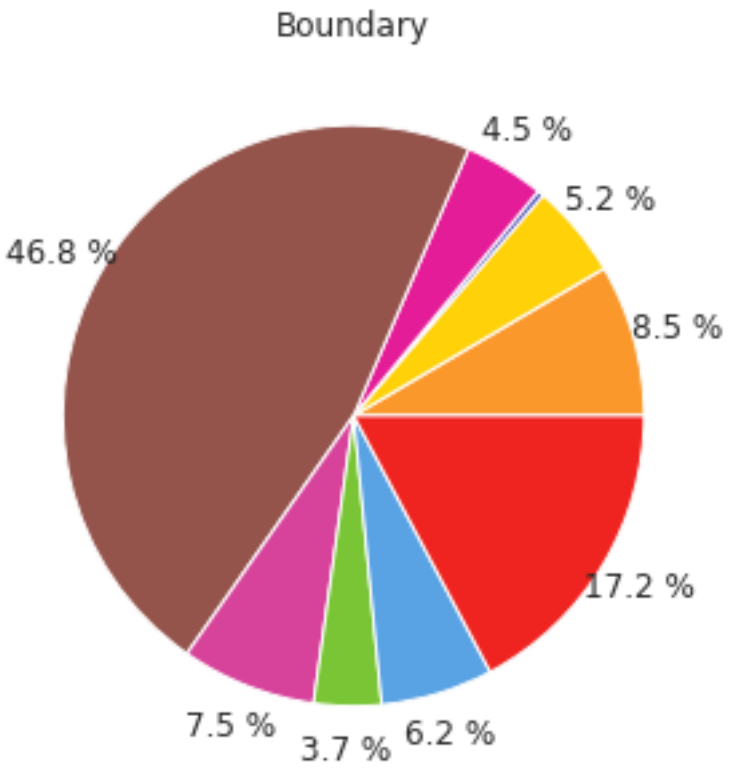
```
    plt.title(domain_labels[domain_idx])
```




Inhibitory
20.8 %
4.1 %
5.3 %
10.0 %
6.1 %
8.1 %
43.4 %

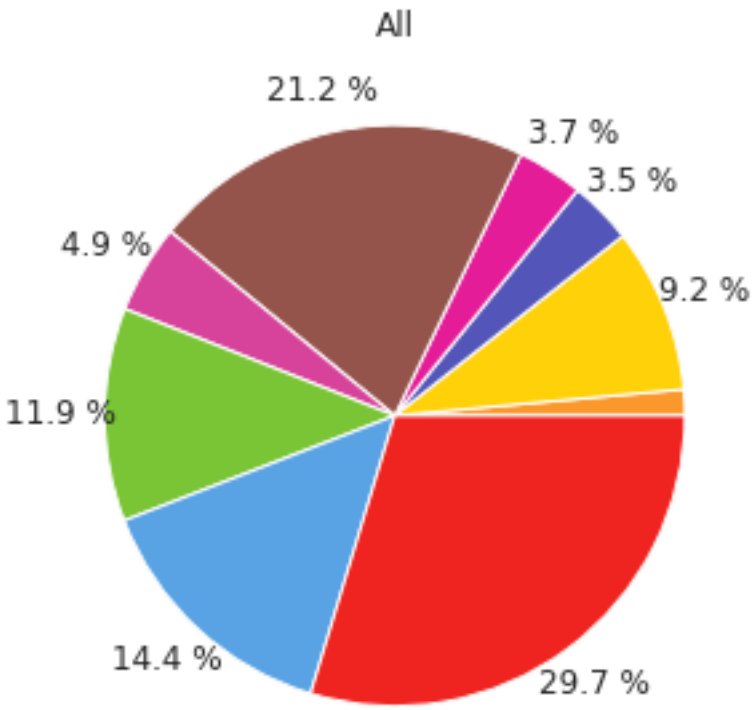

ODs
4.9 %
15.5 %
7.3 %
48.7 %
7.6 %
12.8 %


Ventricle
39.1 %
6.0 %
15.0 %
26.7 %
7.3 %



[67]:
```
plt.figure(figsize=[5, 5])
ds.plot_celltype_composition(domain_index=5,
                             cell_type_colors=cols,
                             cell_type_orders=cell_class_orders,
                             label_cutoff=0.03)
plt.title('All')
```

[67]:
```
Text(0.5, 1.0, 'All')
```



Reproduce original result in Moffit *et al.*

[68]:
```
ct_convert_dic = {
    "Astrocyte": "Astrocyte",
    "Endothelial 1": "Endothelial",
    "Endothelial 2": "Endothelial",
    "Endothelial 3": "Endothelial",
    "Ependymal": "Ependymal",
    "Excitatory": "Excitatory",
    "Inhibitory": "Inhibitory",
    "Microglia": "Microglia",
    "OD Immature 1": "Immature OD",
    "OD Immature 2": "Immature OD",
    "OD Mature 1": "Mature OD",
    "OD Mature 2": "Mature OD",
    "OD Mature 3": "Mature OD",
    "OD Mature 4": "Mature OD",
    "Pericytes": "Mural",
}
```

```
from matplotlib.colors import to_rgba
seg_ct_col_dic = {}
for c1, c2 in ct_convert_dic.items():
    seg_ct_col_dic[c1] = to_rgba(cell_class_colors[c2])
```

[69]:
```
from collections import defaultdict
cell_boundaries = []
with open('zenodo/MERFISH/raw_data/cellboundaries_example_animal.csv') as f:
    f.readline()
    for line in f:
        entries = line.rstrip().split(',')
        if len(entries) == 0:
            break
        if entries[1] == '':
            continue
        x = [float(e) for e in entries[1].split(';')]
        y = [float(e) for e in entries[2].split(';')]
        try:
            cell_type = cid_dic[entries[0]]
            cell_boundaries.append((x, y, cell_type))
        except KeyError:
            pass
```

[70]:
```
from skimage.draw import polygon

img_shape = [1800, 1800]
seg_im = np.zeros(img_shape + [4])
minxy = [9999999999, 9999999999]
maxxy = [-9999999999, -9999999999]
for x, y, _ in cell_boundaries:
    minx, miny, maxx, maxy = min(x), min(y), max(x), max(y)
    if minxy[0] > minx:
        minxy[0] = minx
    if minxy[1] > miny:
        minxy[1] = miny
    if maxxy[0] < maxx:
        maxxy[0] = maxx
    if maxxy[1] < maxy:
        maxxy[1] = maxy
for x, y, cell_type in cell_boundaries:
    X, Y = [], []
    for xx, yy in zip(x, y):
        try:
            xxx, yyy = int(xx - minxy[0]), int(yy - minxy[1])
            X.append(xxx)
            Y.append(yyy)
        except ValueError:
```

```
            continue
    seg_im[polygon(X, Y, img_shape)] = seg_ct_col_dic[cell_type]
#print(maxxy[0] - minxy[0])
#print(maxxy[1] - minxy[1])
```

[71]:
```
plt.figure(figsize=[5, 5])
plt.imshow(seg_im.swapaxes(0, 1))
plt.xlim([5, 1800])
plt.ylim([0, 1795])
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.gca().axis('off')
plt.show()
```