`napari-cosmx` essentials

Evelyn Metzger

2024-06-17

Contents

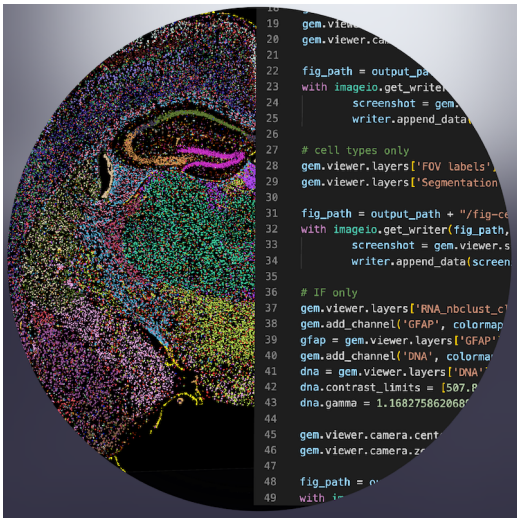

Figure 1: Drawing that represents the duality of `napari-cosmx`. On the left side, cell types within a mouse coronal hemisphere are shown in an interactive Graphical User Interface. In addition to creating images interactively, the right side highlights that images can be generated programmatically. Both sides of `napari-cosmx` are discussed in this post.

1 Introduction

This post is the second installment of the napari series. In the first blog post I introduced the `napari-cosmx` plugin, how CosMx™ Spatial Molecular Imager (SMI) data can be viewed as layers within napari, and a method for processing, or "stitching", raw data that are exported from AtoMx™ Spatial Informatics Portal (SIP).

One of the things that I love about the `napari-cosmx` plugin is its duality. It's flexible enough to quickly explore SMI data in a Graphical User Interface (GUI) yet robust enough to script reproducible results and tap into the underlying python objects. In this post, I'll walk through some of the basic ways in which we can use `napari-cosmx`to view SMI data. I'll make use of this duality by sharing a combination of GUI and programmatic tips.

> **Note**
>
> This is not intended to be official documentation for the `napari-cosmx` plugin. The tips herein are not an exhaustive list of features and methods.

- Section 2 shows how to preprocess the example dataset. If you are using your AtoMx-exported SMI data, this section is optional
- Section 3 shows basic GUI tips for interacting with SMI data

- Section 4 provides several examples of recapitulating the aesthetics seen with the GUI as well as advanced ways we can fine-tune images and more

2 The Example Dataset

The example dataset that I am using is the mouse coronal hemisphere FFPE that is available to download from NanoString's website here. If you are following along with your AtoMx exported data, you can skip most of these pre-processing steps as they are not required (but see Section 2.1.2 if you would like to view metadata).

> **Note**
>
> Large memory (RAM) might be required to work with raw images. Stitching the example data on a laptop might not work for everyone. The raw data size for this example is 183 GBs. Not all raw data are needed to stitch, however, and users can exclude the `CellStatsDir/Morphology3D` folder if downloading locally. If excluding this folder, the raw data is closer to 35 GBs.
> The computer I used to stitch was an M1 Macbook Pro. Processing this 130 FOV, mouse 1K data set took about 10 minutes, ~700% CPU, and a peak memory usage of about 12 GBs (swap space was also used). The size of the napari files combined was an additional 22 GBs of disk space.

2.1 Pre-processing example data

Once downloaded, unzip the `HalfBrain.zip` file on your computer or external hard drive. The format for this dataset differs from the expected AtoMx SIP export so a preprocessing step is necessary.

When uncompressed, the raw data in the HalfBrain folder are actually nested like this:

2.1.1 Expected Raw Data Format

In order for `napari-cosmx` to stitch this non-AtoMx example dataset, we'll need to rearrange the folders so that the nested raw data are at the top level. After rearrangement, the proper file structure should look like this:

There are a few ways to rearrange. The first method retains the original folder structure and simply makes symbolic links to the data in the expected format. Here's how to do it in unix/mac (Windows not shown).

Listing 1 `Terminal`

```
tree -L 4

   AnalysisResult
      HalfBrain_20230406_205644_S1
          AnalysisResults <-- **Raw Data Folder**
|             cp7bjyp7pm
   CellStatsDir
      HalfBrain_20230406_205644_S1
          CellStatsDir <-- **Raw Data Folder**
              CellComposite
              CellOverlay
              FOV001
              FOV002
|           ...
              Morphology2D
              RnD
   RunSummary
       HalfBrain_20230406_205644_S1
           RunSummary <-- **Raw Data Folder**
               Beta12_Affine_Transform_20221103.csv
               FovTracking
               Morphology_ChannelID_Dictionary.txt
               Run1000_20230406_205644_S1_Beta12_ExptConfig.txt
               Run1000_20230406_205644_S1_Beta12_SpatialBC_Metrics4D.csv
               Shading
               c902.fovs.csv
               latest.fovs.csv
```

Alternatively, we could manually move folders. Specifically, in your Finder window, create a folder named `RawData`. Then, move:

- `HalfBrain/AnalysisResult/HalfBrain_20230406_205644_S1/AnalysisResults` to `RawData/AnalysisResults`
- `HalfBrain/CellStatsDir/HalfBrain_20230406_205644_S1/CellStatsDir` to `RawData/CellStatsDir`
- `HalfBrain/RunSummary/HalfBrain_20230406_205644_S1/RunSummary` to `RawData/RunSummary`

Once the file structure is properly formatted, use the stitching widget method from an earlier blog post to create the mouse brain napari files.

Listing 2 `Terminal`

```
tree -L 2

.
   AnalysisResults
       cp7bjyp7pm
   CellStatsDir
       CellComposite
       CellOverlay
       FOV001
       FOV002
...
       Morphology2D
       RnD
   RunSummary
        Beta12_Affine_Transform_20221103.csv
        FovTracking
        Morphology_ChannelID_Dictionary.txt
        Run1000_20230406_205644_S1_Beta12_ExptConfig.txt
        Run1000_20230406_205644_S1_Beta12_SpatialBC_Metrics4D.csv
        Shading
        c902.fovs.csv
        latest.fovs.csv
```

2.1.2 Adding metadata

We will also use the cell typing data from the Seurat file. Let's include the following metadata columns:

- `RNA_nbclust_clusters`: the cell typing results (with abbreviated names)
- `RNA_nbclust_clusters_long`: (optional) human-readable cell type names
- `spatialClusteringAssignments`: spatial niche assignments

Note that the Seurat file contains two sections of mouse brain samples. We need to filter the metadata to include only those cells from `Run1000_S1_Half`. Note that when preparing the metadata for napari, the cell ID must be the first column (*i.e.*, see the `relocate` verb in the code below).

```
# This is R code
library(Seurat)
library(plyr)
```

Listing 3 `Terminal`

```
# Terminal in Mac/Linux

# cd to folder containing HalfBrain. Then,

mkdir -p RawFiles && cd $_
ln -s ../HalfBrain/AnalysisResult/HalfBrain_20230406_205644_S1/AnalysisResults .
ln -s ../HalfBrain/CellStatsDir/HalfBrain_20230406_205644_S1/CellStatsDir .
ln -s ../HalfBrain/RunSummary/HalfBrain_20230406_205644_S1/RunSummary .
```

```
library(dplyr)
# sem_path will be wherever you downloaded your Seurat object
sem_path <- "/path/to/your/muBrainRelease_seurat.RDS"
sem <- readRDS(sem_path)
meta <- sem@meta.data %>%
  filter(Run_Tissue_name=="Run1000_S1_Half") %>%
  select(RNA_nbclust_clusters,
  RNA_nbclust_clusters_long,
  spatialClusteringAssignments)

meta$cell_ID <- row.names(meta) # adds cell_ID column
rownames(meta) <- NULL
meta <- meta %>% relocate(cell_ID) # moves cell_ID to first column position
write.table(meta, file="/path/to/inside/napari-ready-folder/_metadata.csv",
            sep=",", col.names=TRUE, row.names=FALSE, quote=FALSE)
```

Now that the data are ready, drag and drop the slide folder into napari to launch the plugin.

3 Interacting with the GUI

This section focuses on features relevant to the `napari-cosmx` plugin. Users new to napari may find napari's general viewer tutorial helpful as well.

When we open a slide with `napari-cosmx`, by default there will be a few napari layers visible (Initial View tab; Figure 2). These include `FOV labels` and `Segmentation`. Clicking the eye icon next to a layer will change its visibility. Let's turn off those layers for a moment and visualize the cell types from the `RNA_nbclust_clusters` column (Cell Types tab; Figure 3). We can also color cells by their spatialClusteringAssignments values (Niches tab; Figure 4). In the `Color Cells` widget, we can control which cell types or niches we would like to view.

When we activate the `Metadata` layer, hovering over a given cell will display the metadata associated with it as a ribbon at the bottom of the application. To view the IF channels, use the `Morphology Images` widget, In Figure 5 (IF Channels tab) I turned off the visibility of the cell types, added GFAP in red and DNA in cyan, and zoomed into the hippocampus. When I click on a layer, it becomes the activate layer and I can use the `layer controls` widget to adjust attributes to that layer such as contrast limits, gamma, layer blending, and more. Finally, we can view raw transcripts (or proteins). Simply select the target and the color and click `Add layer`. In Figure 6 (Transcripts tab), I zoomed in on a section of the cortex and plotted *Calb1*.

Like other programs that use layers, napari allows the layers to be moved up/down and to blend (not shown below).

3.1 Initial View

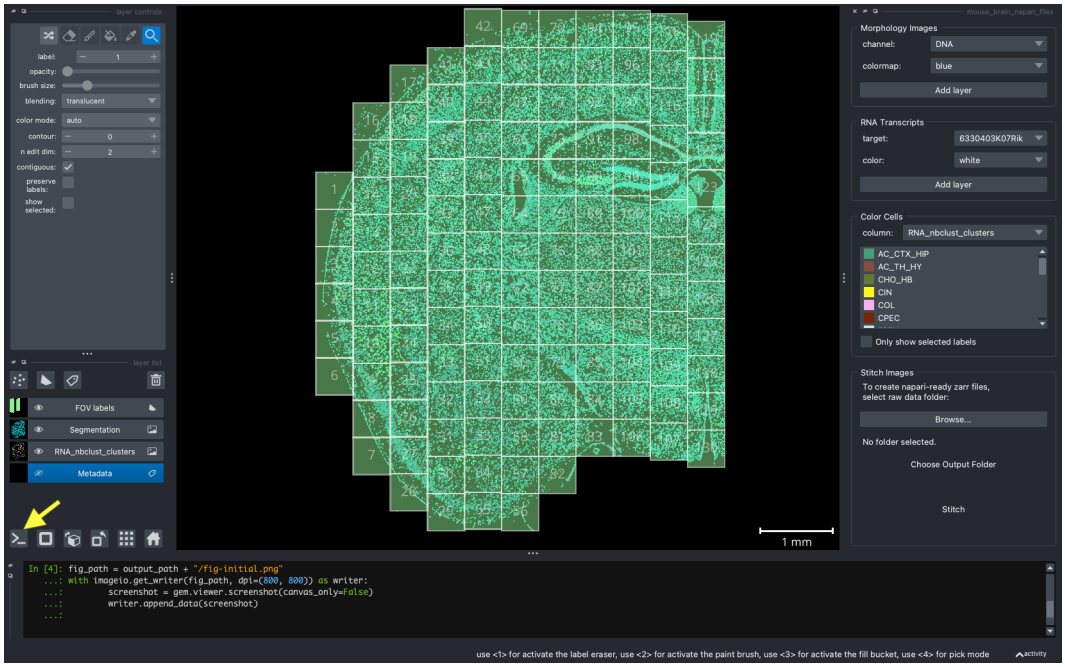

Figure 2: The initial view of the tissue shows the location of FOVs and cell boundary layers. Yellow arrow shows location of ipython terminal.

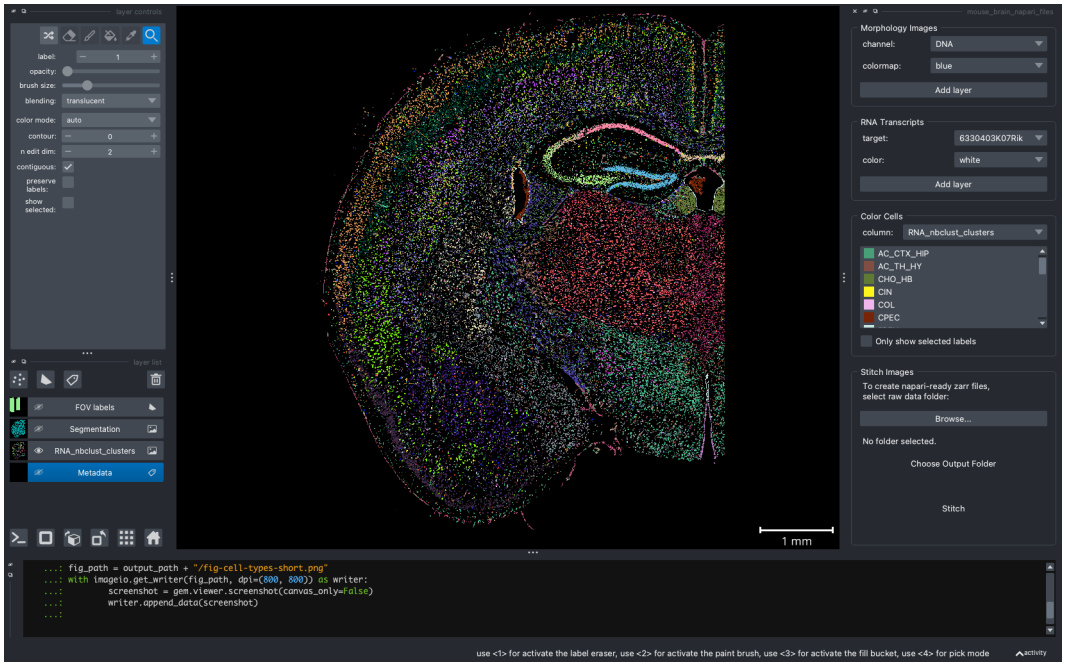

Figure 3: Same extent as Figure 2 and displaying the cell type from the 'RNA_nbclust_clusters'.

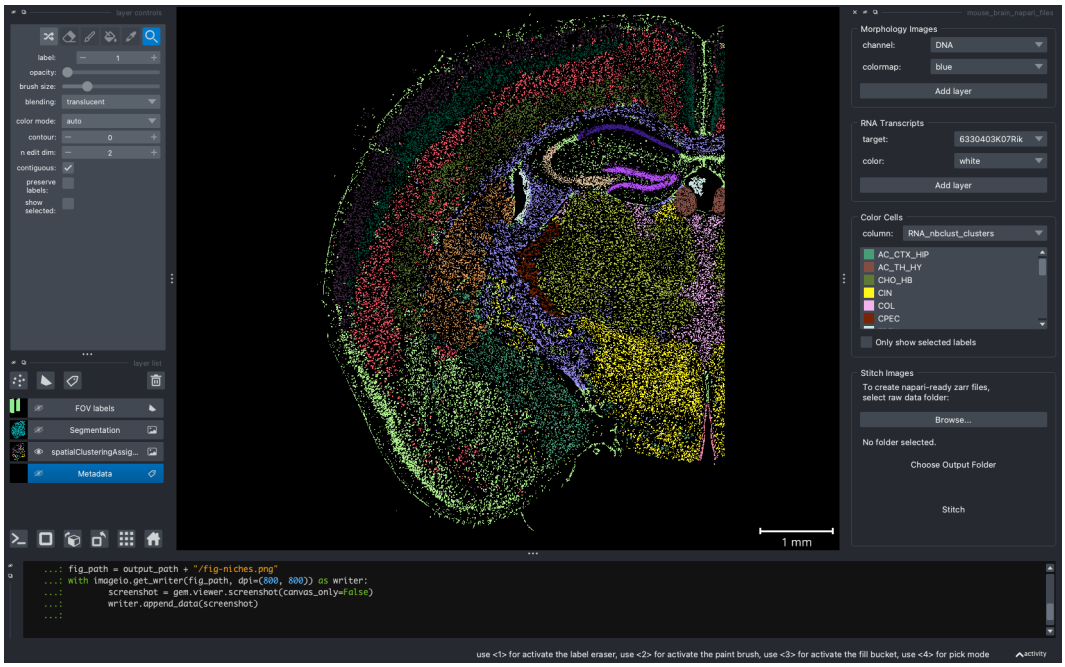

Figure 4: Cells colored by niche

3.2 Cell Types

3.3 Niches

3.4 IF Channels

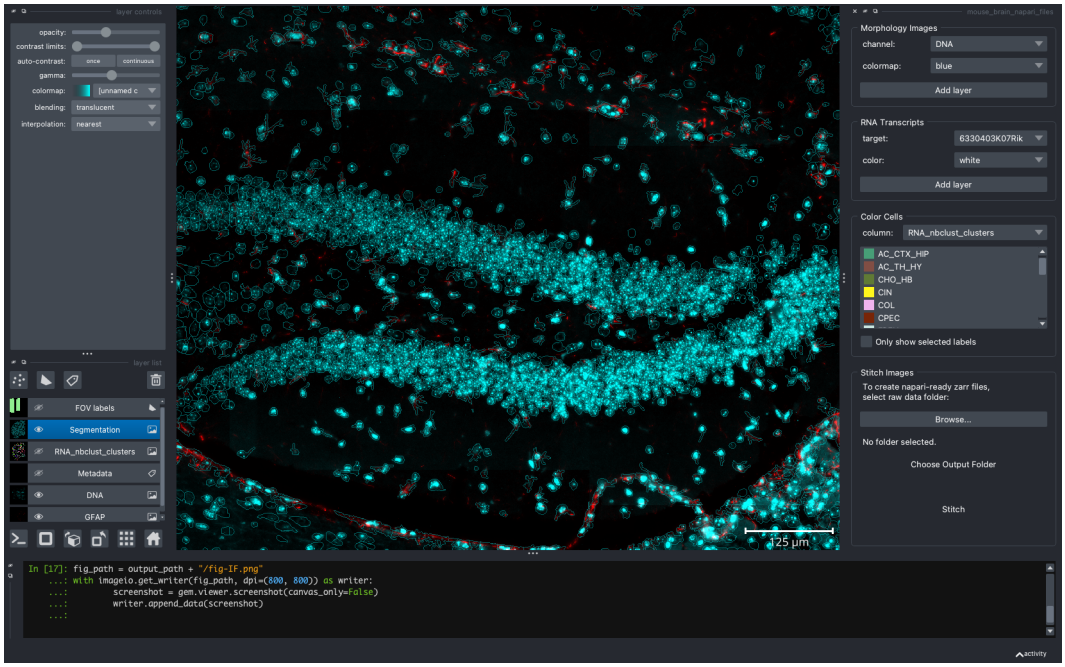

Figure 5: Hippocampal region of tissue with GFAP (red) and DNA (cyan).

3.5 Transcripts

To capture screenshots, simply click `File > Save Screenshot...`. The images above are captured "with viewer" but that is optional.

4 Scripting with `napari-cosmx`

This section is for advanced users who want finer control of the aesthetics.

Most of the items we've covered can also be accessed through various methods in the `gem` object that can found loaded in the `>_ ipython` interpreter (*i.e.*, yellow arrow in Figure 2). You may have noticed in the figures above that there was code being used to take the screenshots. Here's the full script that can help reproduce the figures above. I use reproducible scripts often. This

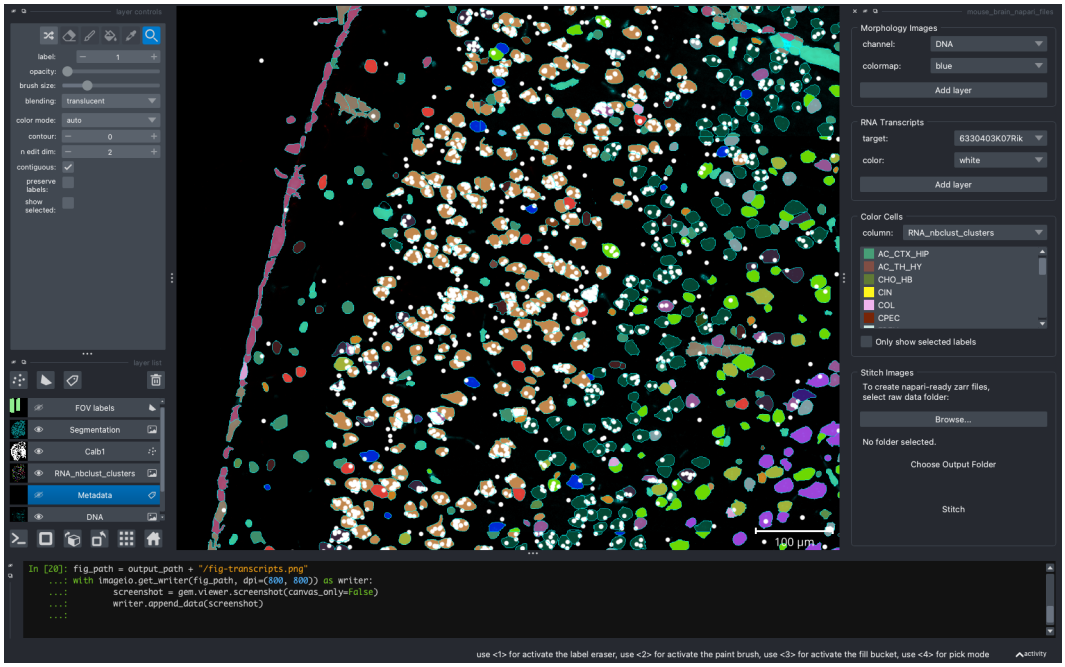

Figure 6: Expression of *Calb1* (white dots) in cortex layer I.

is because I may want to make slight changes to a figure down the road. For example, if a reviewer overall likes an image but asks for the cell colors to be different, I just need to change the colors in the code and the script will pan and zoom where needed, set the IF channels and contrasts, and reproduce other layers programmatically.

```
import imageio

output_path = 'path/to/store/figures'

## Initial
gem.show_widget()
gem.viewer.window.resize(1650,1100)
gem.viewer.camera.center = (0.0, 0.6830708351616575, -57.16103351264418)
gem.viewer.camera.zoom = 128.0

fig_path = output_path + "/fig-initial.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)

## Cell types only
```

```
gem.viewer.layers['FOV labels'].visible = False
gem.viewer.layers['Segmentation'].visible = False

fig_path = output_path + "/fig-cell-types-short.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)

## Niches
gem.color_cells('spatialClusteringAssignments')
fig_path = output_path + "/fig-niches.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)

# IF only
gem.color_cells('RNA_nbclust_clusters')
gem.viewer.layers['RNA_nbclust_clusters'].visible = False
gem.add_channel('GFAP', colormap = 'red')
gfap = gem.viewer.layers['GFAP']

gem.add_channel('DNA', colormap = 'cyan')
dna = gem.viewer.layers['DNA']
dna.contrast_limits = [208.39669421487605, 1328.5289256198346]
dna.gamma = 1.1682758620689655

gem.viewer.camera.center = (0.0, -0.7181216373638928, -55.314605674992876)
gem.viewer.camera.zoom = 1095.856465340922
gem.viewer.layers['Segmentation'].visible = True
gem.viewer.layers['Segmentation'].opacity = 0.371900826446281

fig_path = output_path + "/fig-IF.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)


# Transcripts
cell_type_layer = gem.viewer.layers['RNA_nbclust_clusters']
cell_type_layer.opacity = 0.9
cell_type_layer.visible = True
gem.viewer.camera.center = (0.0, -0.009723512204714457, -59.25760232977486)
```

```
gem.viewer.camera.zoom = 1204.3755331686673
gem.viewer.layers['Segmentation'].visible = True
gem.viewer.layers['Segmentation'].opacity = 0.6
gem.plot_transcripts(gene = "Calb1", color = 'white', point_size=50) # I

fig_path = output_path + "/fig-transcripts.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)
```

In practice, I use the GUI to adjust the settings (*e.g.*, zoom, opacity) and then "jot down" the results in my text editor. For example, when I zoom or pan to another location, that location can be found at:

```
gem.viewer.camera.zoom
gem.viewer.camera.center
```

Similarly, the contrast limits and gamma values for IF channels can be saved as well.

```
dna = gem.viewer.layers['DNA']
dna.contrast_limits = [208.39669421487605, 1328.5289256198346]
dna.gamma = 1.1682758620689655
```

Screenshots can be done programmatically with the napari's `screenshot` method and there are additional settings you can change (*e.g.*, just the canvas, scale) that we won't cover here.

```
fig_path = output_path + "/fig-transcripts.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)
```

There are also methods available in `napari-cosmx` that do not have the GUI equivalent. We won't be able to touch on all of these methods in this post but I want to highlight a few.

4.1 Color cells with outlines

We can plot the cell colors as boundaries instead of filled in polygons (Figure 7).

```
# gem.viewer.camera.center = (0.0, -0.5375926704126319, -54.7415680001114)
# gem.viewer.camera.zoom = 1371.524539264374

gfap.visible = False
dna.visible = False
gem.viewer.layers['Calb1'].visible = False
gem.viewer.layers['Npy'].visible = False
gem.viewer.layers['Targets'].visible = False

gem.viewer.camera.center = (0.0, -0.6346878790298397, -54.95271110236874)
gem.viewer.camera.zoom = 2113.6387223301786
gem.color_cells('RNA_nbclust_clusters', contour=2)

fig_path = output_path + "/fig-contours.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=True)
        writer.append_data(screenshot)
```

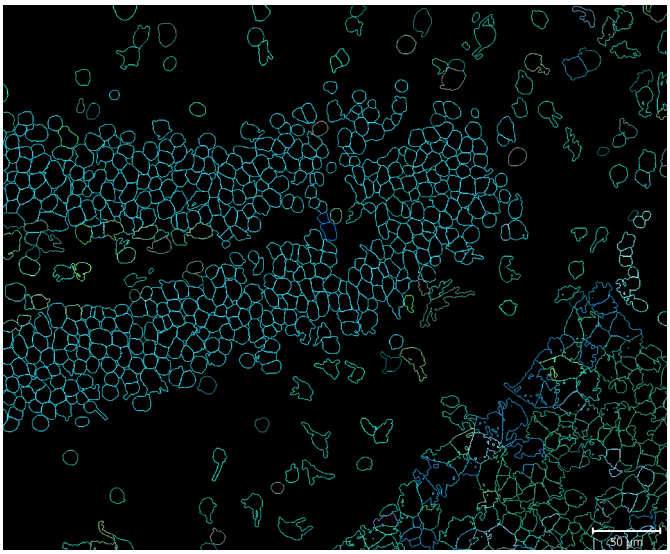

Figure 7: Cells types (or other metadata items) can be represented as cell boundaries.

4.2 Plot transcripts with an expanded color pallette

The GUI offers a handful of colors to plot transcripts. We can specify which color, by name or by hexcode, to plot. For example:

```
gem.plot_transcripts(gene = "Calb1", color = 'pink', point_size=20)
```

which is the same as

```
gem.plot_transcripts(gene = "Calb1", color = '#FFC0CB', point_size=20)
```

4.3 Plotting genes with list comprehensions

We can plot similar genes or targets with the same color. For example, the code that generated Figure 8 is here.

```
gem.viewer.camera.center = (0.0, -0.6346878790298397, -54.95271110236874)
gem.viewer.camera.zoom = 2113.6387223301786

df = gem.targets
filtered_df = df[df.target.str.contains("NegPrb")]

pandas_df = filtered_df.to_pandas_df()
negatives = pandas_df.target.unique().tolist()
[gem.plot_transcripts(gene = x, color = "white", point_size=20) for x in negatives];

fig_path = output_path + "/fig-negatives.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)
```

We can also supply of list of tuples where each tuple is a target and a color.

```
genes = [('Npy', "magenta"), ("Calb1", "white")]
[gem.plot_transcripts(gene = x[0], color = x[1], point_size=20) for x in genes];

for x in negatives:
  gem.viewer.layers[x].visible = False

gem.color_cells('RNA_nbclust_clusters') # reset to filled contours
cell_type_layer = gem.viewer.layers['RNA_nbclust_clusters']
cell_type_layer.opacity = 0.9
cell_type_layer.visible = True
gem.viewer.camera.center = (0.0, -0.026937869510583412, -59.20560304046731)
gem.viewer.camera.zoom = 3820.667999302201
```

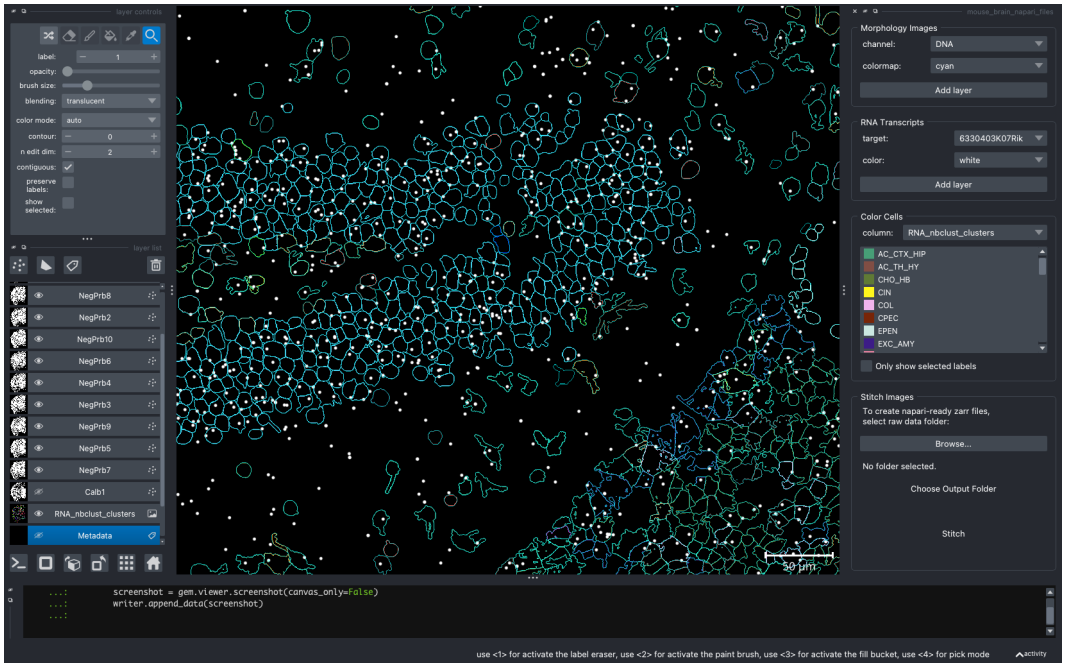

Figure 8: Same extent as Figure 7 but with negatives shown in white.

```
gem.viewer.layers['Segmentation'].visible = True
gem.viewer.layers['Segmentation'].opacity = 0.6

fig_path = output_path + "/fig-crowded-tx.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)
```

4.4 Changing transcript transparency

Sometimes transcripts can be stacked on top of each other to the point that it's difficult to qualitatively determine the number of transcripts. Adjusting the transcript opacity of the layer in the GUI only changes the transparency of a single point. But it's possible to change all points using the `ipython` interpreter.

```
gem.viewer.layers['Npy'].opacity = 0.5
fig_path = output_path + "/fig-tx-opacity.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
```

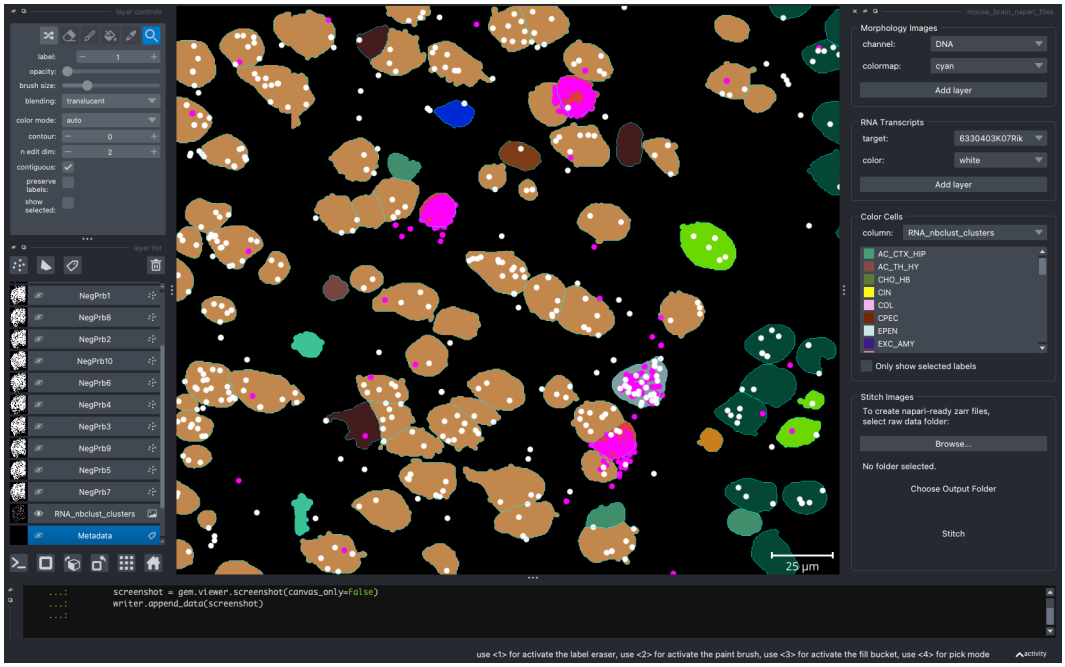

Figure 9: Cortical layer with *Npy* (magenta) and *Calb1* (white).

```
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)
```

4.5 Center to a particular FOV

While zooming (`gem.viewer.camera.zoom`) and panning (`gem.viewer.camera.center`) can control the exact location of the camera, you can programmatically go to a particular fov with the `center_fov` method.

```
# center to fov 123 and zoom in a little (i.e., buffer > 1).
gem.center_fov(fov=123, buffer=1.2)

fig_path = output_path + "/fig-center-to-fov.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)
```

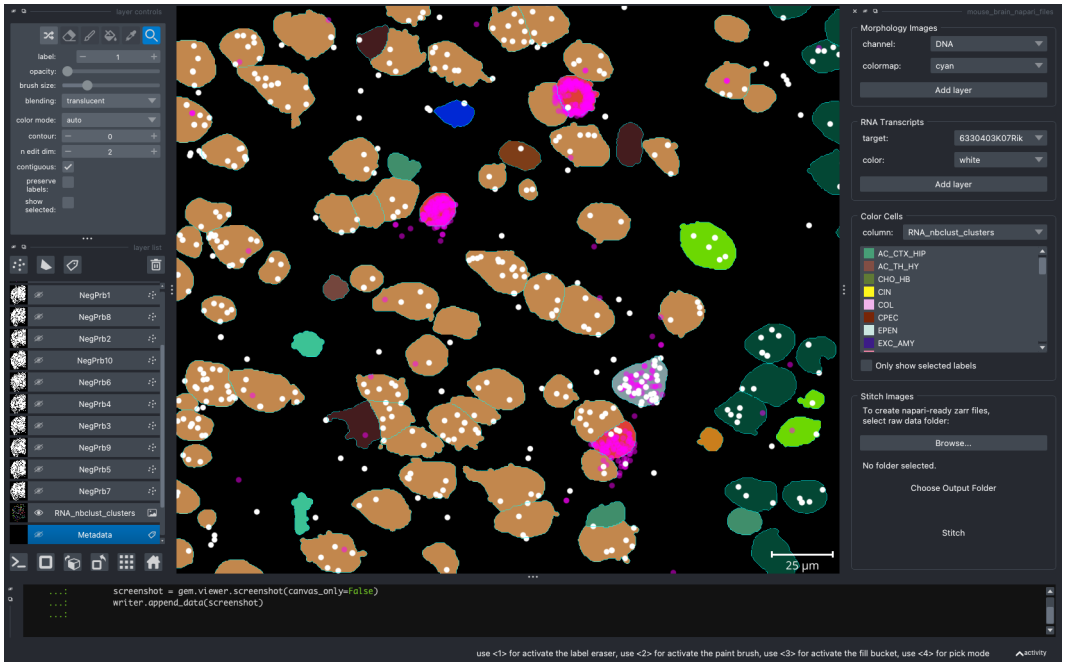

Figure 10: Same extent as Figure 9 but opacity of *Npy* reduced from 1 to 0.5.

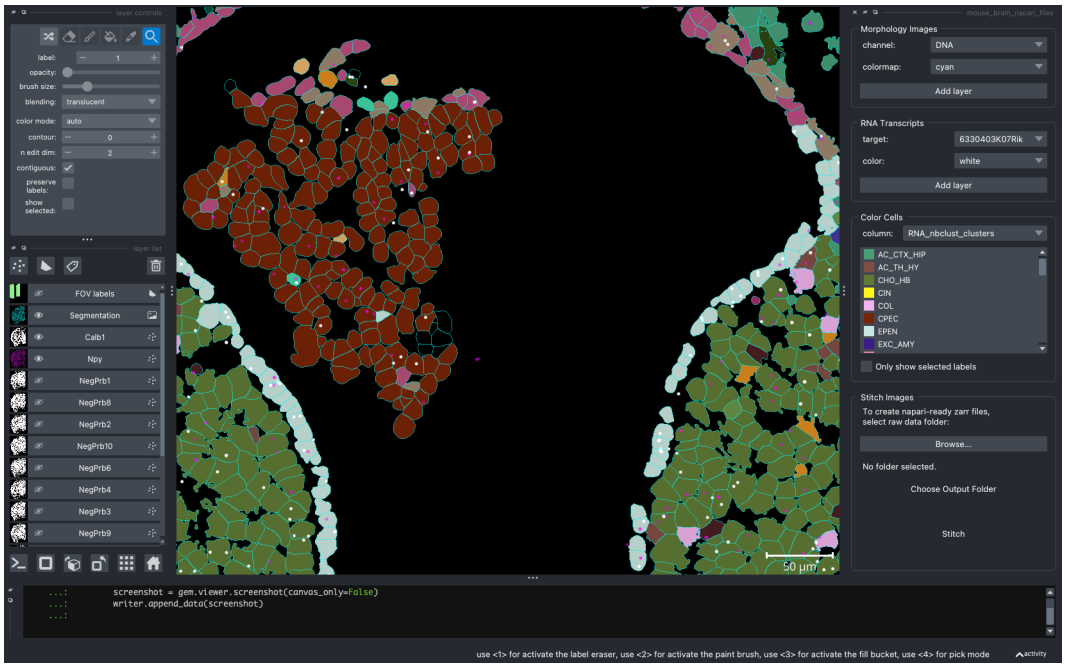

Figure 11: Centering to a particular FOV (123) using the `center_fov` method.

4.6 Plot all transcripts

This is not advised for resource-limited systems as it plots *all* transcripts. The method `add_points` plots all the points for a given FOV. If no FOV is specified, it will plot all transcripts (this can be taxing on resource-limited computers).

```
gem.add_points(fov=123)
gem.viewer.layers['Targets'].opacity = 0.4
fig_path = output_path + "/fig-tx-all.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=False)
        writer.append_data(screenshot)
```

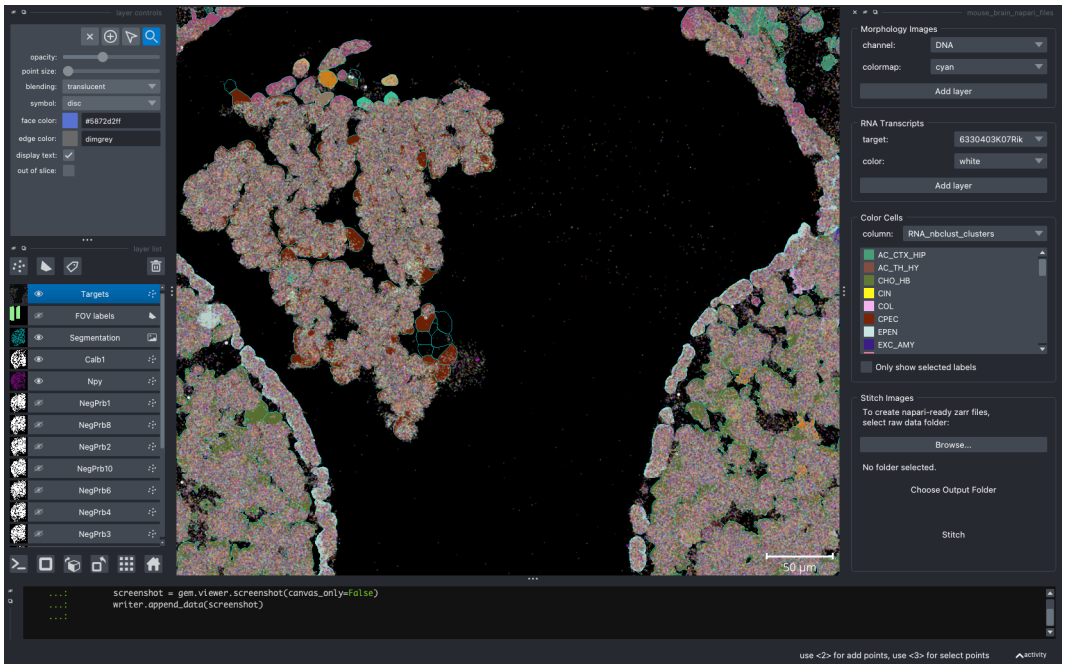

Figure 12: All targets for FOV 123.

4.7 Changing background color

For some publication styles (*e.g.*, posters), turning the background a lighter color might be useful. However, when changing the background, some items might be more difficult to see (compare Figure 7 with Figure 13).

```
gem.viewer.window.qt_viewer.canvas.background_color_override = 'white'
fig_path = output_path + "/fig-white.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=True)
        writer.append_data(screenshot)
```

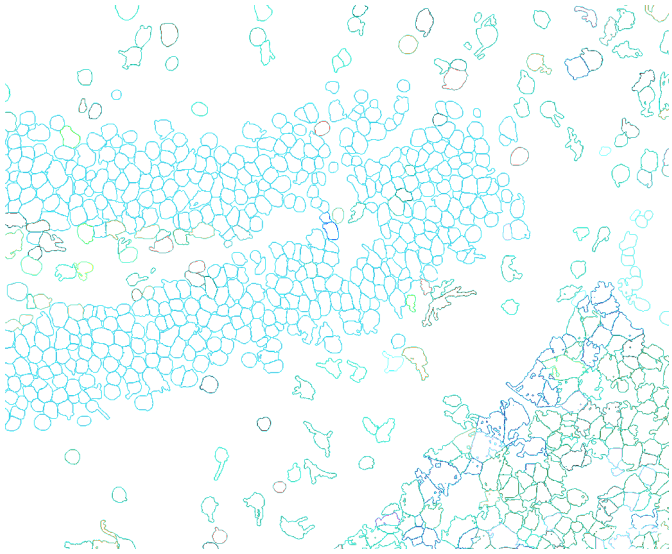

Figure 13: Same extent as Figure 7 but with a white background.

4.8 Scale Bar location

To reposition the scale bar to the bottom left:

```
gem.viewer.window.qt_viewer.canvas.background_color_override = 'black'
gem.viewer.scale_bar.position='bottom_left'
fig_path = output_path + "/fig-scale_bl.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=True)
        writer.append_data(screenshot)
```

4.9 Specify individual cell types

Here's my last tip for this post. Using the `color_cells` method, one can choose the color of the cell types and which cells to color by supplying a dictionary. If a cell type is *not* in the supplied dictionary, it will not be shown as a color.

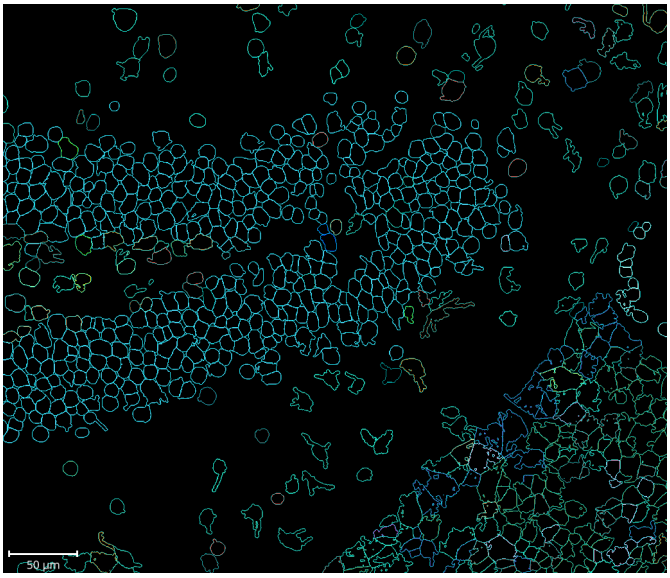

Figure 14: Same extent as Figure 7 but with a scale bar moved to the bottom left.

```
custom_colors = {
"MOL":"#AA0DFE",
"GN":"#85660D",
"CHO_HB":"orange" # need not be hexcode
}

gem.color_cells('RNA_nbclust_clusters', color=custom_colors)
fig_path = output_path + "/fig-color_three.png"
with imageio.get_writer(fig_path, dpi=(800, 800)) as writer:
        screenshot = gem.viewer.screenshot(canvas_only=True)
        writer.append_data(screenshot)
```

5 Conclusion

In this post I showed you some of my go-to `napari-cosmx` plugin features that I use when analyzing SMI data. In my workflow, I take advantage of the plugin's interactivity as well as its underlying functions and methods. This comes in the form of "jotting down" settings for reproducibility or fine-tuning an image's aesthetics ahead of publication. I couldn't cover all the things this plugin can do but look for other tips in future posts.

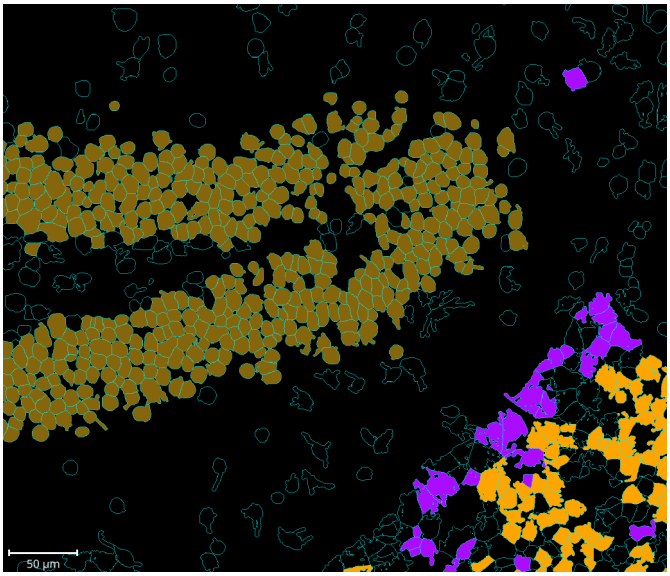

Figure 15: Same extent as Figure 7 highlighting three cell types only. MOL = mature oligodenrocytes = purple; GN = granule neurons = brown; CHO_HB = Cholinergic neurons Habenula = orange; cyan = all other cells.