[🍩] Lightweight thymus feature analysis using CODEX [CytoKit+SPRM]

By team **shlee** for the HuBMAP Hackathon 20241004

- Soo Hee Lee (https://github.com/sooheelee), individual contributor
- One of three winners

[0.1] Context and data sets

Problem and Goal Thymus Hassall's corpuscles (HCs) are thymic epithelial cells in the medulla region of the human thymus. H&E staining of HCs show a rosette pattern with few nuclei. Given cytokit cell segmentation relies on nuclei definitions, the few nuclei in HCs from 2D image data produces 'holes' in their spatial representation. The goal of this analysis is to identify HC protein markers and compare HC protein signal inside and outside of defined cells in CODEX thymus samples. The anaysis further compares the CODEX protein detection to that in a 2D IMC (Imaging Mass Cytometry) thymus sample and to a single-cell RNA-Seq thymus atlas dataset. A secondary goal is to show different ways of wrangling genomic image data.

Background HCs are known to produce cytokine TSLP (thymic stromal lymphopoietin), but their function remains an enigma. HCs vary in size with diameters from 20 to more than 100 µm and grow larger with age. They are abesent from mice, except for one strain in New Zealand, making them difficult to study. The number of HCs increases after puberty and then decreases progressively. They contain keratohyalin. HCs are implicated in diseases such as type 1 diabetes, rheumatoid arthritis, multiple sclerosis, and other autoimmune disease.

Overall Approach Use HuBMAP data portal interactive widgets to explore data and identify a tissue with large structures. Explore data types available for the tissue and study the available samples. Use desktop imaging tools Fiji to estimate size of structure and QuPath to manually create a mask truth set for one of the samples. Parameterize cellpose to segment structures across four CODEX sample images and compare signal from the mask versus cells within the mask. Finally, compare to thymus data sets from 2D Imaging Mass Cytometry (HuBMAP) and CZI single-cell RNA-Seq data (external). Data used in the analysis are as follows.

- 4 HuBMAP CODEX [CytoKit+SPRM] thymus samples
- 1 HuBMAP 2D IMC thymus sample
- (external) CZI Science cellxgene human thymus single-cell dataset

Implementation

- Tools used outside of notebook: QuPath, Fiji
- Python notebook packages: cv2, geopandas, matplotlib, numpy, pandas, scanpy, scipy, seabron, skimage, squidpy, tifffile

Biological Insights Thymus Hassall's corpuscles protein expression is represented by single-cell spatail data.

In [1]:
```
# linked datasets
# Select these thymus data sets from the HuBMAP Datasets page and launch a workspace.
# This will autopopulate the workspace with the data sets!
uuids = [
    "43213991a54ce196d406707ffe2e86bd",  # HBM373.LDGF.766 CODEX
    "822c9163d3be9b427dd0830f69a12305",  # HBM465.HZHH.676 CODEX
```

```
    "d4e9ec618924a8d43cfe1e67c38c1447",  # HBM887.SHVF.747 CODEX
    "37d06bb991afa2beb7b9460e746247ad",  # HBM893.CCKX.496 CODEX
    "9a6403bb0423e62950926a7d4fdab45b",  # HBM256.TVMB.842 2D IMC
]
```

In [2]:
```
# CODEX image data urls
ddir = "datasets/"

# Full resolution, non-pyramidal, multi-page, registered CODEX images, where pixel size == 0
# Registration between channels confirmed quickly by eye, using Fiji.
#
pixelsize = 0.3774
snames = {
    "sample1-a": "43213991a54ce196d406707ffe2e86bd",
    "sample1-b": "d4e9ec618924a8d43cfe1e67c38c1447",
    "sample2-a": "822c9163d3be9b427dd0830f69a12305",
    "sample2-b": "37d06bb991afa2beb7b9460e746247ad",
}

codexsampleurls = {}

for sn, id in snames.items():
    codexsampleurls[sn] = (
        f"{ddir}{id}/stitched/expressions/reg1_stitched_expressions.ome.tiff"
    )
```

In [3]:
```
# CODEX AnnData.zarr urls
codexanndataurls = {}
adatasuffix = "/anndata-zarr/reg1_stitched_expressions-anndata.zarr"

for sn, id in snames.items():
    codexanndataurls[sn] = f"{ddir}{id}{adatasuffix}"
```

[0.2] Python packages

In [ ]:
```
#! pip install cellpose
#! pip install squidpy
#! pip install gitpython
```

In [4]:
```
import anndata as ad
import ast
import cellpose
import cv2
import geopandas as gpd
import numpy as np
import pandas as pd
import scanpy as sc
import scipy
import seaborn as sns
import skimage as ski
import squidpy as sq
import tifffile
import warnings
import zarr

from cellpose import core, utils, io, models, metrics, plot
from collections import Counter
from git import Repo
from matplotlib import pyplot as plt
from shapely.geometry import Polygon
from tqdm import tqdm
```

In [5]:
```
#! pip freeze
```

```
absl-py==2.1.0
aiobotocore==2.5.4
aiohappyeyeballs==2.4.0
aiohttp==3.10.5
aioitertools==0.12.0
aiosignal==1.3.1
alabaster==0.7.16
anndata==0.9.1
annotated-types==0.7.0
anyio==4.4.0
anywidget==0.9.10
app-model==0.2.8
appdirs==1.4.4
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
array_api_compat==1.9
arrow==1.3.0
asciitree==0.3.3
asttokens==2.4.1
astunparse==1.6.3
async-lru==2.0.4
async-timeout==4.0.3
attrs==24.2.0
babel==2.16.0
beautifulsoup4==4.12.3
biopython==1.84
bleach==6.1.0
bokeh==3.4.3
botocore==1.31.17
Brotli @ file:///croot/brotli-split_1714483155106/work
build==1.2.2
cachetools==5.5.0
cachey==0.2.1
cellpose==3.0.11
cellxgene-census==1.15.0
certifi==2024.7.4
cffi==1.17.0
charset-normalizer @ file:///croot/charset-normalizer_1721748349566/work
chex==0.1.86
click==8.1.7
cloudpickle==3.0.0
colorcet==3.1.0
comm==0.2.2
contextlib2==21.6.0
contourpy==1.2.1
cycler==0.12.1
Cython @ file:///croot/cython_1712282251484/work
dask==2024.8.0
dask-expr==1.1.10
dask-image==2024.5.3
datashader==0.16.3
debugpy==1.8.5
decorator==5.1.1
defusedxml==0.7.1
distributed==2024.8.0
docrep==0.3.2
docstring_parser==0.16
docutils==0.21.2
etils==1.5.2
exceptiongroup==1.2.2
executing==2.0.1
fasteners==0.19
fastjsonschema==2.20.0
fastremap==1.15.0
filelock @ file:///croot/filelock_1700591183607/work
flatbuffers==24.3.25
```

```
flax==0.8.5
flexcache==0.3
flexparser==0.3.1
fonttools==4.53.1
fqdn==1.5.1
freetype-py==2.5.1
frozenlist==1.4.1
fsspec==2023.6.0
gast==0.4.0
geopandas==1.0.1
get-annotations==0.1.2
gitdb==4.0.11
GitPython==3.1.43
gmpy2 @ file:///tmp/build/80754af9/gmpy2_1645438755360/work
google-auth==2.34.0
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
grpcio==1.66.1
h11==0.14.0
h5py==3.11.0
HeapDict==1.0.1
hsluv==5.0.4
httpcore==1.0.5
httpx==0.27.2
humanize==4.10.0
idna @ file:///croot/idna_1714398848350/work
igraph==0.11.6
imagecodecs==2024.9.22
imageio==2.35.1
imagesize==1.4.1
importlib_metadata==8.4.0
importlib_resources==6.4.4
in-n-out==0.2.1
inflect==7.4.0
ipykernel==6.29.5
ipython==8.18.1
ipywidgets==8.1.1
isoduration==20.11.0
jax==0.4.30
jaxlib==0.4.30
jedi==0.19.1
Jinja2 @ file:///croot/jinja2_1716993405101/work
jmespath==1.0.1
joblib==1.4.2
json5==0.9.25
jsonpointer==3.0.0
jsonschema==4.23.0
jsonschema-specifications==2023.12.1
jupyter-events==0.10.0
jupyter-lsp==2.2.5
jupyter_client==8.6.2
jupyter_core==5.7.2
jupyter_server==2.14.2
jupyter_server_terminals==0.5.3
jupyterlab==4.2.5
jupyterlab_pygments==0.3.0
jupyterlab_server==2.27.3
jupyterlab_widgets==3.0.13
keras==2.11.0
kiwisolver==1.4.5
lazy_loader==0.4
legacy-api-wrap==1.4
leidenalg==0.10.2
libclang==18.1.1
lightning==2.4.0
lightning-utilities==0.11.7
```

```
llvmlite==0.41.1
locket==1.0.0
louvain==0.8.2
magicgui==0.9.1
Markdown==3.7
markdown-it-py==3.0.0
MarkupSafe @ file:///croot/markupsafe_1704205993651/work
matplotlib==3.8.4
matplotlib-inline==0.1.7
matplotlib-scalebar==0.8.1
mdurl==0.1.2
mistune==3.0.2
mkl-service==2.4.0
mkl_fft @ file:///croot/mkl_fft_1725370245198/work
mkl_random @ file:///croot/mkl_random_1725370241878/work
ml_collections==0.1.1
ml_dtypes==0.5.0
more-itertools==10.5.0
mpmath @ file:///croot/mpmath_1690848262763/work
msgpack==1.1.0
mudata==0.2.4
multidict==6.1.0
multipledispatch==1.0.0
multiscale_spatial_image==1.0.1
napari==0.5.3
napari-console==0.0.9
napari-plugin-engine==0.2.0
napari-plugin-manager==0.1.1
napari-svg==0.2.0
natsort==8.4.0
nbclient==0.10.0
nbconvert==7.16.4
nbformat==5.10.4
nest-asyncio==1.6.0
networkx @ file:///croot/networkx_1717597493534/work
notebook_shim==0.2.4
npe2==0.7.7
numba==0.58.0
numcodecs==0.12.1
numpy==1.25.0
numpydoc==1.8.0
numpyro==0.15.2
oauthlib==3.2.2
ome-zarr==0.9.0
omnipath==1.0.8
opencv-python-headless==4.10.0.84
opt-einsum==3.3.0
optax==0.2.3
orbax-checkpoint==0.6.3
overrides==7.7.0
packaging==24.1
pandas==2.2.2
pandocfilters==1.5.1
param==2.1.1
parso==0.8.4
partd==1.4.2
patsy==0.5.6
pexpect==4.9.0
pillow @ file:///croot/pillow_1721059439630/work
PIMS==0.7
Pint==0.24.3
platformdirs==4.2.2
pooch==1.8.2
prometheus_client==0.20.0
prompt_toolkit==3.0.47
protobuf==3.19.6
```

```
psutil==6.0.0
psygnal==0.11.1
ptyprocess==0.7.0
pure_eval==0.2.3
pyarrow==17.0.0
pyarrow-hotfix==0.6
pyasn1==0.6.1
pyasn1_modules==0.4.1
pyconify==0.1.6
pycparser==2.22
pyct==0.5.0
pydantic==2.9.1
pydantic-compat==0.1.2
pydantic_core==2.23.3
Pygments==2.18.0
pynndescent==0.5.13
pyogrio==0.9.0
PyOpenGL==3.1.7
pyparsing==3.1.4
pyproj==3.6.1
pyproject_hooks==1.1.0
PyQt5==5.15.11
PyQt5-Qt5==5.15.15
PyQt5_sip==12.15.0
pyro-api==0.1.2
pyro-ppl==1.9.1
PySocks @ file:///tmp/build/80754af9/pysocks_1605305812635/work
python-dateutil==2.9.0.post0
python-json-logger==2.0.7
pytorch-lightning==2.4.0
pytz==2024.2
PyWavelets==1.4.1
PyYAML @ file:///croot/pyyaml_1698096049011/work
pyzmq==26.2.0
qtconsole==5.6.0
QtPy==2.4.1
referencing==0.35.1
requests @ file:///croot/requests_1721410876868/work
requests-oauthlib==2.0.0
rfc3339-validator==0.1.4
rfc3986-validator==0.1.1
rich==13.8.1
roifile==2024.9.15
rpds-py==0.20.0
rsa==4.9
s3fs==2023.6.0
scanpy==1.9.8
scikit-image==0.21.0
scikit-learn==1.5.1
scipy==1.11.4
scvi-tools==1.1.6.post2
seaborn==0.13.2
Send2Trash==1.8.3
session-info==1.0.0
shapely==2.0.6
shellingham==1.5.4
six==1.16.0
slicerator==1.1.0
smmap==5.0.1
sniffio==1.3.1
snowballstemmer==2.2.0
somacore==1.0.11
sortedcontainers==2.4.0
soupsieve==2.6
spatial_image==1.1.0
spatialdata==0.2.2
```

```
Sphinx==7.4.7
sphinxcontrib-applehelp==2.0.0
sphinxcontrib-devhelp==2.0.0
sphinxcontrib-htmlhelp==2.1.0
sphinxcontrib-jsmath==1.0.1
sphinxcontrib-qthelp==2.0.0
sphinxcontrib-serializinghtml==2.0.0
squidpy==1.6.1
stack-data==0.6.3
statsmodels==0.14.1
stdlib-list==0.10.0
superqt==0.6.7
sympy @ file:///croot/sympy_1724938189289/work
tabulate==0.9.0
tblib==3.0.0
tensorboard==2.11.2
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.11.1
tensorflow-estimator==2.11.0
tensorflow-io-gcs-filesystem==0.37.1
tensorstore==0.1.65
termcolor==2.4.0
terminado==0.18.1
texttable==1.7.0
threadpoolctl==3.5.0
tifffile==2024.8.30
tiledb==0.29.1
tiledbsoma==1.11.4
tinycss2==1.3.0
tomli==2.0.1
tomli_w==1.0.0
toolz==0.12.1
torch==2.4.1
torchaudio==2.4.1
torchmetrics==1.4.2
torchvision==0.19.1
tornado==6.4.1
tqdm==4.66.5
traitlets==5.14.3
triangle==20230923
typeguard==4.3.0
typer==0.12.5
types-python-dateutil==2.9.0.20240821
typing_extensions @ file:///croot/typing_extensions_1715268824938/work
tzdata==2024.1
umap-learn==0.5.6
uri-template==1.3.0
urllib3==1.26.20
validators==0.34.0
vispy==0.14.3
wcwidth==0.2.13
webcolors==24.8.0
webencodings==0.5.1
websocket-client==1.8.0
Werkzeug==3.0.4
widgetsnbextension==4.0.13
wrapt==1.16.0
xarray==2024.7.0
xarray-dataclasses==1.8.0
xarray-datatree==0.0.14
xarray-schema==0.0.3
xarray-spatial==0.4.0
xyzservices==2024.9.0
yarl==1.13.1
zarr==2.18.2
```

```
zict==3.0.0
zipp==3.20.1
```

In [5]:
```
# Copy cleaned up files from shlee's github repo
git_url = "https://github.com/sooheelee/notebooks/"
repo_dir = "hubmap-hackathon-2024"

# Files of interest are in hubmap-hackathon-2024/shlee-hh-lightweight-thymus/
# This will error if the folder already exists
Repo.clone_from(git_url, repo_dir)

repofilepath = f"{repo_dir}/{repo_dir}/shlee-hh-lightweight-thymus/"
```

[1] Segment structures with cellpose

[1.1] Segment structures with cellpose: load model, data and truth polygons

In [7]:
```
# Print progress (cellpose stdout)
io.logger_setup()
```

```
2024-11-14 16:51:29,740 [INFO] WRITING LOG OUTPUT TO /hive/users/sleep/.cellpose/run.log
2024-11-14 16:51:29,741 [INFO]
cellpose version:       3.0.11
platform:               linux
python version:         3.9.19
torch version:          2.4.1
```

Out[7]:
```
(<Logger cellpose.io (INFO)>, PosixPath('/hive/users/sleep/.cellpose/run.log'))
```

In [8]:
```
# Cellpose enables use of GPUs; however, not currently for M1 architecture Macs, which I hav
# Thus, this notebook does not configure for GPU use.
use_GPU = core.use_gpu()
yn = ["NO", "YES"]
print(f">>> GPU activated? {yn[use_GPU]}")
```

```
2024-11-14 16:51:31,266 [INFO] Neither TORCH CUDA nor MPS version not installed/working.
>>> GPU activated? NO
```

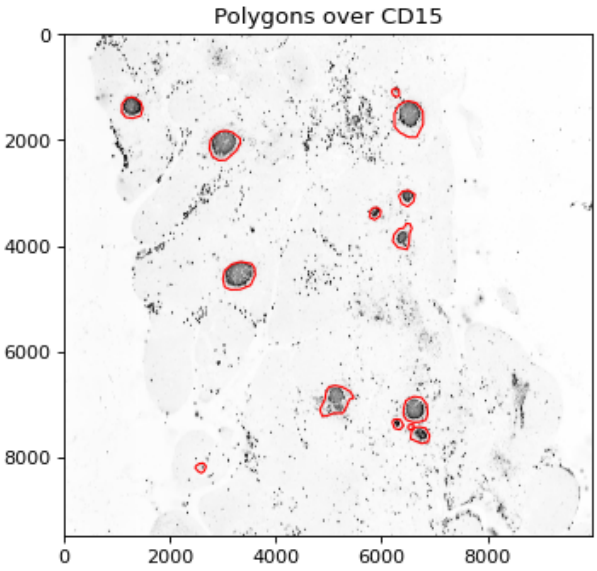
In [9]:
```
# Use cyto3 cellpose model.
# Cellpose offers functionality to train custom models, but the pretrained model works well
model = models.Cellpose(gpu=False, model_type="cyto3")
```

```
2024-11-14 16:51:31,576 [INFO] >>>> using CPU
2024-11-14 16:51:31,576 [INFO] >>>> using CPU
2024-11-14 16:51:31,577 [INFO] >> cyto3 << model set to be used
2024-11-14 16:51:31,637 [INFO] >>>> loading model /hive/users/sleep/.cellpose/models/cyto3
2024-11-14 16:51:31,715 [INFO] >>>> model diam_mean =  30.000 (ROIs rescaled to this size dur
ing training)
/hive/user-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-pac
kages/cellpose/resnet_torch.py:280: FutureWarning: You are using `torch.load` with `weights_o
nly=False` (the current default value), which uses the default pickle module implicitly. It i
s possible to construct malicious pickle data which will execute arbitrary code during unpick
ling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more
details). In a future release, the default value for `weights_only` will be flipped to `True
`. This limits the functions that could be executed during unpickling. Arbitrary objects will
no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the
user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only
=True` for any use case where you don't have full control of the loaded file. Please open an
issue on GitHub for any issues related to this experimental feature.
  state_dict = torch.load(filename, map_location=torch.device("cpu"))
```

In [10]:
```
# Created truth set polygons of HC regions manually, in QuPath, for sample1-a. QuPath output
codgjurl = f"{repofilepath}sample1-a-3ch.geojson"
cdf = pd.read_json(codgjurl)
```

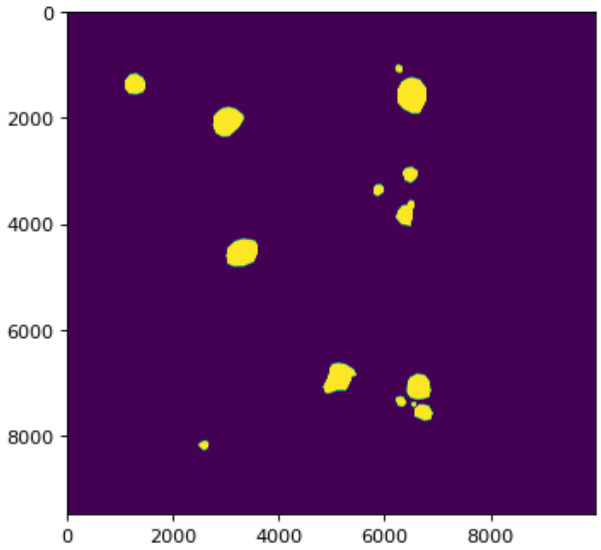
```
codgj = gpd.read_file(codgjurl)
codgj["scalefactor"] = pixelsize
tester = "sample1-a"
```

In [11]:
```
# Visualize truth set polygons
plt.rcParams["figure.figsize"] = (5, 5)
plt.rcParams["figure.dpi"] = 80

with tifffile.TiffFile(codexsampleurls[tester]) as ctif:
    cpage = ctif.pages[2]
    cimg = cpage.asarray()

    fig, ax = plt.subplots()
    plt.imshow(cimg, cmap="binary", vmax=np.max(cimg) / 1)
    codgj.plot(ax=ax, facecolor="none", edgecolor="red", aspect=1)
    plt.title("Polygons over CD15", size=12)
    plt.axis("scaled")
    plt.show()
```



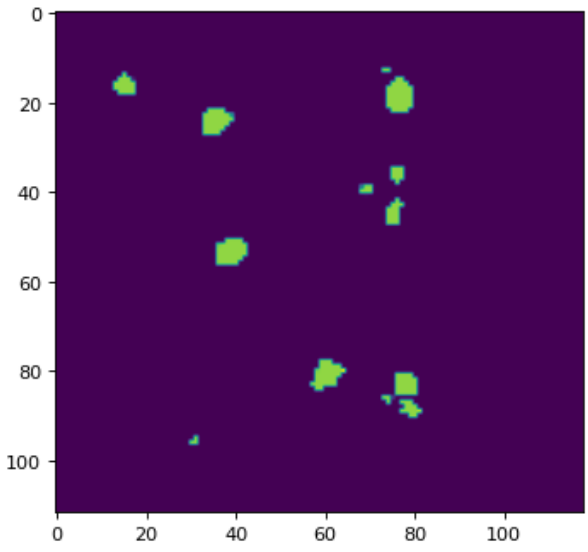
In [12]:
```
# create a mask of the polygons
mm = np.zeros(cimg.shape)

for i, row in cdf.iterrows():
    p = np.array(row.features["geometry"]["coordinates"]).squeeze()
    r, c = ski.draw.polygon(p[:, 1], p[:, 0], cimg.shape)
    mm[r, c] = 1

plt.imshow(mm)
```

Out[12]:
```
<matplotlib.image.AxesImage at 0x149799e717f0>
```



In [13]:
```
# Create downsampled versions of the mask; factor 0 gives 1 µm pixel size
factors = [0, 1, 2, 3, 4, 5]
dmasks = {}

for factor in factors:
    dm = scipy.ndimage.zoom(mm, pixelsize / 2**factor)
    dmasks[factor] = dm

plt.imshow(dm)
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x1497990a40d0>
```



[1.2] Segment structures with cellpose: parameterize with one sample

In [14]:
```
# Create subsampled images for use with cellpose; factor 0 gives 1 µm pixel size
factors = [0, 1, 2, 3, 4, 5]
images = {}

with tifffile.TiffFile(codexsampleurls[tester]) as ctif:
    cpage = ctif.pages[2].asarray()
    print(cpage.shape)

    for factor in factors:
        img = scipy.ndimage.zoom(cpage, pixelsize / 2**factor)
```

```
        images[factor] = img
        print(factor, img.shape)
```

```
(9492, 9973)
0 (3582, 3764)
1 (1791, 1882)
2 (896, 941)
3 (448, 470)
4 (224, 235)
5 (112, 118)
```

In [15]:
```
# Cellpose can take single or multipage fluorescent images, e.g. nucleus + cytosolic stains.
# Using single-image mode.
channels = np.array([[0, 0]])

# Cellpose default diameter size is 30 pixels. Test various diameters (given in microns).
diameters = [64, 96, 128, 160, 192]
```

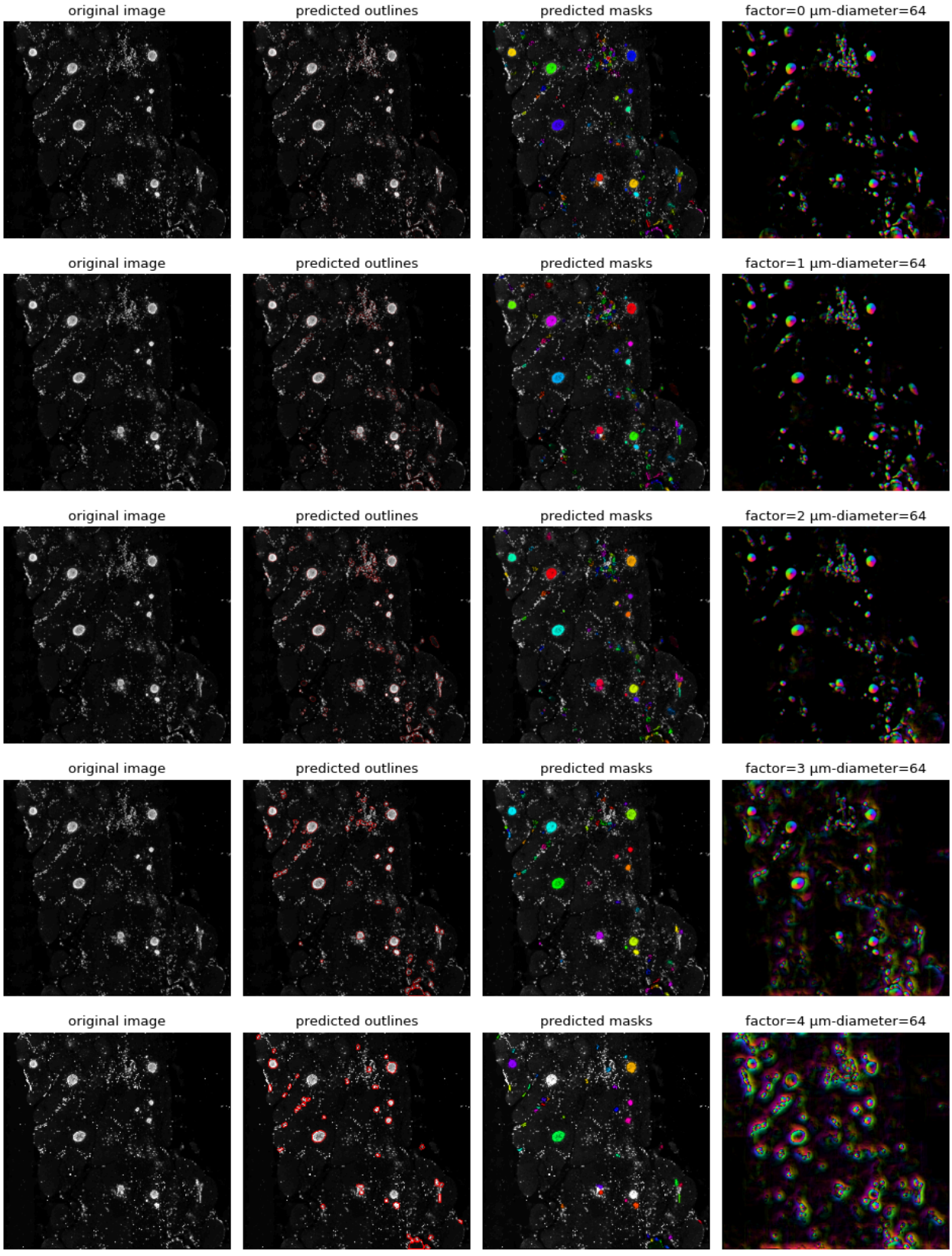
In [16]:
```
# Takes ~2 minutes per diameter or <~10 minutes in total
results = {}

for diameter in diameters:
    results[diameter] = {}

    for factor, img in images.items():
        # Rescale micron diameter to match image pixels
        diam = diameter / (2**factor)
        print(factor, diam)

        # Run cellpose
        masks, flows, styles, diams = model.eval(img, diameter=diam, channels=channels)
        results[diameter][factor] = [masks, flows, styles, diams]
```

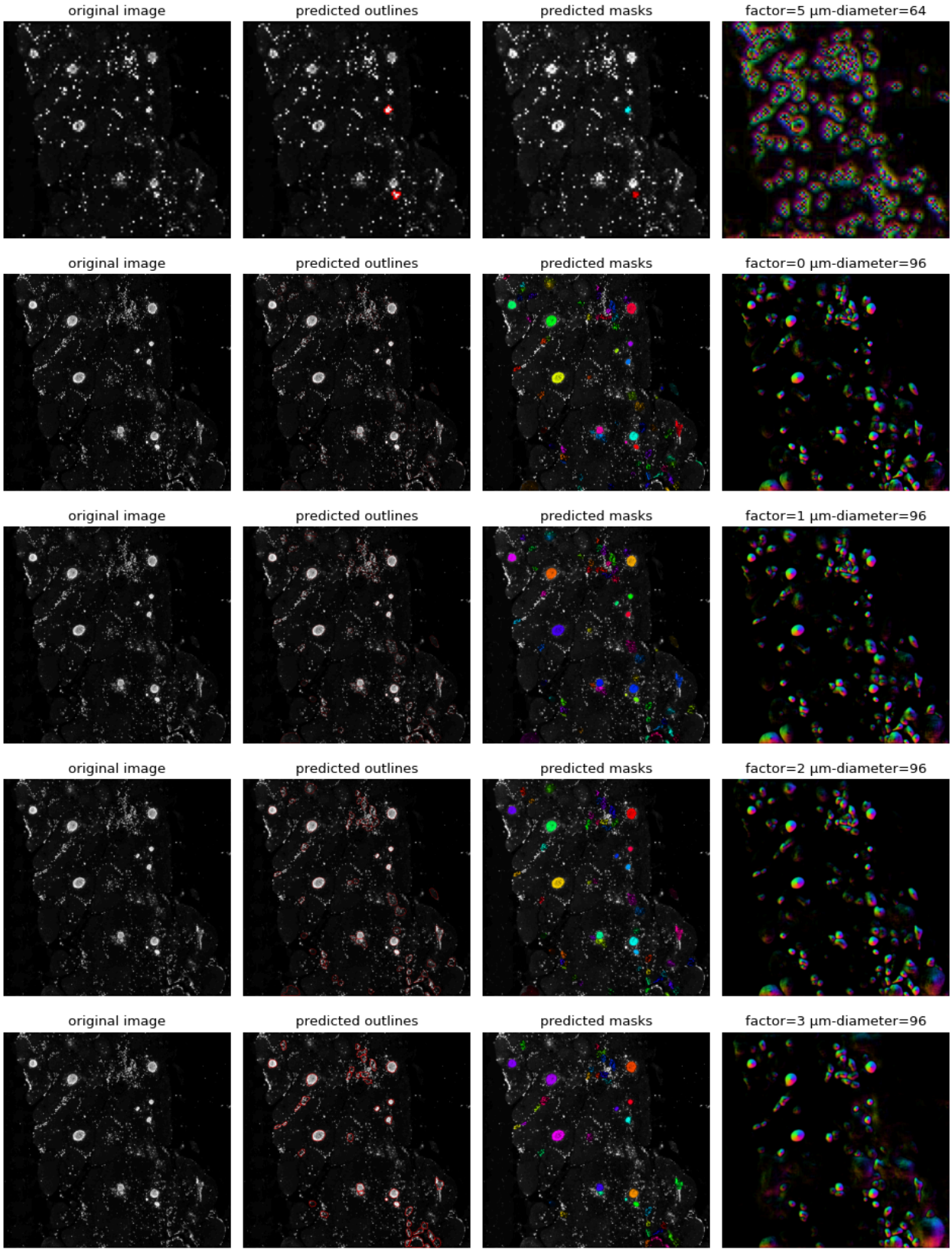
```
0 64.0
2024-11-14 16:06:42,120 [INFO] channels set to [[0 0]]
2024-11-14 16:06:42,121 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:07:12,432 [INFO] >>>> TOTAL TIME 30.31 sec
1 32.0
2024-11-14 16:07:12,434 [INFO] channels set to [[0 0]]
2024-11-14 16:07:12,434 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:07:25,228 [INFO] >>>> TOTAL TIME 12.79 sec
2 16.0
2024-11-14 16:07:25,230 [INFO] channels set to [[0 0]]
2024-11-14 16:07:25,230 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:07:36,539 [INFO] >>>> TOTAL TIME 11.31 sec
3 8.0
2024-11-14 16:07:36,540 [INFO] channels set to [[0 0]]
2024-11-14 16:07:36,540 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:07:47,892 [INFO] >>>> TOTAL TIME 11.35 sec
4 4.0
2024-11-14 16:07:47,894 [INFO] channels set to [[0 0]]
2024-11-14 16:07:47,894 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:07:58,808 [INFO] >>>> TOTAL TIME 10.91 sec
5 2.0
2024-11-14 16:07:58,809 [INFO] channels set to [[0 0]]
2024-11-14 16:07:58,810 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:08:09,623 [INFO] >>>> TOTAL TIME 10.81 sec
0 96.0
2024-11-14 16:08:09,624 [INFO] channels set to [[0 0]]
2024-11-14 16:08:09,624 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:08:43,198 [INFO] >>>> TOTAL TIME 33.57 sec
1 48.0
2024-11-14 16:08:43,200 [INFO] channels set to [[0 0]]
2024-11-14 16:08:43,200 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:08:51,860 [INFO] >>>> TOTAL TIME 8.66 sec
2 24.0
2024-11-14 16:08:51,861 [INFO] channels set to [[0 0]]
2024-11-14 16:08:51,862 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:08:57,776 [INFO] >>>> TOTAL TIME 5.91 sec
3 12.0
2024-11-14 16:08:57,777 [INFO] channels set to [[0 0]]
2024-11-14 16:08:57,777 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:09:03,336 [INFO] >>>> TOTAL TIME 5.56 sec
4 6.0
2024-11-14 16:09:03,337 [INFO] channels set to [[0 0]]
2024-11-14 16:09:03,337 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:09:08,892 [INFO] >>>> TOTAL TIME 5.55 sec
5 3.0
2024-11-14 16:09:08,893 [INFO] channels set to [[0 0]]
2024-11-14 16:09:08,893 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:09:14,303 [INFO] >>>> TOTAL TIME 5.41 sec
0 128.0
2024-11-14 16:09:14,304 [INFO] channels set to [[0 0]]
2024-11-14 16:09:14,304 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:09:54,232 [INFO] >>>> TOTAL TIME 39.93 sec
1 64.0
2024-11-14 16:09:54,234 [INFO] channels set to [[0 0]]
2024-11-14 16:09:54,234 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:10:01,032 [INFO] >>>> TOTAL TIME 6.80 sec
2 32.0
2024-11-14 16:10:01,033 [INFO] channels set to [[0 0]]
2024-11-14 16:10:01,033 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:10:04,253 [INFO] >>>> TOTAL TIME 3.22 sec
3 16.0
2024-11-14 16:10:04,254 [INFO] channels set to [[0 0]]
2024-11-14 16:10:04,254 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:10:07,143 [INFO] >>>> TOTAL TIME 2.89 sec
4 8.0
2024-11-14 16:10:07,145 [INFO] channels set to [[0 0]]
```

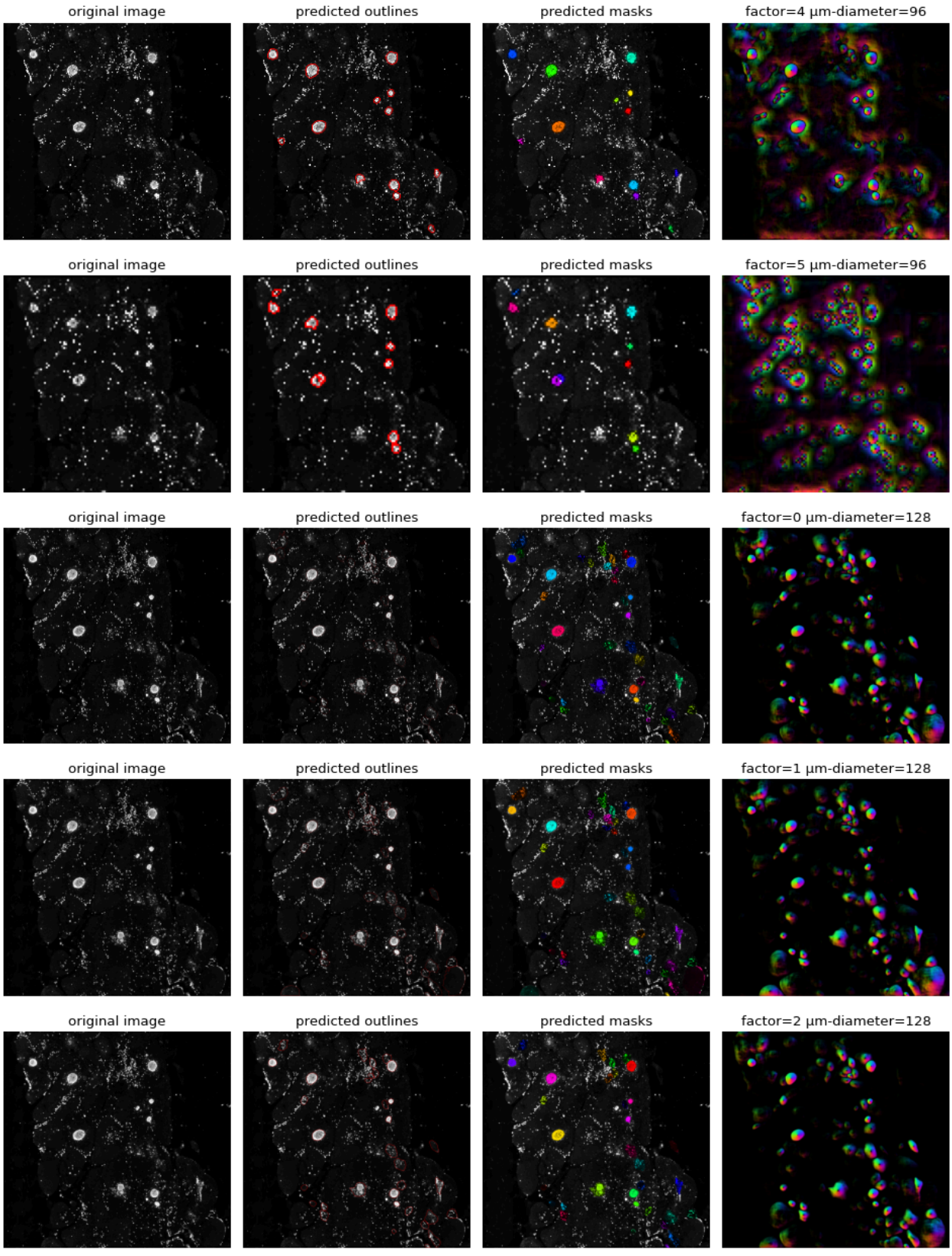
```
2024-11-14 16:10:07,145 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:10:09,930 [INFO] >>>> TOTAL TIME 2.79 sec
5 4.0
2024-11-14 16:10:09,931 [INFO] channels set to [[0 0]]
2024-11-14 16:10:09,931 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:10:12,697 [INFO] >>>> TOTAL TIME 2.77 sec
0 160.0
2024-11-14 16:10:12,698 [INFO] channels set to [[0 0]]
2024-11-14 16:10:12,698 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:10:55,901 [INFO] >>>> TOTAL TIME 43.20 sec
1 80.0
2024-11-14 16:10:55,903 [INFO] channels set to [[0 0]]
2024-11-14 16:10:55,903 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:11:01,731 [INFO] >>>> TOTAL TIME 5.83 sec
2 40.0
2024-11-14 16:11:01,732 [INFO] channels set to [[0 0]]
2024-11-14 16:11:01,733 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:11:04,000 [INFO] >>>> TOTAL TIME 2.27 sec
3 20.0
2024-11-14 16:11:04,001 [INFO] channels set to [[0 0]]
2024-11-14 16:11:04,001 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:11:05,845 [INFO] >>>> TOTAL TIME 1.84 sec
4 10.0
2024-11-14 16:11:05,846 [INFO] channels set to [[0 0]]
2024-11-14 16:11:05,847 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:11:07,626 [INFO] >>>> TOTAL TIME 1.78 sec
5 5.0
2024-11-14 16:11:07,627 [INFO] channels set to [[0 0]]
2024-11-14 16:11:07,627 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:11:09,785 [INFO] >>>> TOTAL TIME 2.16 sec
0 192.0
2024-11-14 16:11:09,785 [INFO] channels set to [[0 0]]
2024-11-14 16:11:09,786 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:12:00,673 [INFO] >>>> TOTAL TIME 50.89 sec
1 96.0
2024-11-14 16:12:00,675 [INFO] channels set to [[0 0]]
2024-11-14 16:12:00,675 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:12:06,868 [INFO] >>>> TOTAL TIME 6.19 sec
2 48.0
2024-11-14 16:12:06,869 [INFO] channels set to [[0 0]]
2024-11-14 16:12:06,869 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:12:09,207 [INFO] >>>> TOTAL TIME 2.34 sec
3 24.0
2024-11-14 16:12:09,208 [INFO] channels set to [[0 0]]
2024-11-14 16:12:09,208 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:12:11,073 [INFO] >>>> TOTAL TIME 1.87 sec
4 12.0
2024-11-14 16:12:11,074 [INFO] channels set to [[0 0]]
2024-11-14 16:12:11,074 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:12:12,871 [INFO] >>>> TOTAL TIME 1.80 sec
5 6.0
2024-11-14 16:12:12,872 [INFO] channels set to [[0 0]]
2024-11-14 16:12:12,873 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:12:14,689 [INFO] >>>> TOTAL TIME 1.82 sec
```

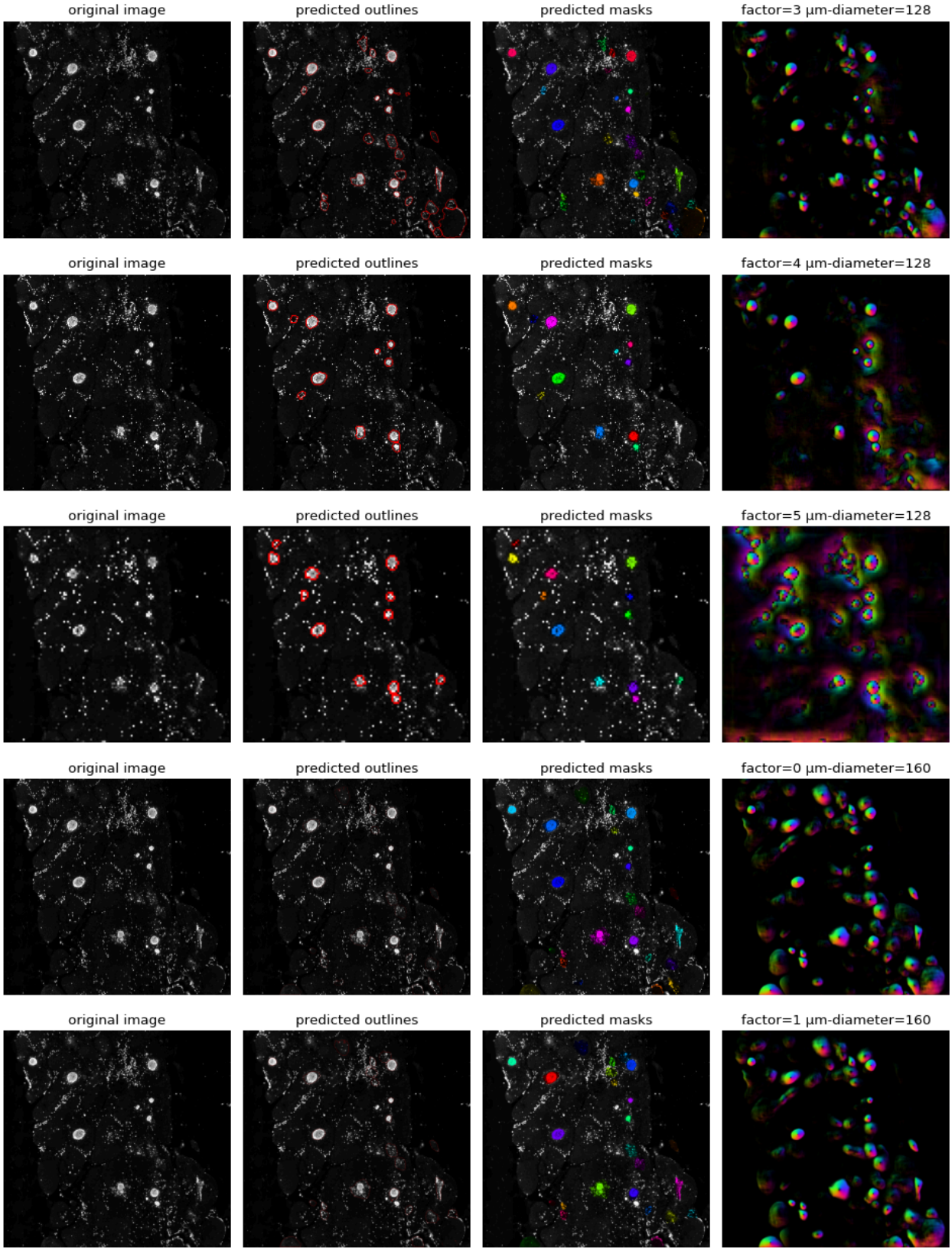
In [17]:

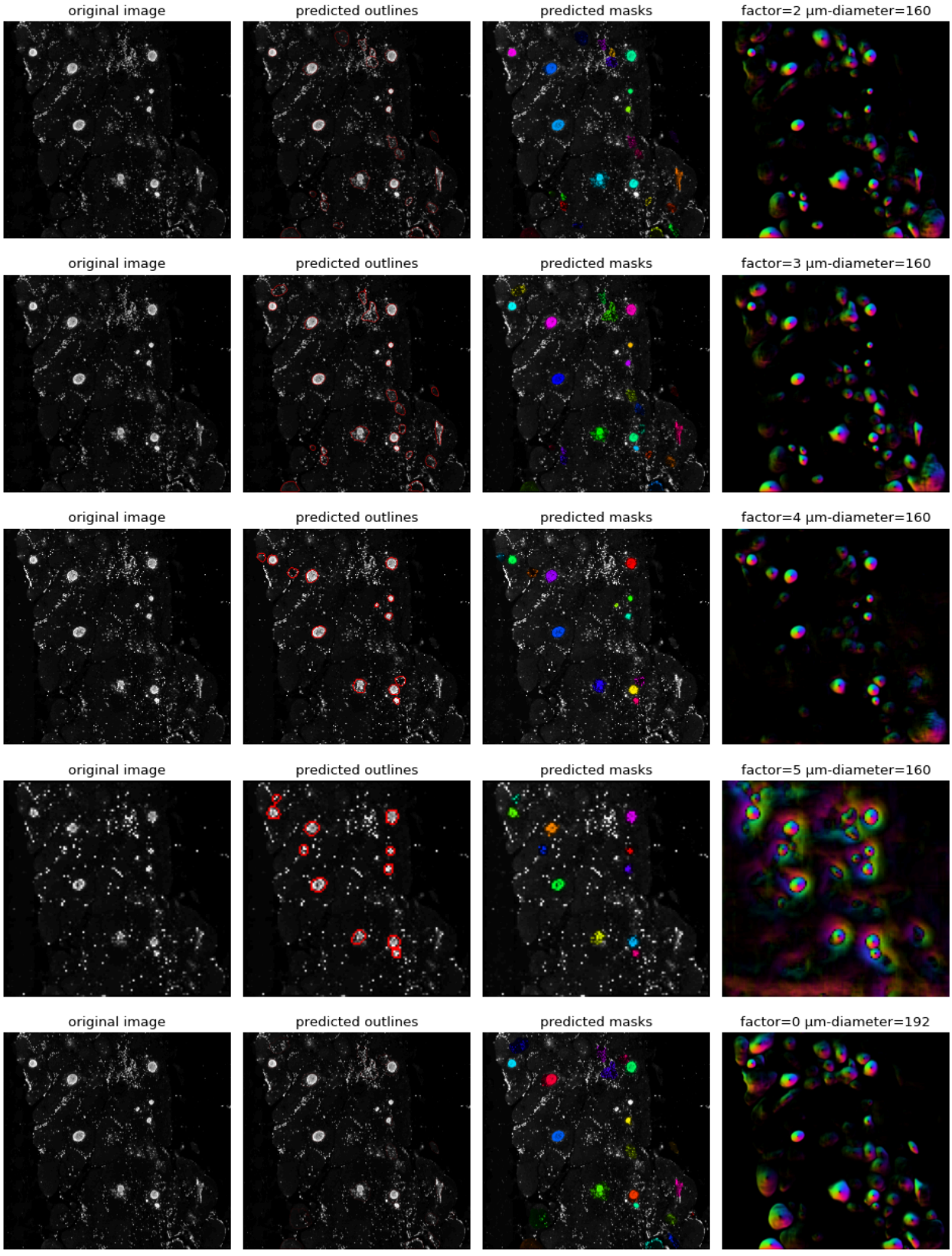
```python
# Plot run results; 4th column lists parameters
for diameter, result in results.items():
    for factor, r in result.items():
        masks, flows, styles, diams = r
        fig = plt.figure(figsize=(12, 5))
        plot.show_segmentation(
            fig, images[factor], masks, flows[0], channels=channels[0]
        )
        plt.tight_layout()
        plt.title(f"factor={factor} µm-diameter={diameter}")
        plt.show()
```

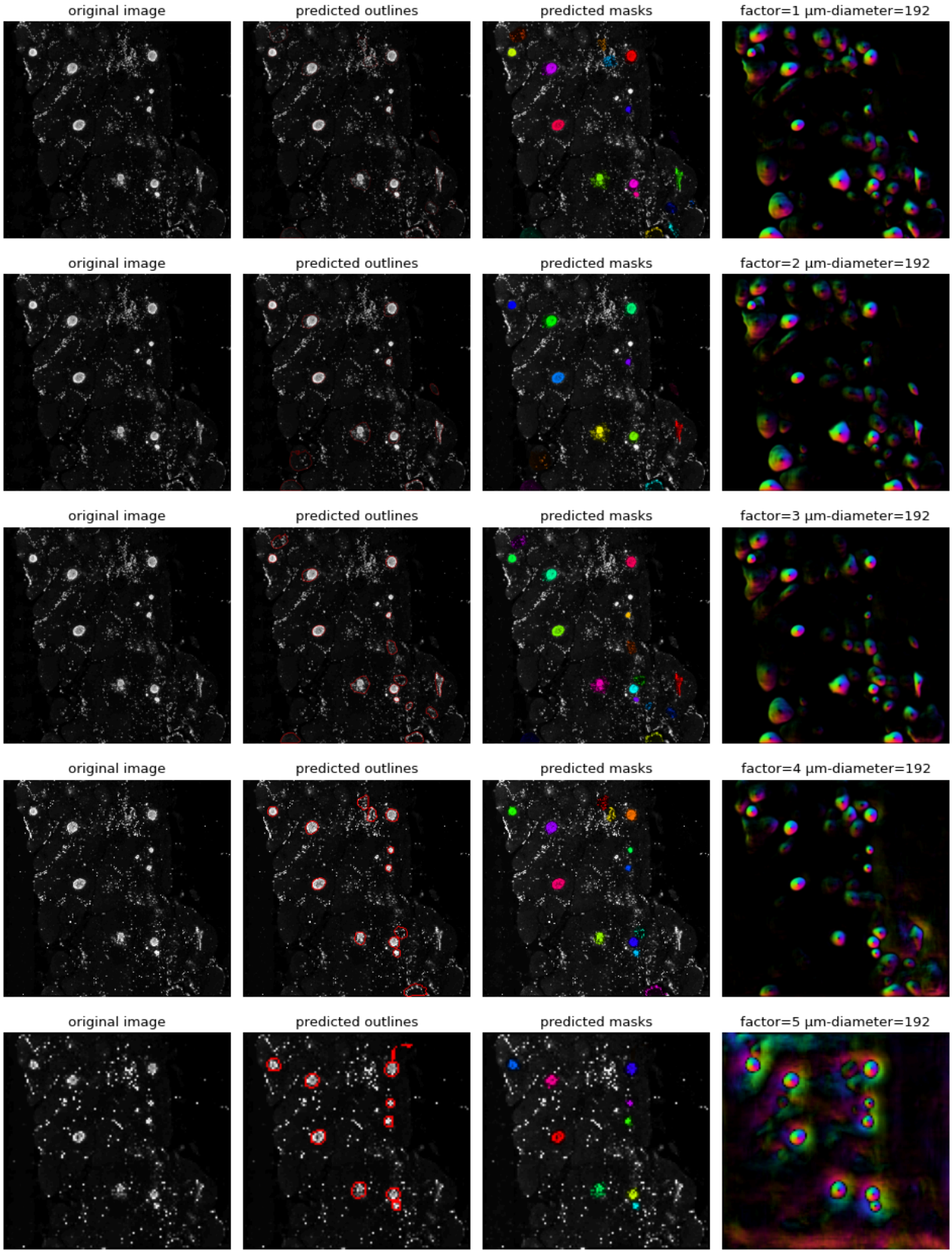

original image
predicted outlines
predicted masks
factor=0 µm-diameter=64
original image
predicted outlines
predicted masks
factor=1 µm-diameter=64
original image
predicted outlines
predicted masks
factor=2 µm-diameter=64
original image
predicted outlines
predicted masks
factor=3 µm-diameter=64
original image
predicted outlines
predicted masks
factor=4 µm-diameter=64


original image
predicted outlines
predicted masks
factor=5 µm-diameter=64
original image
predicted outlines
predicted masks
factor=0 µm-diameter=96
original image
predicted outlines
predicted masks
factor=1 µm-diameter=96
original image
predicted outlines
predicted masks
factor=2 µm-diameter=96
original image
predicted outlines
predicted masks
factor=3 µm-diameter=96


original image
predicted outlines
predicted masks
factor=4 µm-diameter=96
original image
predicted outlines
predicted masks
factor=5 µm-diameter=96
original image
predicted outlines
predicted masks
factor=0 µm-diameter=128
original image
predicted outlines
predicted masks
factor=1 µm-diameter=128
original image
predicted outlines
predicted masks
factor=2 µm-diameter=128


original image
predicted outlines
predicted masks
factor=3 µm-diameter=128
original image
predicted outlines
predicted masks
factor=4 µm-diameter=128
original image
predicted outlines
predicted masks
factor=5 µm-diameter=128
original image
predicted outlines
predicted masks
factor=0 µm-diameter=160
original image
predicted outlines
predicted masks
factor=1 µm-diameter=160

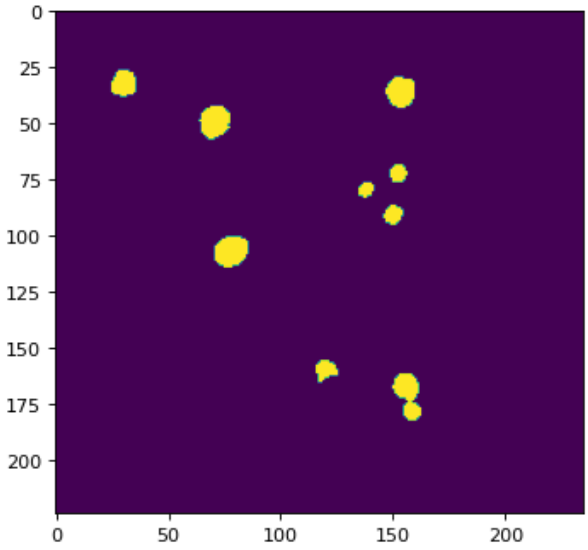

original image
predicted outlines
predicted masks
factor=2 µm-diameter=160
original image
predicted outlines
predicted masks
factor=3 µm-diameter=160
original image
predicted outlines
predicted masks
factor=4 µm-diameter=160
original image
predicted outlines
predicted masks
factor=5 µm-diameter=160
original image
predicted outlines
predicted masks
factor=0 µm-diameter=192



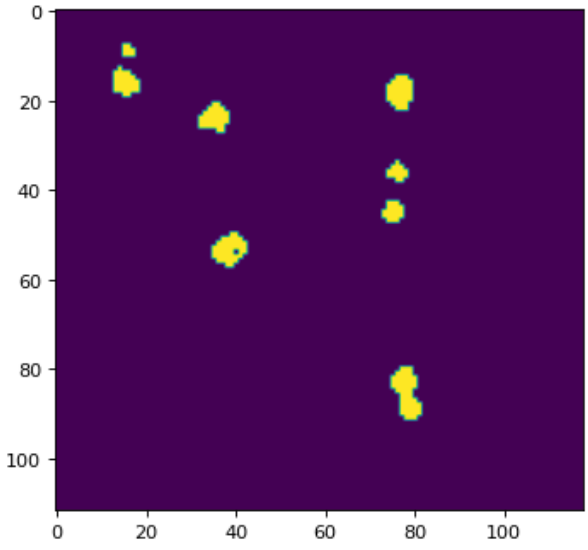
In [18]:
```
# Fraction overlap with truth set polygons and number of polygons
for diameter, result in results.items():
    for factor, r in result.items():
        masks, flows, styles, diams = r

        truthmask = dmasks[factor]
        intersection_mask = np.logical_and(masks, truthmask)
```

```
        # Fraction overlap
        score = round(np.sum(intersection_mask) / np.sum(truthmask), 3)

        # Number of polygons
        nummasks = len(Counter(masks.flatten())) - 1

        print(f"factor={factor} µm-diameter={diameter} score={score} num={nummasks}")

        # Plot good segmentations
        if (score < 1.1) and (score > 0.9) and (nummasks <= 16) and (nummasks >= 10):
            plt.imshow(intersection_mask)
            plt.show()
```

```
factor=0 µm-diameter=64 score=0.768 num=128
factor=1 µm-diameter=64 score=0.81 num=120
factor=2 µm-diameter=64 score=0.806 num=99
factor=3 µm-diameter=64 score=0.867 num=59
factor=4 µm-diameter=64 score=0.665 num=42
factor=5 µm-diameter=64 score=0.13 num=2
factor=0 µm-diameter=96 score=0.955 num=77
factor=1 µm-diameter=96 score=0.88 num=60
factor=2 µm-diameter=96 score=0.838 num=57
factor=3 µm-diameter=96 score=0.839 num=41
factor=4 µm-diameter=96 score=1.014 num=13
```




```
factor=5 µm-diameter=96 score=1.027 num=10
```

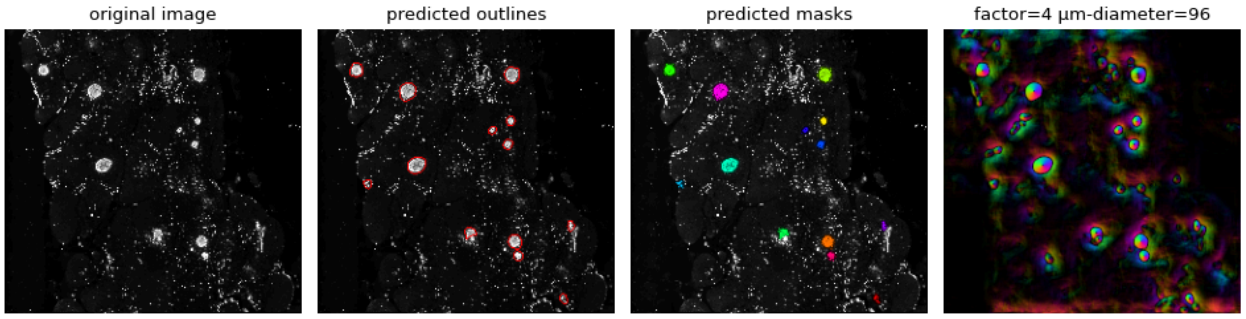


```
factor=0 µm-diameter=128 score=1.166 num=45
factor=1 µm-diameter=128 score=1.089 num=41
factor=2 µm-diameter=128 score=1.004 num=34
factor=3 µm-diameter=128 score=0.968 num=31
factor=4 µm-diameter=128 score=1.1 num=12
factor=5 µm-diameter=128 score=1.275 num=12
factor=0 µm-diameter=160 score=0.99 num=30
factor=1 µm-diameter=160 score=1.119 num=27
factor=2 µm-diameter=160 score=0.97 num=25
factor=3 µm-diameter=160 score=1.046 num=24
factor=4 µm-diameter=160 score=1.194 num=13
factor=5 µm-diameter=160 score=1.544 num=11
factor=0 µm-diameter=192 score=1.665 num=25
factor=1 µm-diameter=192 score=1.192 num=20
factor=2 µm-diameter=192 score=1.455 num=12
factor=3 µm-diameter=192 score=1.107 num=16
factor=4 µm-diameter=192 score=1.176 num=13
factor=5 µm-diameter=192 score=1.658 num=10
```

We have a winner: `factor=4 µm-diameter=96 score=1.014, num=13`

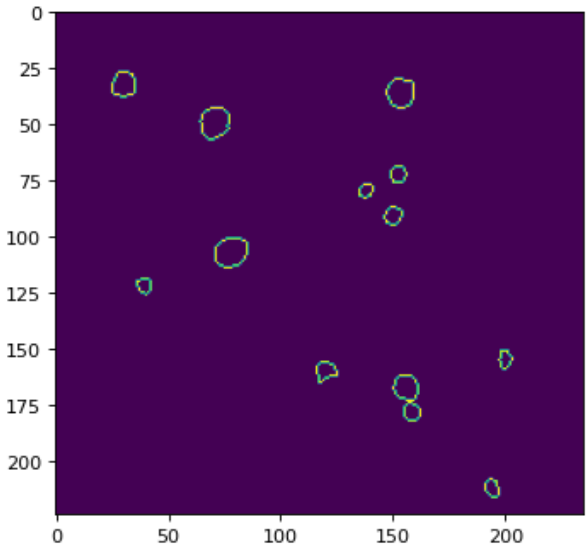
In [19]:
```python
# Best parameters and result
factor = 4
diameter = 96
masks, flows, styles, diams = results[diameter][factor]
```

In [20]:
```python
# Visualize winning result
fig = plt.figure(figsize=(12, 5))
plot.show_segmentation(fig, images[factor], masks, flows[0], channels=channels[0])
plt.tight_layout()
plt.title(f"factor={factor} µm-diameter={diameter}")
plt.show()
```



In [21]:
```python
# Visualize the outlines alone
plt.imshow(utils.masks_to_outlines(masks))
```

Out[21]:
```
<matplotlib.image.AxesImage at 0x1496d29a0a00>
```



[1.3] Segment structures with cellpose: apply best parameters to all samples

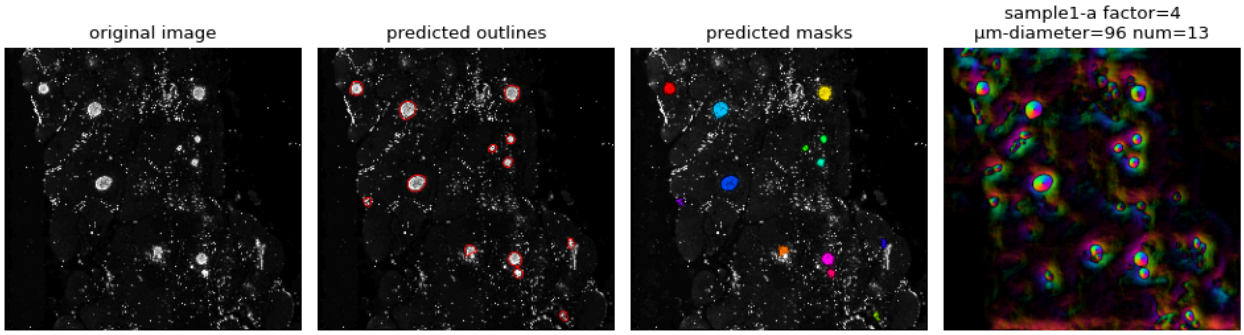
In [22]:
```
# Extract and downsample CD15 channel image, and save the original image dimensions
diameter = 96
images = {}
originalshapes = {}

for sample, sampleurl in codexsampleurls.items():
    with tifffile.TiffFile(sampleurl) as ctif:
        ct = ctif.pages[2].asarray()
        originalshapes[sample] = ct.shape

        img = scipy.ndimage.zoom(ct, pixelsize / 2**factor)
        images[sample] = img
        print(sample, ct.shape, img.shape)
```

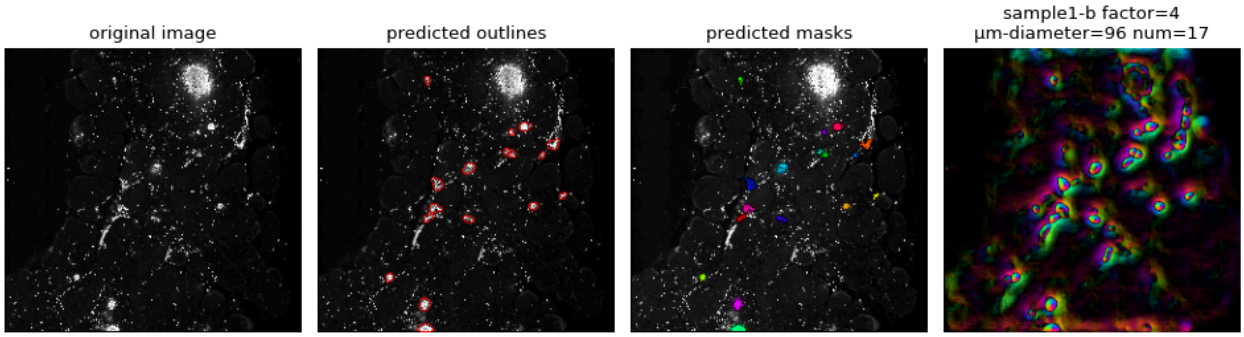
```
sample1-a (9492, 9973) (224, 235)
sample1-b (9486, 9971) (224, 235)
sample2-a (9485, 10000) (224, 236)
sample2-b (9493, 9973) (224, 235)
```

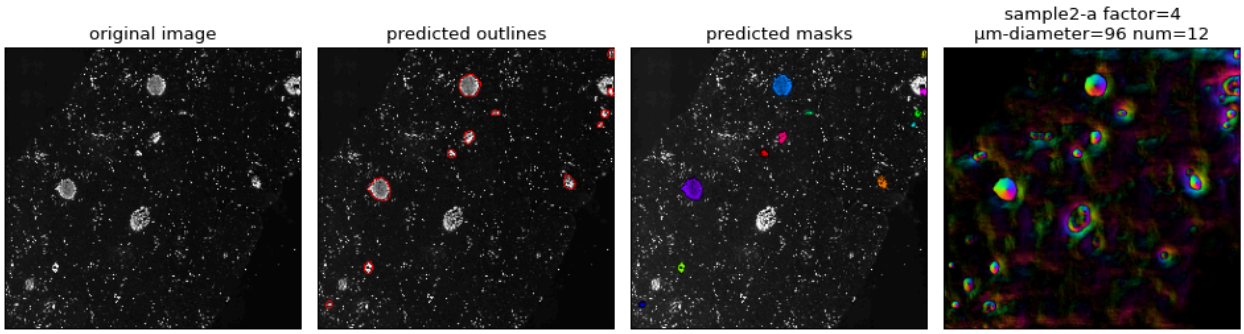
In [23]:
```
# Run cellpose on each sample with winning parameters
sresults = {}
diam = diameter / (2**factor)
print(factor, diam)

for sample, img in images.items():
    print(sample)
    masks, flows, styles, diams = model.eval(img, diameter=diam, channels=channels)
    sresults[sample] = [masks, flows, styles, diams]
```

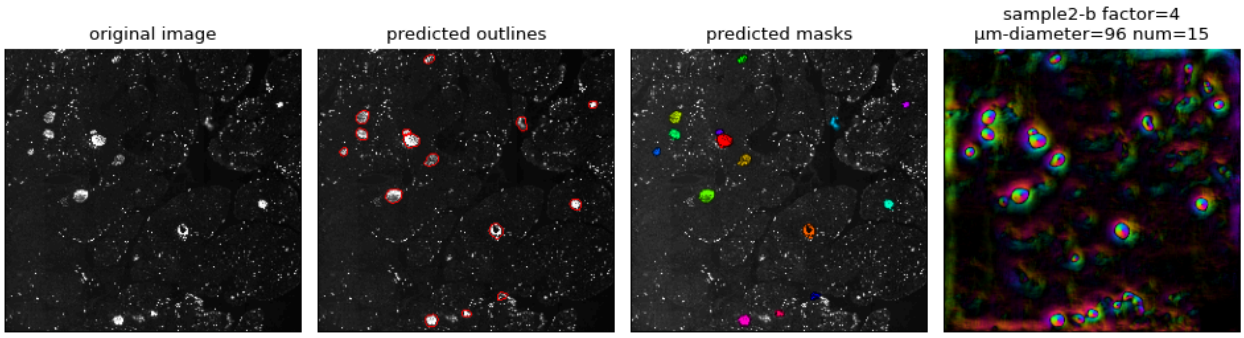
```
4 6.0
sample1-a
2024-11-14 16:13:33,253 [INFO] channels set to [[0 0]]
2024-11-14 16:13:33,254 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:13:38,748 [INFO] >>>> TOTAL TIME 5.49 sec
sample1-b
2024-11-14 16:13:38,749 [INFO] channels set to [[0 0]]
2024-11-14 16:13:38,749 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:13:44,236 [INFO] >>>> TOTAL TIME 5.49 sec
sample2-a
2024-11-14 16:13:44,237 [INFO] channels set to [[0 0]]
2024-11-14 16:13:44,237 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:13:49,707 [INFO] >>>> TOTAL TIME 5.47 sec
sample2-b
2024-11-14 16:13:49,708 [INFO] channels set to [[0 0]]
2024-11-14 16:13:49,708 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:13:55,229 [INFO] >>>> TOTAL TIME 5.52 sec
```

In [24]:
```python
# Plot results for samples
for sample, result in sresults.items():
    masks, flows, styles, diams = result
    # nummasks,_ = cv2.connectedComponents(masks.astype(np.uint8))
    nummasks = len(Counter(masks.flatten())) - 1
    fig = plt.figure(figsize=(12, 5))
    plot.show_segmentation(fig, images[sample], masks, flows[0], channels=channels[0])
    plt.tight_layout()
    plt.title(f"{sample} factor={factor}\nμm-diameter={diameter} num={nummasks}")
    plt.show()
```









We got most, but not all. We are missing at least two large ones.

[1.4] Segment structures with cellpose: rescue missed features and add to best results

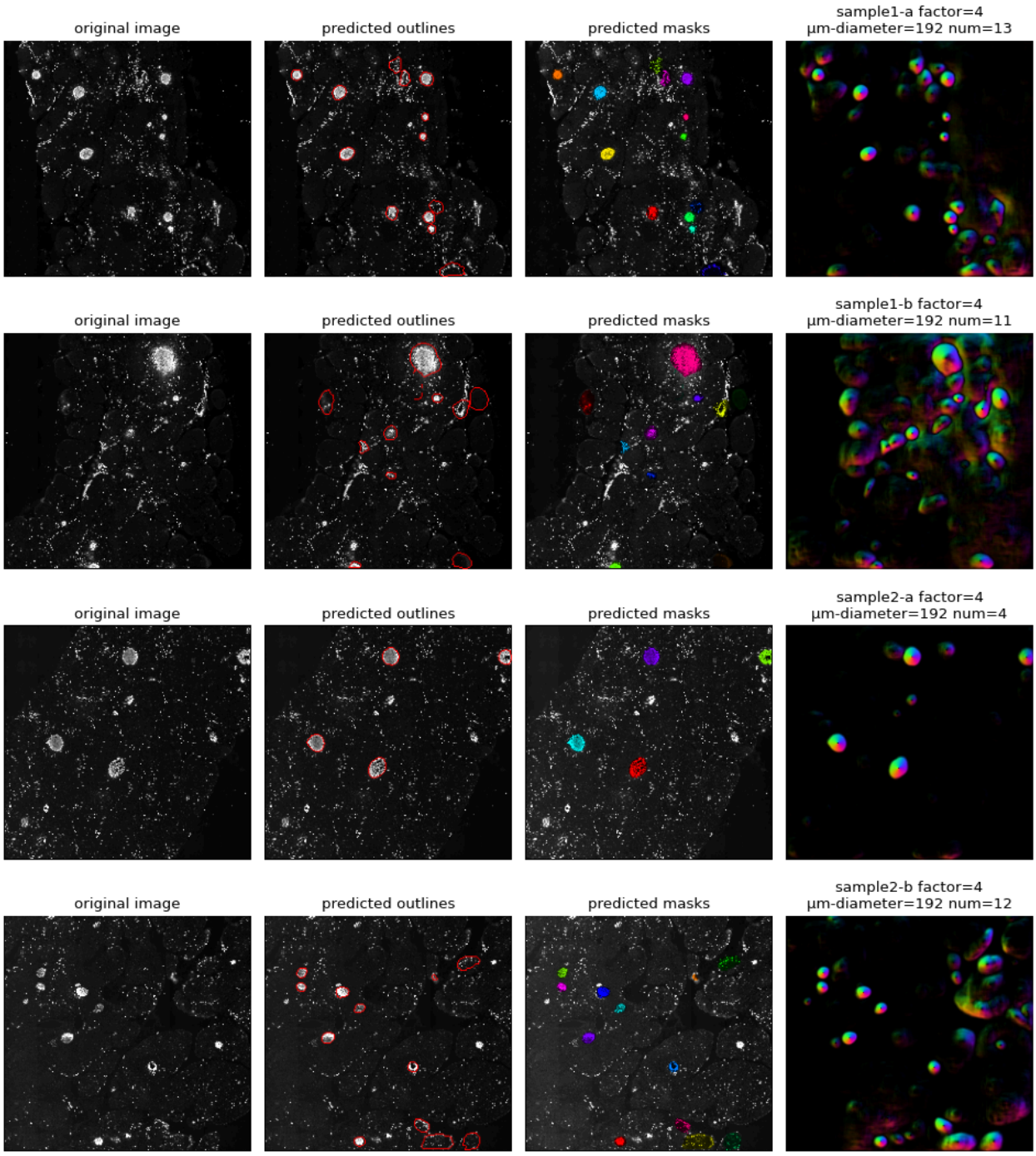
In [25]:
```
# Rescue larger structures with an increased diameter cellpose run
diameter = diameter * 2
print(diameter)
```

```
192
```

In [26]:
```
sresults2 = {}
diam = diameter / (2**factor)
print(factor, diam)

for sample, img in images.items():
    print(sample)
    masks, flows, styles, diams = model.eval(img, diameter=diam, channels=channels)
    sresults2[sample] = [masks, flows, styles, diams]
```

```
4 12.0
sample1-a
2024-11-14 16:13:56,461 [INFO] channels set to [[0 0]]
2024-11-14 16:13:56,461 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:13:58,266 [INFO] >>>> TOTAL TIME 1.80 sec
sample1-b
2024-11-14 16:13:58,267 [INFO] channels set to [[0 0]]
2024-11-14 16:13:58,267 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:14:00,108 [INFO] >>>> TOTAL TIME 1.84 sec
sample2-a
2024-11-14 16:14:00,108 [INFO] channels set to [[0 0]]
2024-11-14 16:14:00,109 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:14:01,883 [INFO] >>>> TOTAL TIME 1.77 sec
sample2-b
2024-11-14 16:14:01,884 [INFO] channels set to [[0 0]]
2024-11-14 16:14:01,884 [INFO] ~~~ FINDING MASKS ~~~
2024-11-14 16:14:03,665 [INFO] >>>> TOTAL TIME 1.78 sec
```

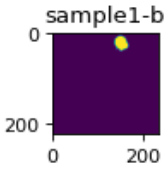
In [27]:
```
# Plot results for samples
for sample, result in sresults2.items():
    masks, flows, styles, diams = result
    nummasks = len(Counter(masks.flatten())) - 1
    fig = plt.figure(figsize=(12, 5))
    plot.show_segmentation(fig, images[sample], masks, flows[0], channels=channels[0])
    plt.tight_layout()
    plt.title(f"{sample} factor={factor}\nµm-diameter={diameter} num={nummasks}")
    plt.show()
```

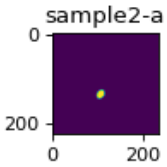


In [28]:

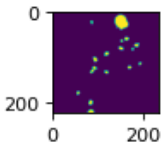
```
plt.rcParams["figure.figsize"] = (1, 1)
plt.rcParams["figure.dpi"] = 80

# For the two samples, subset masks to the missed larger ones
rescuethese = {}

for sample in ["sample1-b", "sample2-a"]:
    mask = sresults2[sample][0]
    contours, _ = cv2.findContours(
        mask.astype(np.uint8), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )

    sizecompare = 0
    largest = None

    for i, contour in enumerate(contours):
        objectmask = np.zeros_like(mask)
```

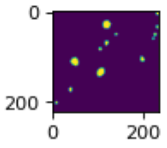
```
        cv2.drawContours(objectmask, [contour], -1, 255, -1)
        size = len(contour)

        # For sample2-a, interestingly cellpose called the size=29 object and missed the siz
        # So our assumption the largest object is the missed object is erroneous. This requi
        if (sample == "sample2-a") and (size == 28):
            size = size * 2

        if size >= sizecompare:
            sizecompare = size
            largest = objectmask

    rescuethese[sample] = largest
    plt.imshow(largest)
    plt.title(sample)
    plt.show()
```





In [29]:
```
# Merge rescued mask objects to corresponding masks (masks,flows,styles,diams)
updatedmasks = {}

for sample, cellposeresult in sresults.items():
    masks = cellposeresult[0]

    if sample in rescuethese.keys():
        addmask = rescuethese[sample]
        masks = np.ma.mask_or(masks, addmask)
        plt.imshow(masks)
        plt.show()
    else:
        masks = masks.astype(bool)

    updatedmasks[sample] = masks
```

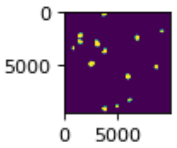




In [30]:
```
# Transform masks to full-resolution size and save to file
codexsamplemasks = {}

for sample, masks in updatedmasks.items():
    img = ski.transform.resize(masks, originalshapes[sample], order=0)
    img = img.astype(int)
    codexsamplemasks[sample] = img
```

```
    # Write masks to NPY files
    np.save(f"{sample}-factor-diameter-mask.npy", img, allow_pickle=True)

# Visualize one to check rescale
plt.imshow(img)
```

Out[30]: <matplotlib.image.AxesImage at 0x1497a2bdad00>



In [6]:
```
# Read in saved masks
codexsamplemasks = {}
for sample in snames:
    codexsamplemasks[sample] = np.load(f"{sample}-factor-diameter-mask.npy")
```

[2] Apply masks to CODEX

[2.1] Apply masks to CODEX: images to quantify protein signal in HCs across samples

A more refined analysis would subtract the cytokit cell masks from the HC masks.

In [11]:
```
# Wrangle CODEX 29 channel metadata labels. I cleaned these up from other metadata.
codexgenemap = pd.read_csv(f"{repofilepath}codexgenemap-shlee.csv", header=None)
codexgenemap.columns = ["codex-raw", "codex"]
codexgenemap[:2]
```

Out[11]:

	codex-raw	codex
0	CD107a	LAMP1
1	CD11c	ITGAX

In [12]:
```
# Wrangle CODEX 29 channel metadata labels. I cleaned these up from other metadata.
codexgenemap = pd.read_csv(f"{repofilepath}codexgenemap-shlee.csv", header=None)
codexgenemap.columns = ["codex-raw", "codex"]
codexgenemap[:2]
```

Out[12]:

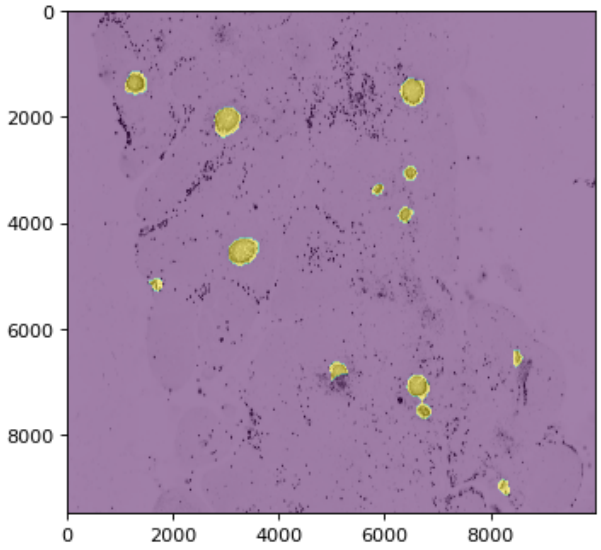
	codex-raw	codex
0	CD107a	LAMP1
1	CD11c	ITGAX

In [13]:
```
def get_pixel_values_for_mask_components(image, mask):
    num_labels, labels = cv2.connectedComponents(mask.astype(np.uint8))
    cpixelvalues, cpixelsums, cpixelareas, cmeanpixels = [], [], [], []

    for label in range(1, num_labels):  # Skip label==0==background
        cmask = labels == label
        cpixels = image[cmask]
        cpixelvalues.append(cpixels)

        pixelsum = np.sum(cpixels)
        area = np.sum(cmask)
        cpixelsums.append(pixelsum)
```

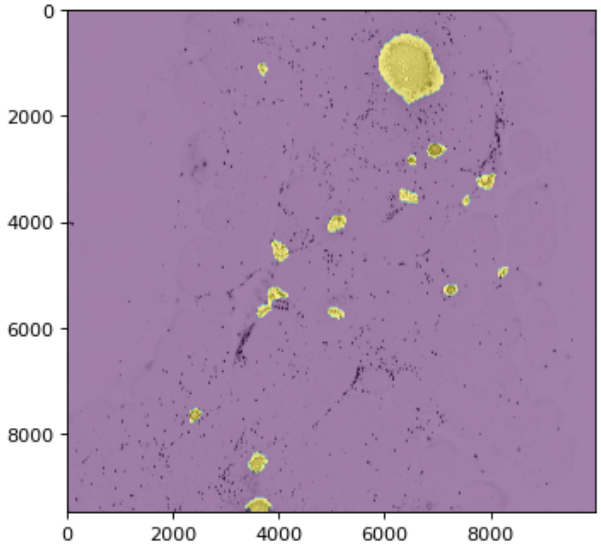
```
        cpixelareas.append(area)
        cmeanpixels.append(round(pixelsum / area, 2))

    return cpixelvalues, cpixelsums, cpixelareas, cmeanpixels
```

In [14]:
```
#### Extract protein signal from images corresponding to cellpose masks across samples and c
cmeanpixels = {}
cpixelareas = {}

for sample, samurl in codexsampleurls.items():
    print(sample)
    mask = codexsamplemasks[sample]

    # Shrink the mask a bit to pertain only to HC internal area;
    # Visual inspection showed CD15 is mutually exclusive with CD3E, so use CD3E as the mark
    for i in range(30):
        mask = ski.morphology.binary_erosion(mask)

    with tifffile.TiffFile(samurl) as ctif:
        smeanpixels = {}

        for pag in tqdm(range(len(ctif.pages)), delay=5):
            cpage = ctif.pages[pag]
            cimg = cpage.asarray()
            _, _, careas, meanpixels = get_pixel_values_for_mask_components(cimg, mask)
            smeanpixels[pag] = meanpixels

            if pag == 2:
                plt.rcParams["figure.figsize"] = (5, 5)
                plt.rcParams["figure.dpi"] = 80
                fig, ax = plt.subplots()
                plt.imshow(cimg, cmap="binary", vmax=np.max(cimg) / 1)
                plt.imshow(mask, aspect=1, alpha=0.5)
                plt.show()

        ctif.close()
        cmeanpixels[sample] = smeanpixels
        cpixelareas[sample] = careas
```

```
sample1-a
```

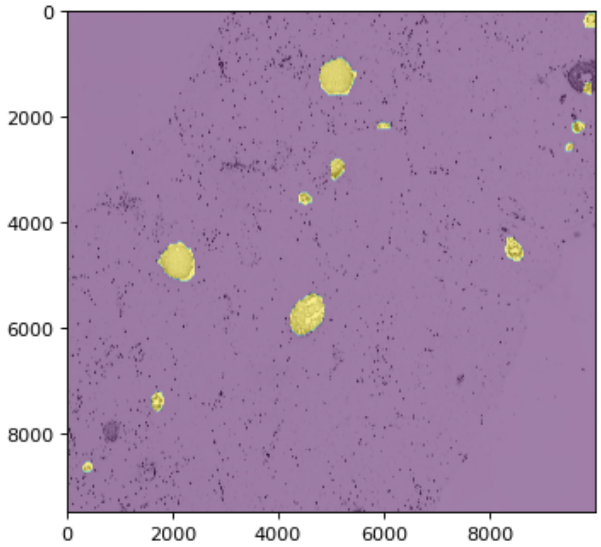



```
100%|██████████| 29/29 [00:33<00:00,  1.15s/it]
sample1-b
```



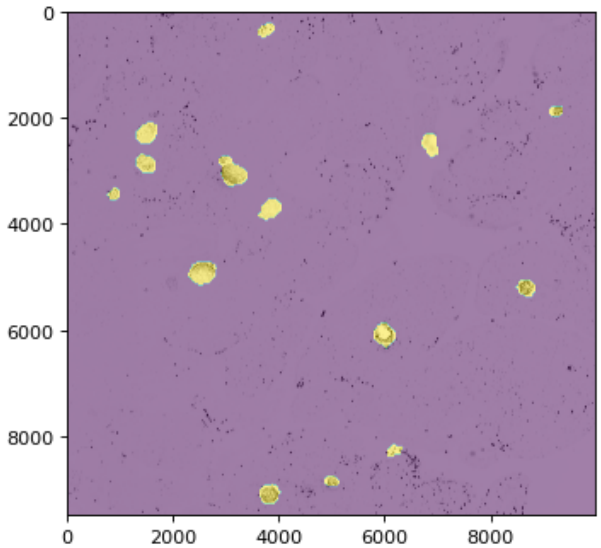

```
100%|██████████| 29/29 [00:35<00:00,  1.22s/it]
```

sample2-a




```
100%|██████████| 29/29 [00:36<00:00,  1.24s/it]
```

sample2-b




```
100%|██████████| 29/29 [00:32<00:00,  1.14s/it]
```

In [15]:
```
# Format mean pixels into dataframes; create unique barcode ids for sample HCs
samplebarcodes = {}
cmpixdf = {}

for sample, array in cmeanpixels.items():
    df = pd.DataFrame(array)
    cmpixdf[sample] = df
    numbcs = len(df)
    persamplebcs = []

    for n in range(numbcs):
        nstr = str(n).zfill(3)

        bc = f"{sample}-hc-{nstr}"
        persamplebcs.append(bc)

    samplebarcodes[sample] = persamplebcs

cmpixdf[sample][:2]
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	...	19	
0	1654.31	534.95	9330.27	470.38	404.34	362.28	2014.93	180.00	552.42	174.54	...	677.99	4221
1	1390.88	468.49	23438.16	348.06	245.80	303.44	1267.47	177.78	389.28	162.15	...	655.13	4185

2 rows × 29 columns

In [16]:
```
# Create single AnnData objects of the data. Brings the data into a familiar data format par
adfs = []

for sample, df in cmpixdf.items():
    adata = ad.AnnData(X=df)
    # Define barcodes
    adata.obs.index = samplebarcodes[sample]
    adata.obs["sample"] = sample
    adata.obs["identification-method"] = "cellpose CD15+"
    # TODO: Add spatial information for each HC, e.g. centroid positions and the size (area)
    adata.obs["area"] = cpixelareas[sample]

    # Define feature names
    adata.var.index = codexgenemap["codex"]
    adata.var["codex-raw"] = codexgenemap["codex-raw"]
    adata.var["feature_type"] = "CODEX Antibody"

    adfs.append(adata)
    print(adata)

# Given unique barcodes across samples, concatenate into single anndata object
cad = ad.concat(adfs)
```

```
AnnData object with n_obs × n_vars = 12 × 29
    obs: 'sample', 'identification-method', 'area'
    var: 'codex-raw', 'feature_type'
AnnData object with n_obs × n_vars = 16 × 29
    obs: 'sample', 'identification-method', 'area'
    var: 'codex-raw', 'feature_type'
AnnData object with n_obs × n_vars = 13 × 29
    obs: 'sample', 'identification-method', 'area'
    var: 'codex-raw', 'feature_type'
AnnData object with n_obs × n_vars = 14 × 29
    obs: 'sample', 'identification-method', 'area'
    var: 'codex-raw', 'feature_type'
```

```
/hive/user-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-pac
kages/anndata/_core/anndata.py:117: ImplicitModificationWarning: Transforming to str index.
  warnings.warn("Transforming to str index.", ImplicitModificationWarning)
/hive/user-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-pac
kages/anndata/_core/anndata.py:117: ImplicitModificationWarning: Transforming to str index.
  warnings.warn("Transforming to str index.", ImplicitModificationWarning)
/hive/user-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-pac
kages/anndata/_core/anndata.py:117: ImplicitModificationWarning: Transforming to str index.
  warnings.warn("Transforming to str index.", ImplicitModificationWarning)
/hive/user-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-pac
kages/anndata/_core/anndata.py:117: ImplicitModificationWarning: Transforming to str index.
  warnings.warn("Transforming to str index.", ImplicitModificationWarning)
/hive/user-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-pac
kages/anndata/_core/anndata.py:117: ImplicitModificationWarning: Transforming to str index.
  warnings.warn("Transforming to str index.", ImplicitModificationWarning)
/hive/user-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-pac
kages/anndata/_core/anndata.py:117: ImplicitModificationWarning: Transforming to str index.
  warnings.warn("Transforming to str index.", ImplicitModificationWarning)
/hive/user-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-pac
kages/anndata/_core/anndata.py:117: ImplicitModificationWarning: Transforming to str index.
  warnings.warn("Transforming to str index.", ImplicitModificationWarning)
/hive/user-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-pac
kages/anndata/_core/anndata.py:117: ImplicitModificationWarning: Transforming to str index.
  warnings.warn("Transforming to str index.", ImplicitModificationWarning)
```

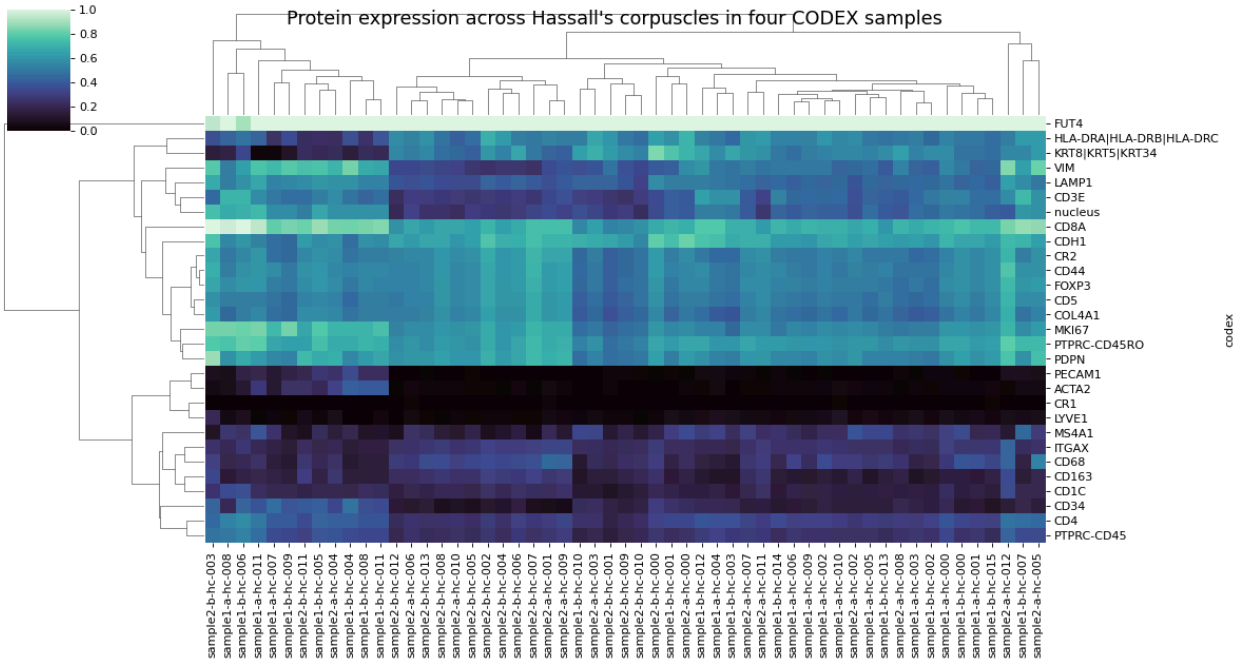
In [17]:
```python
# Process as if single-cell RNA-Seq data.
sc.pp.filter_cells(cad, min_counts=2)
sc.pp.filter_genes(cad, min_cells=2)
cad.layers["raw"] = cad.X.copy()
sc.pp.normalize_total(cad, inplace=True)
sc.pp.log1p(cad)
sc.pp.pca(cad)
sc.pp.neighbors(cad)
sc.tl.umap(cad)
sc.tl.leiden(cad, n_iterations=2)
```

```
2024-11-14 22:29:20.185068: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorF
low binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following
CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-11-14 22:29:22.400710: W tensorflow/compiler/xla/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: c
annot open shared object file: No such file or directory; LD_LIBRARY_PATH: /hive/user-workspa
ces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-packages/cv2/../../li
b64:/lib/:/lib64/:/usr/local/lib
2024-11-14 22:29:22.410538: I tensorflow/compiler/xla/stream_executor/cuda/cudart_stub.cc:29]
Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2024-11-14 22:29:34.254153: W tensorflow/compiler/xla/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: canno
t open shared object file: No such file or directory; LD_LIBRARY_PATH: /hive/user-workspaces/
sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-packages/cv2/../../lib6
4:/lib/:/lib64/:/usr/local/lib
2024-11-14 22:29:34.254409: W tensorflow/compiler/xla/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_plug
in.so.7: cannot open shared object file: No such file or directory; LD_LIBRARY_PATH: /hive/us
er-workspaces/sleep/1338/.JupyterLabJobPythonNonGPUCommon_venv/lib/python3.9/site-packages/cv
2/../../lib64:/lib/:/lib64/:/usr/local/lib
2024-11-14 22:29:34.254421: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Wa
rning: Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with Tensor
RT, please make sure the missing libraries mentioned above are installed properly.
```

In [18]:
```python
# Save to h5ad
cad.write_h5ad("cad.h5ad")
```

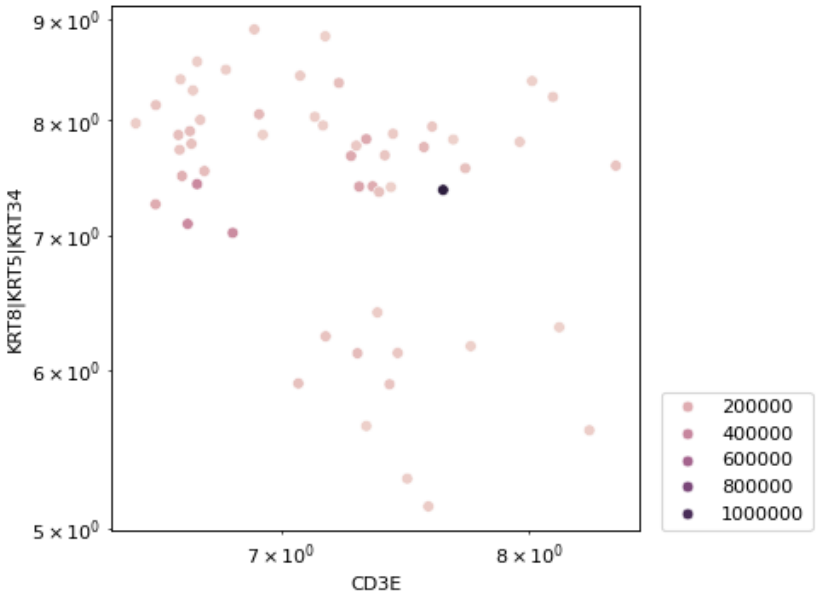
In [19]:
```python
g = sns.clustermap(
    cad.to_df().T,
```

```
    standard_scale=1,
    cmap="mako",
    figsize=(15, 8),
)
g.fig.suptitle(
    "Protein expression across Hassall's corpuscles in four CODEX samples", size=16
)
plt.show()
```



A subset of HCs show differential expression.

In [20]:
```
# Explore gene x gene relationships while factoring for HC size.
sns.scatterplot(
    x="CD3E",
    y="KRT8|KRT5|KRT34",
    data=cad.to_df().merge(cad.obs.area, how="left", left_index=True, right_index=True),
    hue="area",
)
plt.legend(loc=(1.04, 0))
plt.xscale("log")
plt.yscale("log")
plt.show()
```



[2.2] Apply masks to CODEX: Cytokit segmented cells to identify cell markers and subset HC cells across samples

In [21]:
```
warnings.filterwarnings("ignore", category=Warning)
```

In [22]:
```
codexanndataurls.values()
```

Out[22]:
```
dict_values(['datasets/43213991a54ce196d406707ffe2e86bd/anndata-zarr/reg1_stitched_expressions-anndata.zarr', 'datasets/d4e9ec618924a8d43cfe1e67c38c1447/anndata-zarr/reg1_stitched_expressions-anndata.zarr', 'datasets/822c9163d3be9b427dd0830f69a12305/anndata-zarr/reg1_stitched_expressions-anndata.zarr', 'datasets/37d06bb991afa2beb7b9460e746247ad/anndata-zarr/reg1_stitched_expressions-anndata.zarr'])
```

In [23]:
```
# Process AnnData Zarrs across samples
fulladatas = {}

for sample, url in tqdm(codexanndataurls.items()):
    print(url)

    # HuBMAP formats anndata objects as Zarr
    adata = ad.read_zarr(url)

    # If from h5ad:
    # adata = ad.read_h5ad(f'{sample}.h5ad')

    adata.obs["sample"] = sample
    adata.obs["patient"] = sample.split("-")[0]

    # Switch out feature labels
    adata.var = adata.var.merge(
        codexgenemap, how="left", left_index=True, right_on="codex-raw"
    )
    adata.var.set_index("codex", inplace=True)

    fulladatas[sample] = adata
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
datasets/43213991a54ce196d406707ffe2e86bd/anndata-zarr/reg1_stitched_expressions-anndata.zarr
 25%|██▌       | 1/4 [00:04<00:12,  4.30s/it]
datasets/d4e9ec618924a8d43cfe1e67c38c1447/anndata-zarr/reg1_stitched_expressions-anndata.zarr
 50%|█████     | 2/4 [00:07<00:06,  3.39s/it]
datasets/822c9163d3be9b427dd0830f69a12305/anndata-zarr/reg1_stitched_expressions-anndata.zarr
```

```
 75%|███████   | 3/4 [00:15<00:05,  5.79s/it]
datasets/37d06bb991afa2beb7b9460e746247ad/anndata-zarr/reg1_stitched_expressions-anndata.zarr
100%|██████████| 4/4 [00:22<00:00,  5.64s/it]
```

In [24]:
```python
# Label cells as inside or outside of HCs using masks we got from cellpose
for sample, mask in tqdm(codexsamplemasks.items()):
    adata = fulladatas[sample]
    insideoroutside = []

    # Shrink the mask a bit to pertain only to HC internal area;
    # Visual inspection showed CD15 is mutually exclusive with CD3E, so use CD3E as the mark
    for i in range(30):
        mask = ski.morphology.binary_erosion(mask)

    for coordinates in adata.obsm["xy"]:
        x = int(coordinates[1])
        y = int(coordinates[0])

        if x < mask.shape[0] and y < mask.shape[1]:
            if mask[x, y]:
                insideoroutside.append(True)
            else:
                insideoroutside.append(False)
        else:
            insideoroutside.append(False)

    adata.obs["HC"] = insideoroutside
```

```
100%|██████████| 4/4 [01:02<00:00, 15.65s/it]
```

In [25]:
```python
# Subset anndata objects to cells inside of HCs
subdfs = {}

for sample, adata in fulladatas.items():
    sub = adata[adata.obs["HC"]]

    # Create unique barcode ids
    sub.obs.index = (
        np.array(sub.obs["sample"])
        + "_"
        + np.array([str(x) for x in list(sub.obs.index)])
    )
    subdfs[sample] = sub
```

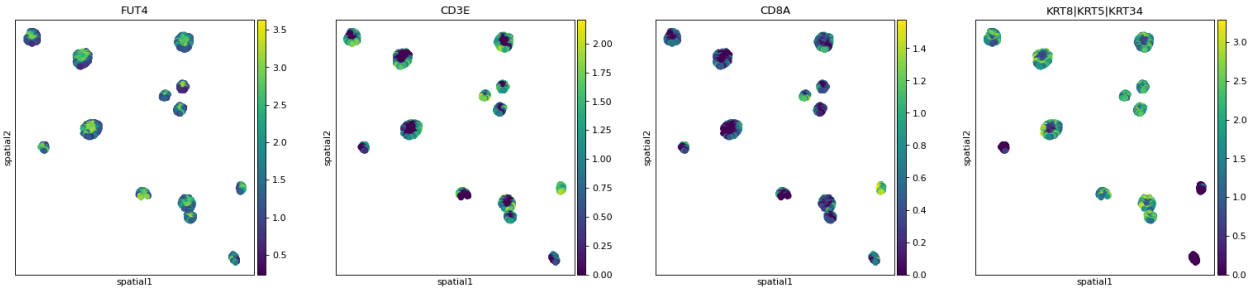
In [26]:
```python
# Merge the sample anndata objects
cad2 = ad.concat(subdfs.values())
```

In [27]:
```python
# Format spatial data for squidpy
xyzip = np.array([list(t) for t in zip(*cad2.obsm["xy"])])
cad2.obsm["x"] = xyzip[0]
cad2.obsm["y"] = xyzip[1]
cad2.obsm["spatial"] = cad2.obsm["xy"]

# Process like single-cell RNA-Seq
sc.pp.normalize_total(cad2, inplace=True)
sc.pp.log1p(cad2)
sc.pp.pca(cad2)
sc.pp.neighbors(cad2)
sc.tl.umap(cad2)
sc.tl.leiden(cad2)
```

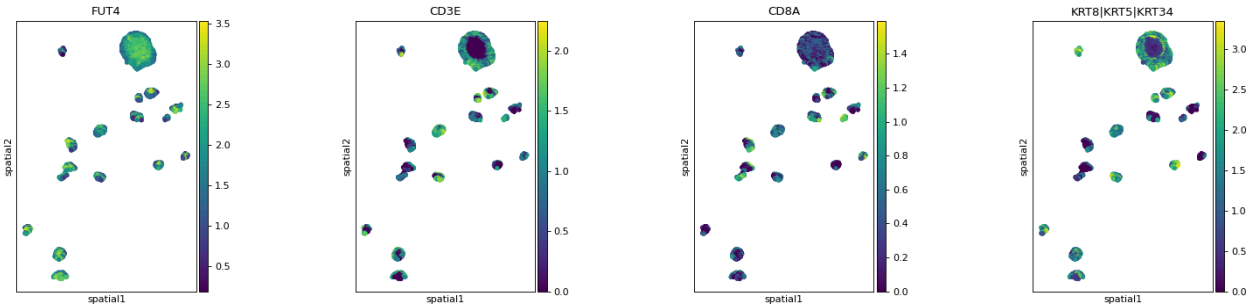
In [28]:
```python
# Write to h5ad
cad2.write_h5ad("cad2.h5ad")
```

In [29]:
```
# Visualize each tissue sample for a subset of features
mygenes = ["FUT4", "CD3E", "CD8A", "KRT8|KRT5|KRT34"]

for sample in snames:
    print(sample)
    sq.pl.spatial_scatter(
        cad2[cad2.obs["sample"] == sample],
        library_id="spatial",
        shape=None,
        color=mygenes,
        # wspace=0.4,
    )
    plt.show()
```

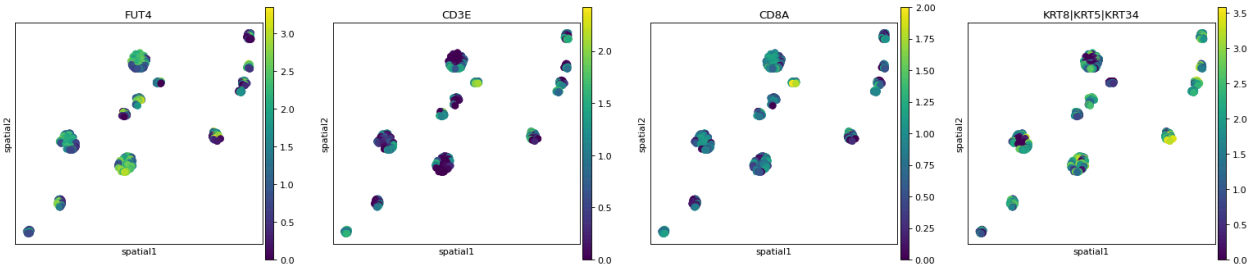
sample1-a



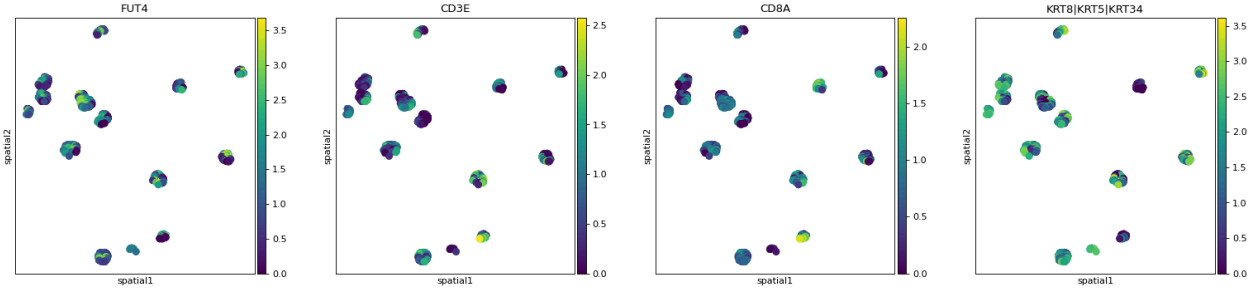
sample1-b



sample2-a



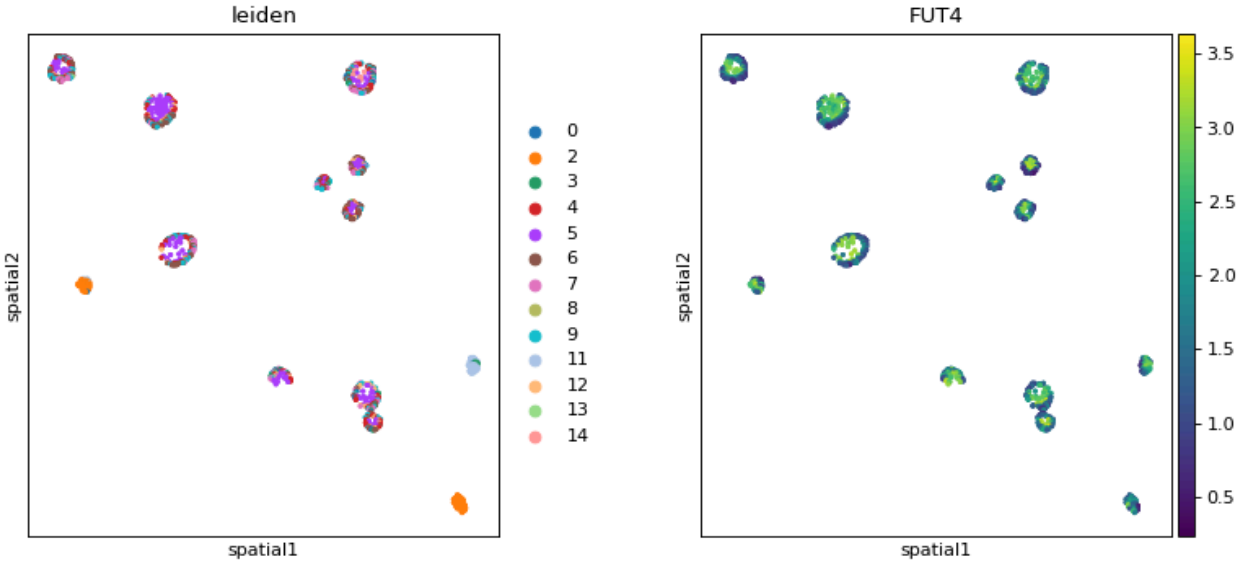
sample2-b



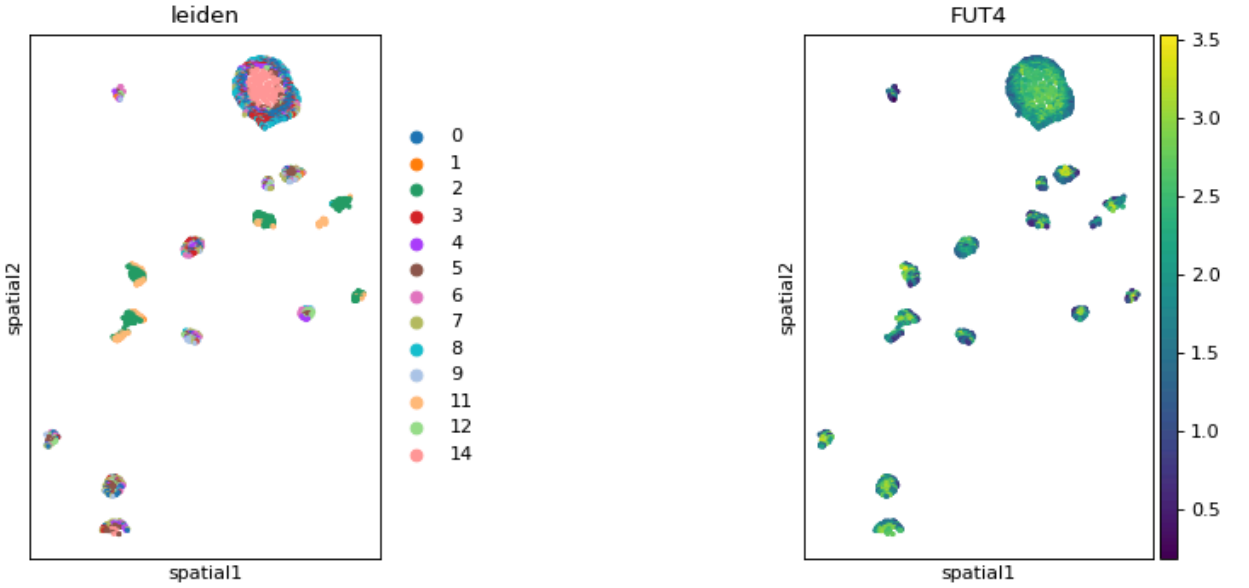
In [30]:
```
# Plot each sample
for sample in snames:
    print(sample)
    sq.pl.spatial_scatter(
        cad2[cad2.obs["sample"] == sample],
        library_id="spatial",
```

```
        shape=None,
        size=12,
        color=["leiden", "FUT4"],
    )
    # plt.title(sample, size=16)
    plt.show()
```

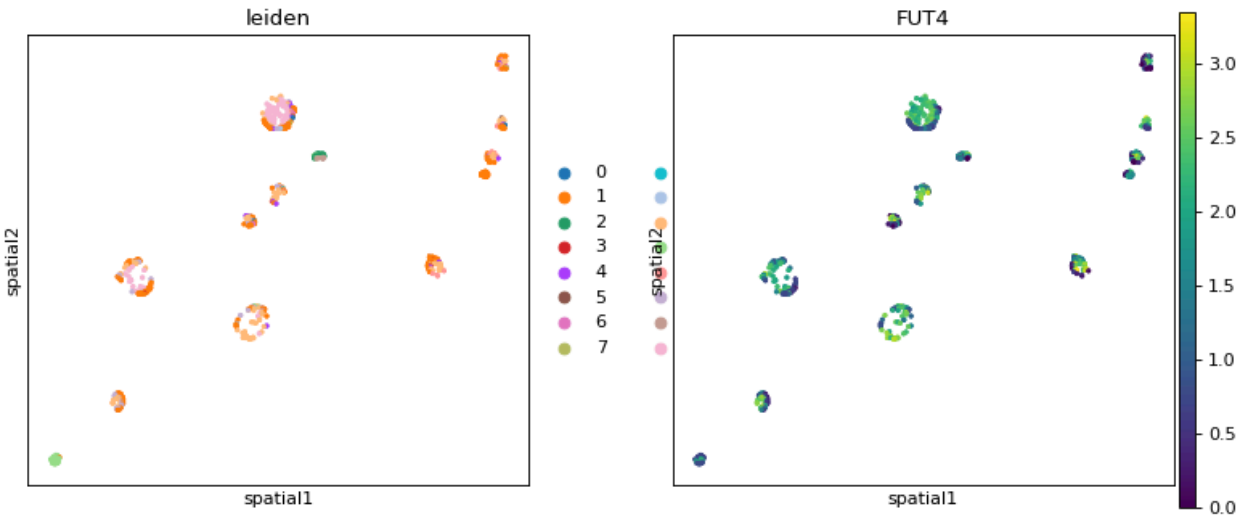
sample1-a



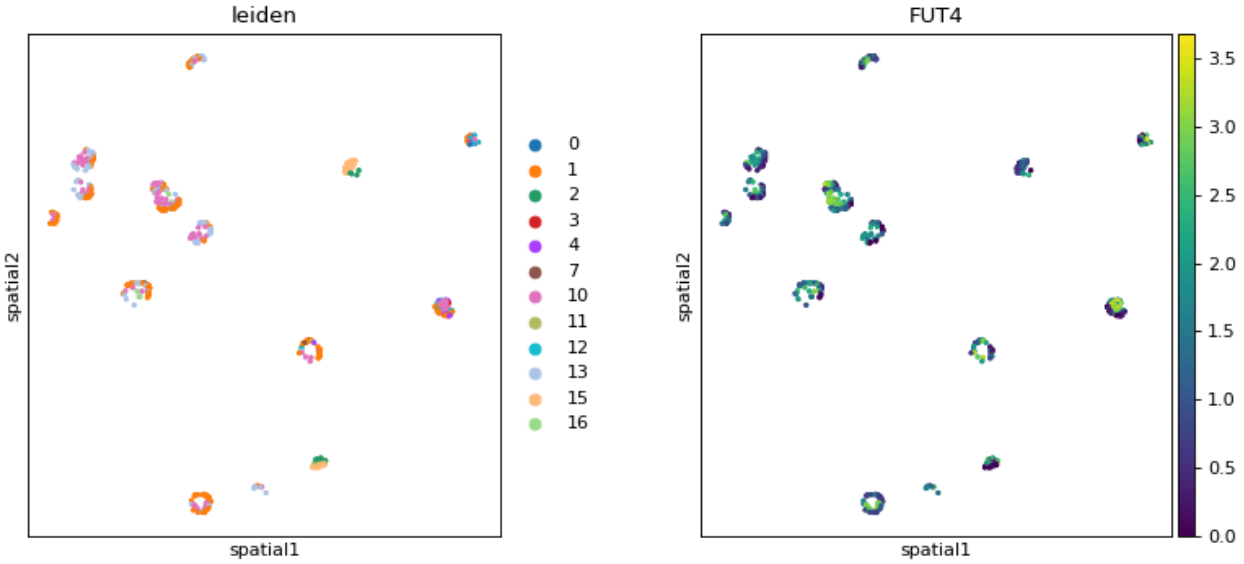
sample1-b



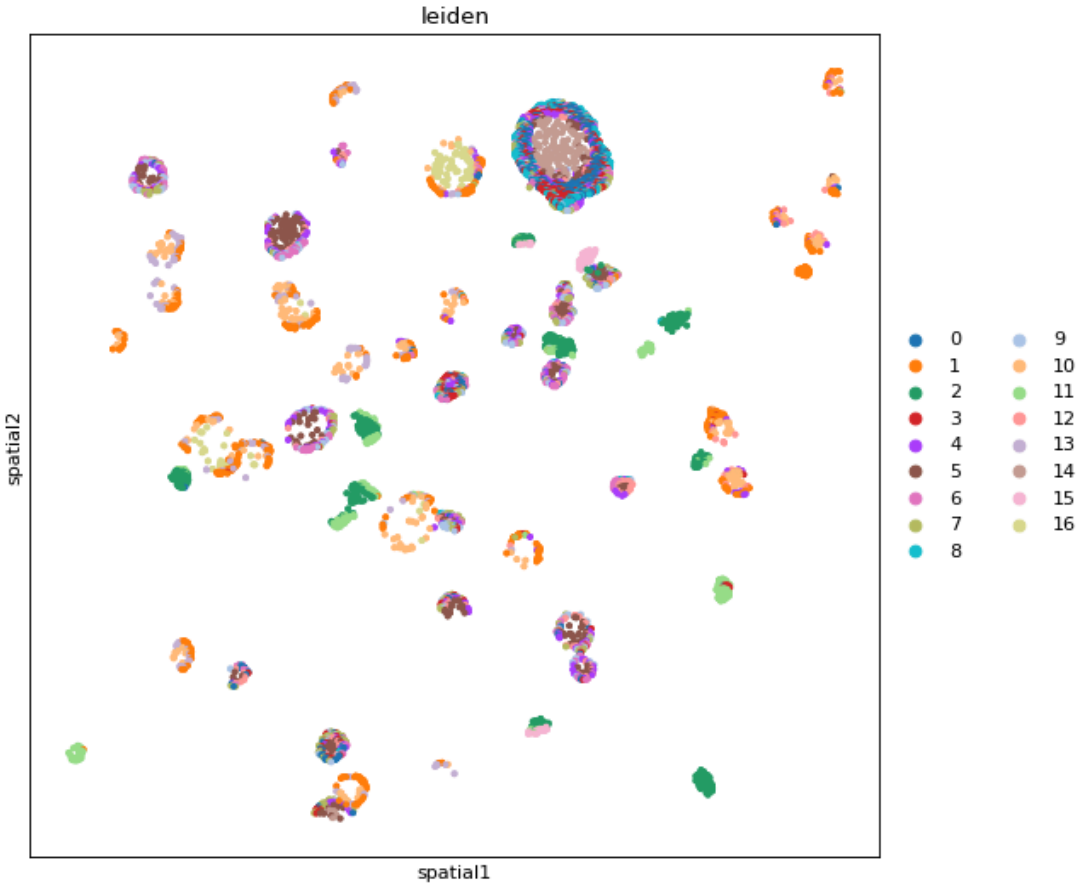
sample2-a



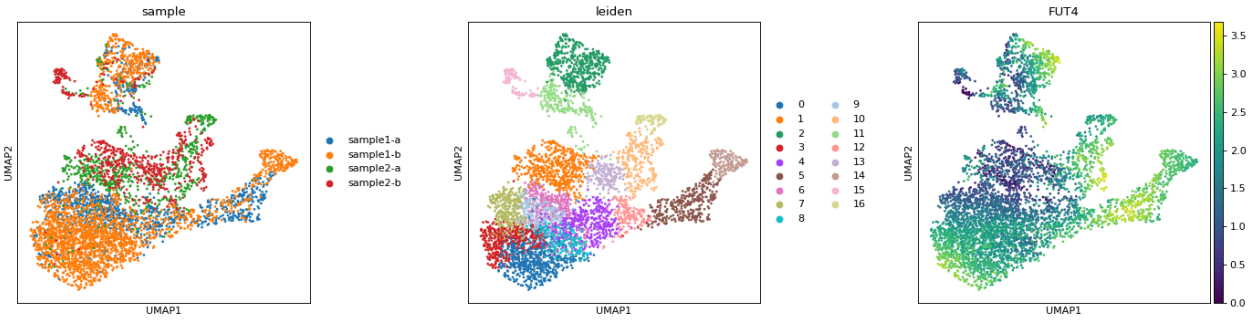
sample2-b



In [31]:
```
# Plot all samples in same space
sq.pl.spatial_scatter(
    cad2,
    library_id="spatial",
    shape=None,
    color=["leiden"],
    figsize=(8, 8),
)
# plt.title(sample, size=16)
plt.show()
```



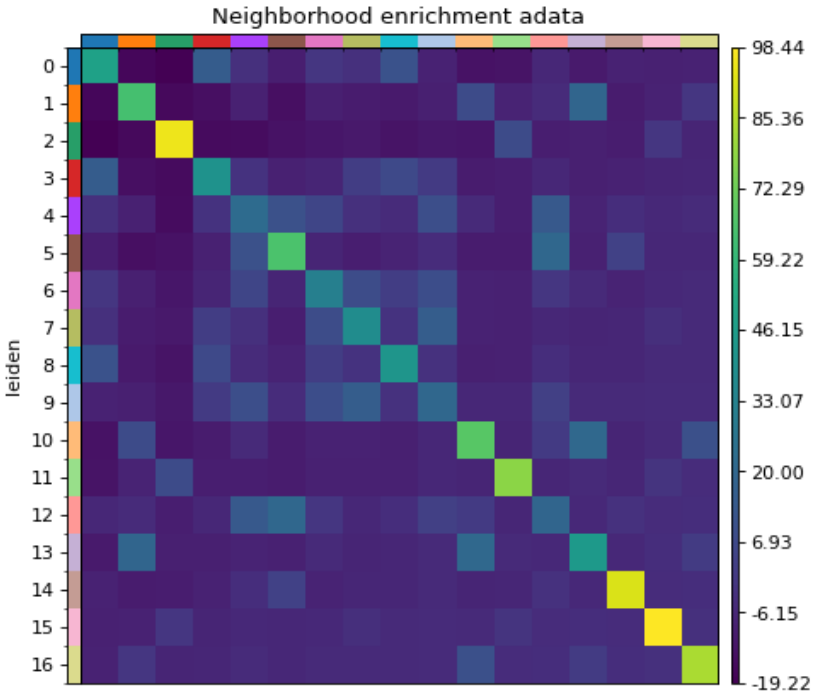
In [32]:
```
# visualize umap of clusters for all samples
sc.pl.umap(
    cad2,
    color=["sample", "leiden", "FUT4"],
    wspace=0.4,
)
```



Sample1 and sample2 individuals are different. The 'a' and 'b' labels denote different tissue slices from an individual. The difference between the two patient samples could be real or could be a batch effect.

In [33]:
```
# Neighborhood enrichment
sq.gr.spatial_neighbors(cad2, coord_type="generic", delaunay=True)
sq.gr.nhood_enrichment(cad2, cluster_key="leiden")
sq.pl.nhood_enrichment(
    cad2,
    cluster_key="leiden",
    figsize=(5, 5),
    title="Neighborhood enrichment adata",
)
```

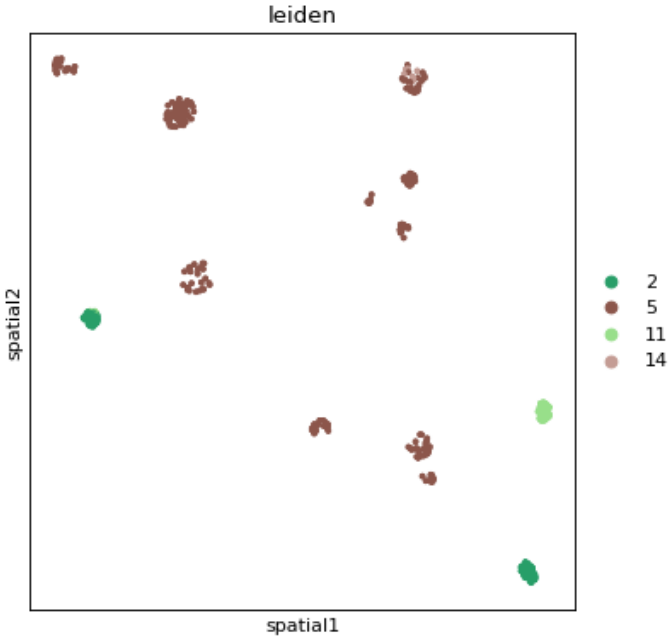
```
0%|          | 0/1000 [00:00<?, ?/s]
```



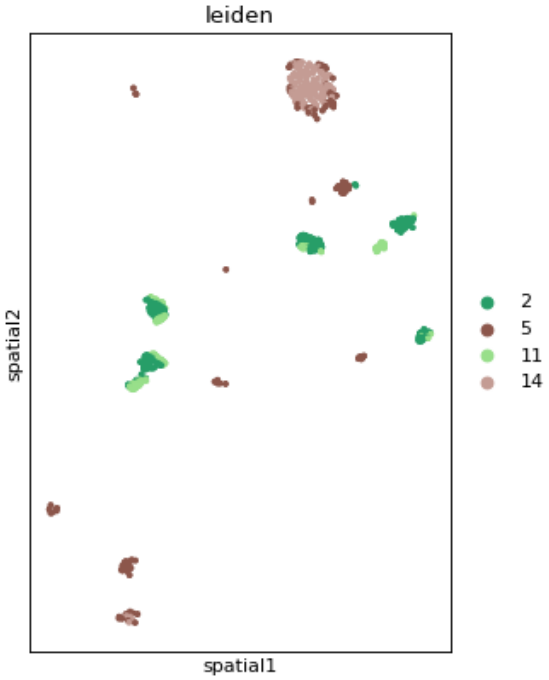
In [34]:
```
lclustersoi = ["2", "5", "10", "11", "14", "16"]
```

In [35]:
```
# Plot specific leiden clusters
for sample in snames:
    print(sample)
    sq.pl.spatial_scatter(
        cad2[(cad2.obs["leiden"].isin(lclustersoi)) & (cad2.obs["sample"] == sample)],
        library_id="spatial",
        shape=None,
        color=["leiden"],
        figsize=(5, 5),
        size=20,
    )
    plt.show()
```

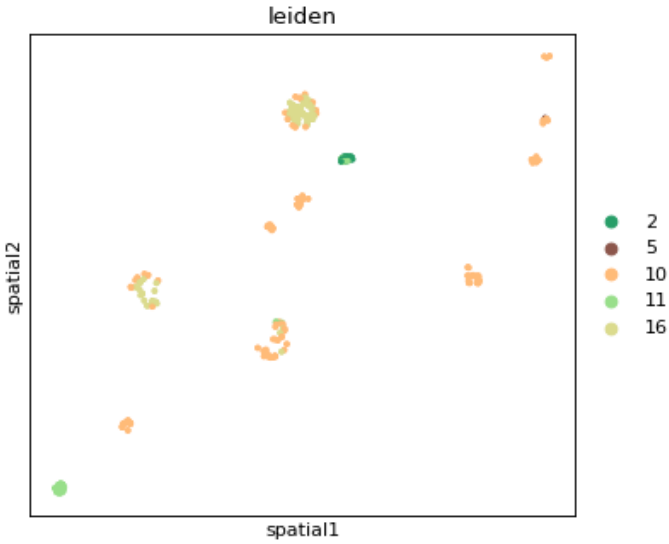
sample1-a



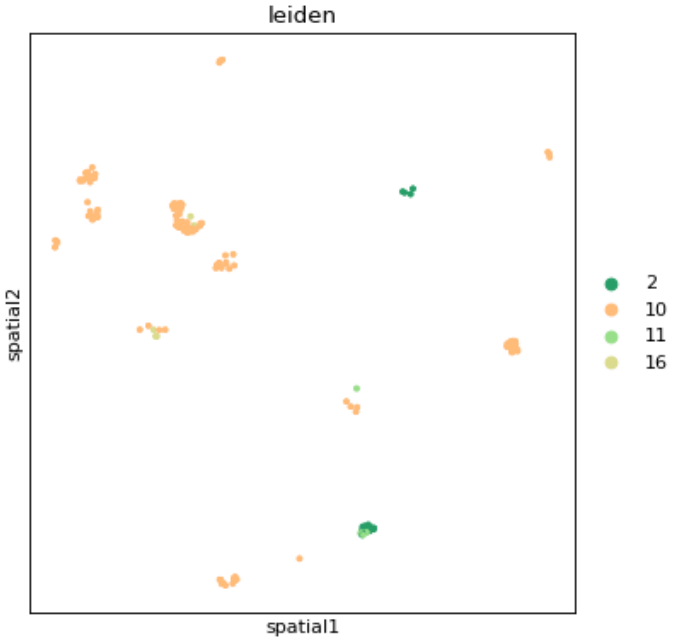
sample1-b



sample2-a

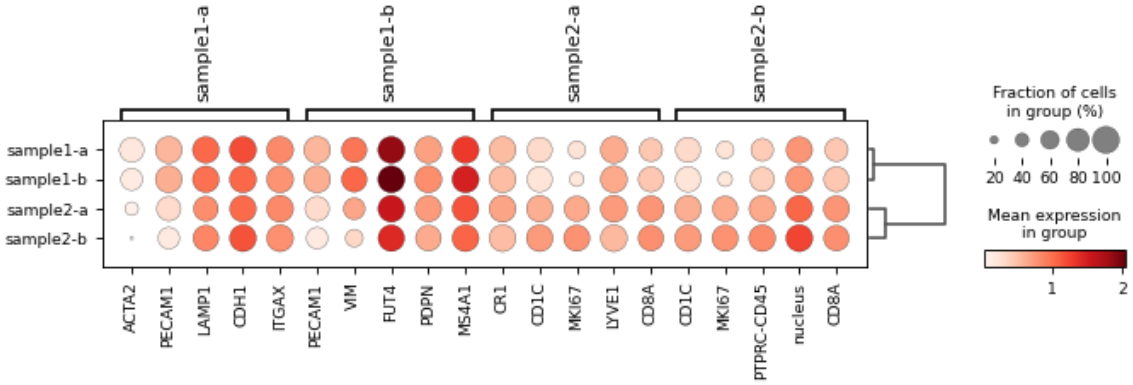


sample2-b



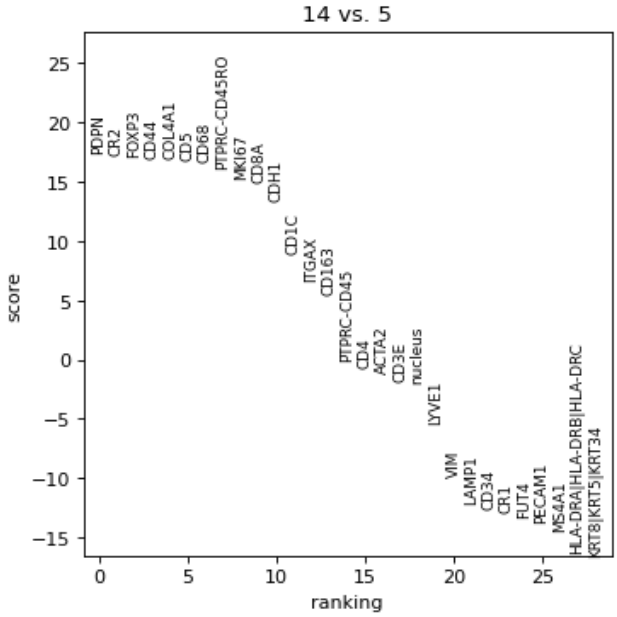
In [36]:
```
sc.tl.rank_genes_groups(cad2, "sample", method="wilcoxon", key_added="wilcoxon")
sc.pl.rank_genes_groups_dotplot(cad2, n_genes=5, key="wilcoxon", groupby="sample")
```

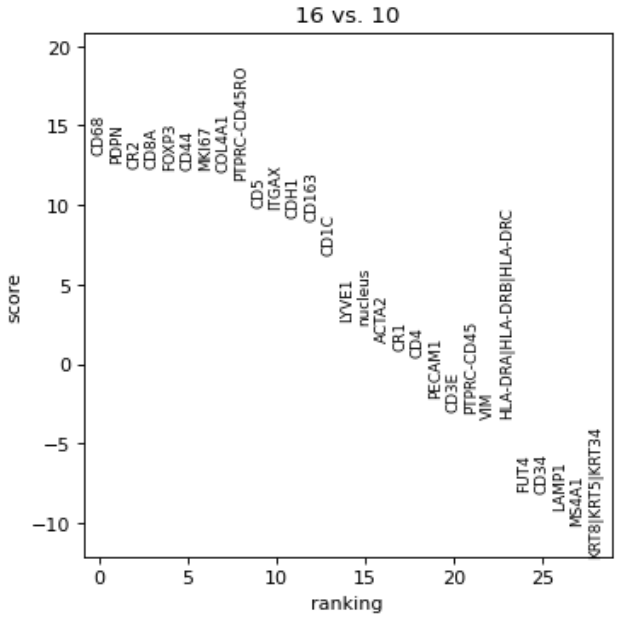
```
WARNING: dendrogram data not found (using key=dendrogram_sample). Running `sc.tl.dendrogram`
with default parameters. For fine tuning it is recommended to run `sc.tl.dendrogram` independ
ently.
```



In [37]:
```
# sc.tl.rank_genes_groups(cad2, 'leiden', method='wilcoxon', key_added = "wilcoxon")
# sc.pl.rank_genes_groups_dotplot(cad2, n_genes=5, key="wilcoxon", groupby="leiden")
```

In [38]:
```
# Identify dferentally expressed genes for specific clusters against a reference
csets = [
    ["5", "14"],
    ["10", "16"],  # ['2', '11'], ['11', '14'], ['5', '11'], ['5', '2']
]
for pair in csets:
    sc.tl.rank_genes_groups(
        cad2, "leiden", method="wilcoxon", groups=[pair[1]], reference=pair[0]
    )
    sc.pl.rank_genes_groups(cad2, groups=[pair[1]], n_genes=29)
```



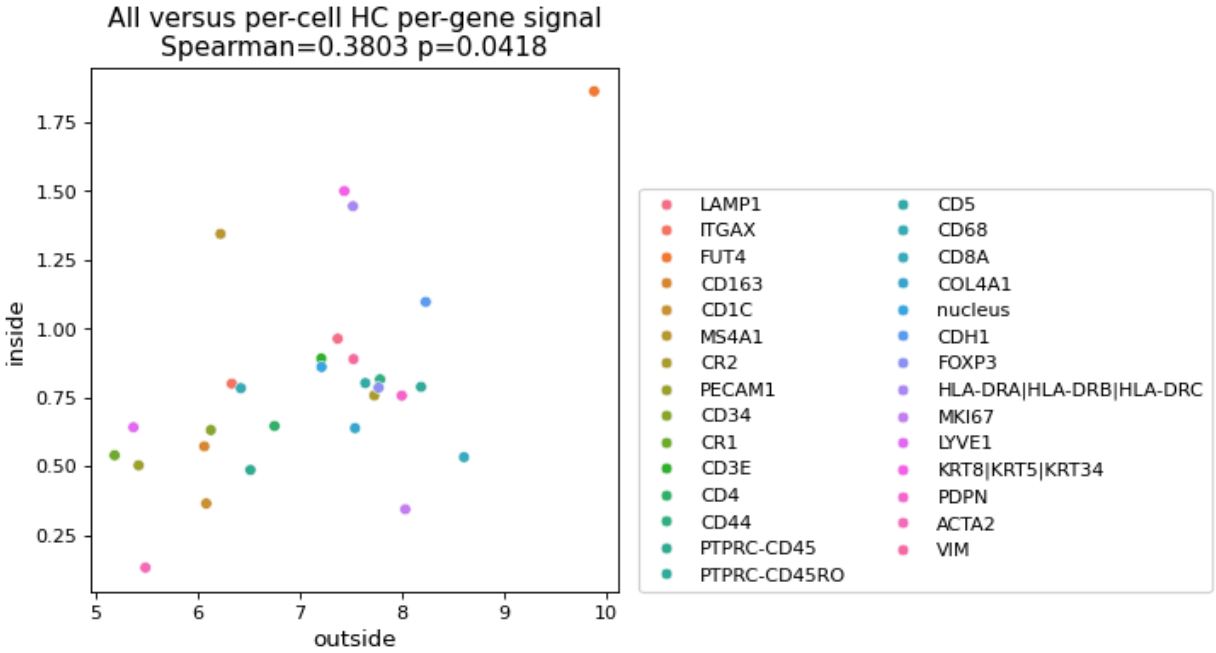


[3] Pseudobulk comparisons

[3.1] Pseduobulk comparisons: CODEX protein expression inside versus outside of cells in HCs

In [39]:
```
# Pseudobulk per-gene comparison normalized by number of ids (structures or cells)
x = cad.to_df().sum() / len(cad)
y = cad2.to_df().sum() / len(cad2)
srs, srsp = scipy.stats.spearmanr(x, y)
srs = round(srs, 4)
srsp = round(srsp, 4)

sns.scatterplot(x=x, y=y, hue=x.index)
plt.xlabel("outside", size=12)
plt.ylabel("inside", size=12)
plt.title(f"All versus per-cell HC per-gene signal\nSpearman={srs} p={srsp}", size=14)
```

```
plt.legend(loc=(1.04, 0), ncol=2)
plt.show()
```



We see a positive correlation with a weak rank score. MS4A1, KRTs and HLA-DRs are above the diagnal, while MKI67 and CD8A are below the diagnal. These correspond to differentially expressed in the previous section's gene rank plots of

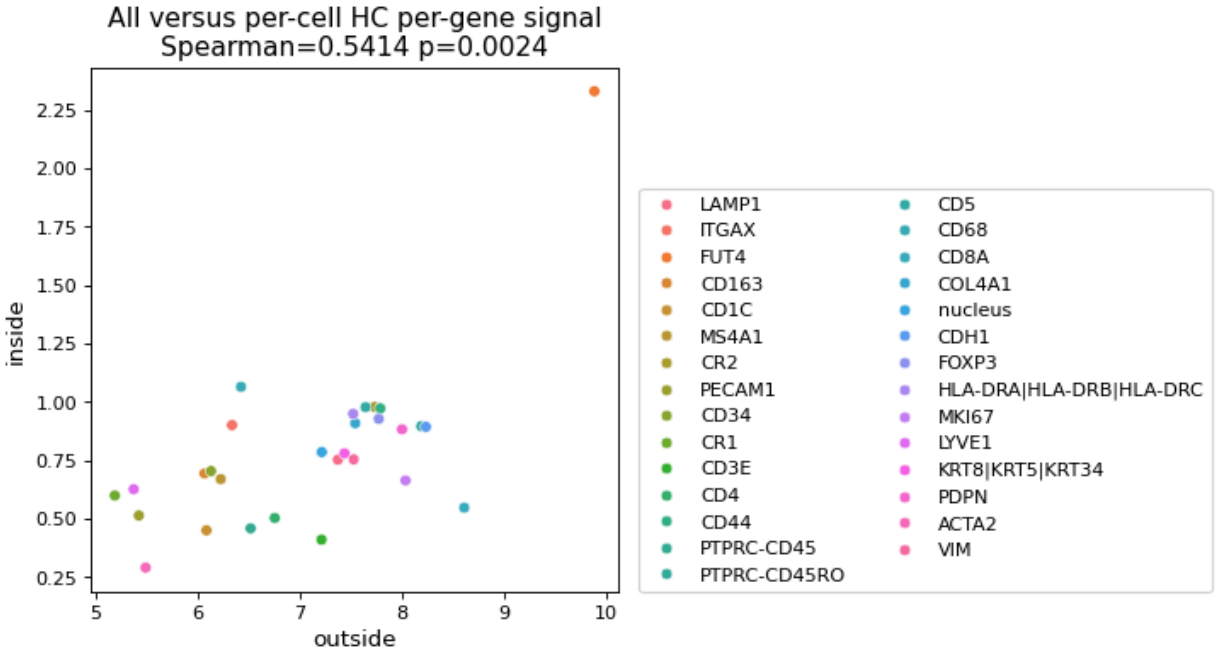
- Inside leiden clusters 14 & 16
- Outside leiden clusters 5 & 10

In [40]:
```
lclustersoi
```

Out[40]:
```
['2', '5', '10', '11', '14', '16']
```

In [41]:
```
discrepantgenes = [
    "MS4A1",
    "KRT8|KRT5|KRT34",
    "HLA-DRA|HLA-DRB|HLA-DRC",
    "MKI67",
    "CD8A",
]
```

In [42]:
```
# Subset the correlation to the more central leiden clusters
x = cad.to_df().sum() / len(cad)
y = cad2[cad2.obs["leiden"].isin(lclustersoi)].to_df().sum() / len(
    cad2[cad2.obs["leiden"].isin(lclustersoi)]
)
srs, srsp = scipy.stats.spearmanr(x, y)
srs = round(srs, 4)
srsp = round(srsp, 4)

sns.scatterplot(x=x, y=y, hue=x.index)
plt.xlabel("outside", size=12)
plt.ylabel("inside", size=12)
plt.title(f"All versus per-cell HC per-gene signal\nSpearman={srs} p={srsp}", size=14)
plt.legend(loc=(1.04, 0), ncol=2)
plt.show()
```



[3.2] Pseduobulk comparisons: CODEX versus 2D IMC thymus HCs

HuBMAP offers thymus 2D Imaging Mass Cytometry (IMC) data as well as a 3D IMC data. Look into one of the 2D samples.

In [43]:
```
# Manually curated mask from QuPath and the image path
imsdir = f"{ddir}9a6403bb0423e62950926a7d4fdab45b/ometiff/20191202_HuBMAP_Thymus/"
imsimgurl = f"{imsdir}20191202_HuBMAP_Thymus_s0_p8_r6_a6_ac.ome.tiff"  # non-pyramidal

# Feature map I cleaned up previously
channamesurl = f"{repofilepath}channelnames-shlee.csv"
channames = pd.read_csv(channamesurl)
channames[:2]
```

Out[43]:

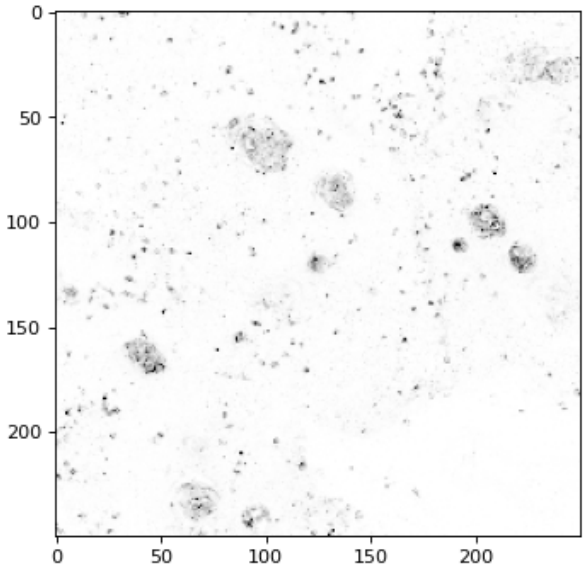
	metal	channel	target	name
0	ArAr80	0	80ArAr	na
1	Y89	1	HLA-ABC_19602744Y89	HLA-A|HLA-B|HLA-C

In [44]:
```
# Create data object for 2D IMS sample image data by binning.
# This dataset has stains that enable nuclei and cell segmentation. There are many tutorials
# Taking simple approach of 4x4 binning the signal. Each pixel is originally 1 µm in size.
ichans = {}
iarrays = {}
factor = 0.25

for i in range(len(channames)):

    with tifffile.TiffFile(imsimgurl) as itif:
        ipage = itif.pages[i]
        iimg = ipage.asarray()
        resample = scipy.ndimage.zoom(
            iimg,
            factor,
            order=1,
        )  # grid_mode=True)
        iimgdf = pd.DataFrame(resample)
        iarrays[i] = resample
        ichans[i] = iimgdf
```

```
print(iimg.shape, resample.shape)
plt.imshow(iarrays[44], cmap="binary", vmax=np.max(iarrays[44]) / 3)
```

```
(1000, 1000) (250, 250)
```

Out[44]: <matplotlib.image.AxesImage at 0x14d448e60b80>



In [45]:
```
# Create barcode ids
barcodes = []
spatial = []

for c in range(len(ichans[0])):
    for r in range(len(ichans[0])):
        clabel = "{:03d}".format(c)
        rlabel = "{:03d}".format(r)
        barcodes.append(f"imc-x{clabel}y{rlabel}")
        spatial.append([c, r])

print(len(barcodes))
spatial = np.array(spatial)
```

```
62500
```

In [46]:
```
# col,row
percellarrays = []
features = []

for k, a in iarrays.items():
    gene = list(channames[channames["channel"] == k]["name"])[0]

    if gene == "na":
        continue
    else:
        percellarray = []
        features.append(gene)

        # Create barcode ids
        for c in range(len(ichans[0])):
            for r in range(len(ichans[0])):
                clabel = "{:03d}".format(c)
                rlabel = "{:03d}".format(r)
                label = f"imc-x{clabel}y{rlabel}"

                pv = a[c][r]
                percellarray.append(pv)
```

```
        percellarrays.append(np.array(percellarray))
```

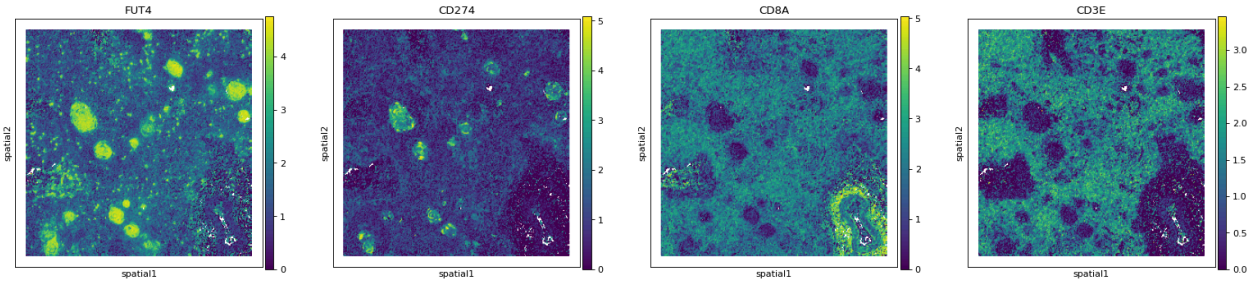
In [47]:
```
# Transpose the per cell arrays into per feature arrays
perfeaturearrays = np.array(list(zip(*percellarrays)))

# Create AnnData object
idata = sc.AnnData(perfeaturearrays, obsm={"spatial": spatial})
idata.var.index = features
idata.obs["ID"] = barcodes
idata.obs.index = idata.obs["ID"]
xyzip = np.array([list(t) for t in zip(*idata.obsm["spatial"])])
x = xyzip[0]
y = xyzip[1]
idata.obsm["x"] = x
idata.obsm["y"] = y
```

In [48]:
```
sc.pp.filter_cells(idata, min_counts=10)
sc.pp.filter_genes(idata, min_cells=10)
idata.layers["counts"] = idata.X.copy()
sc.pp.normalize_total(idata, inplace=True)
sc.pp.log1p(idata)
sc.pp.pca(idata)
sc.pp.neighbors(idata)
sc.tl.umap(idata)
```

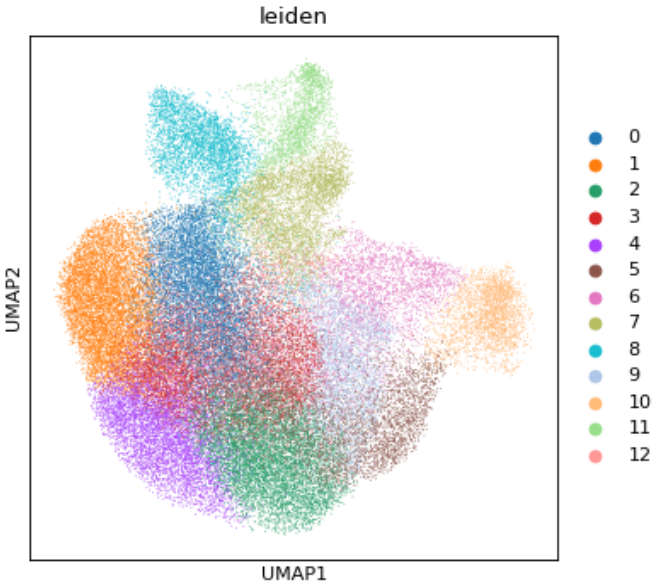
In [49]:
```
# Takes some minutes
sc.tl.leiden(idata)
# idata.write(f'{ddir}idata-20241004-shlee.h5ad')
```

In [50]:
```
# Write to h5ad
idata.write_h5ad("idata.h5ad")
```

In [51]:
```
sq.pl.spatial_scatter(
    idata,
    library_id="spatial",
    color=["FUT4", "CD274", "CD8A", "CD3E"],
    shape=None,
    size=2,
    img=False,
)
```

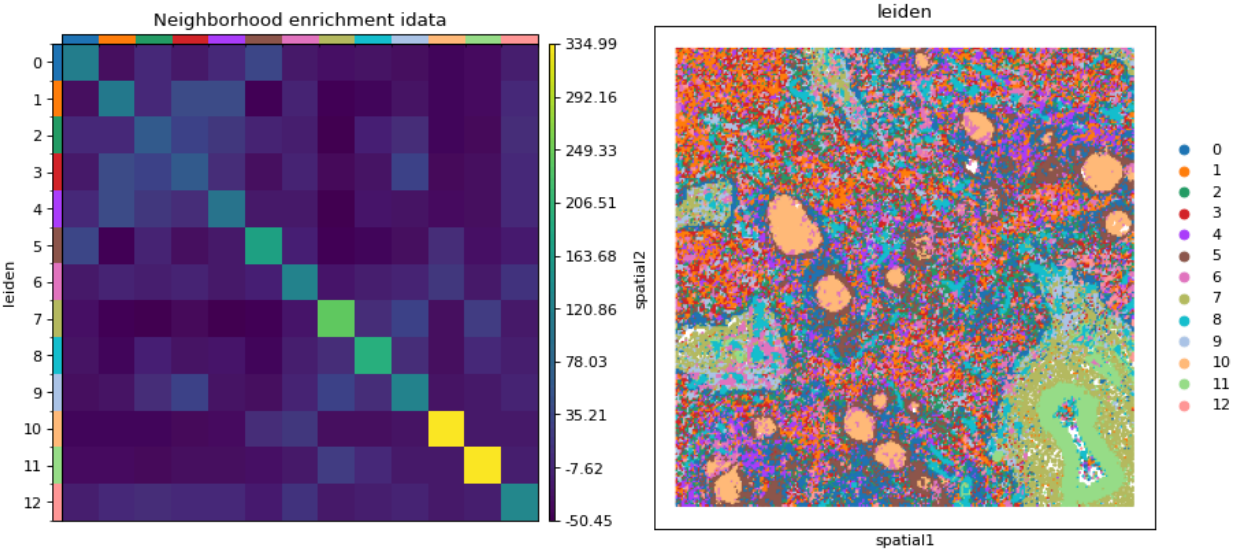


In [52]:
```
sc.pl.umap(
    idata,
    color=["leiden"],
)
```



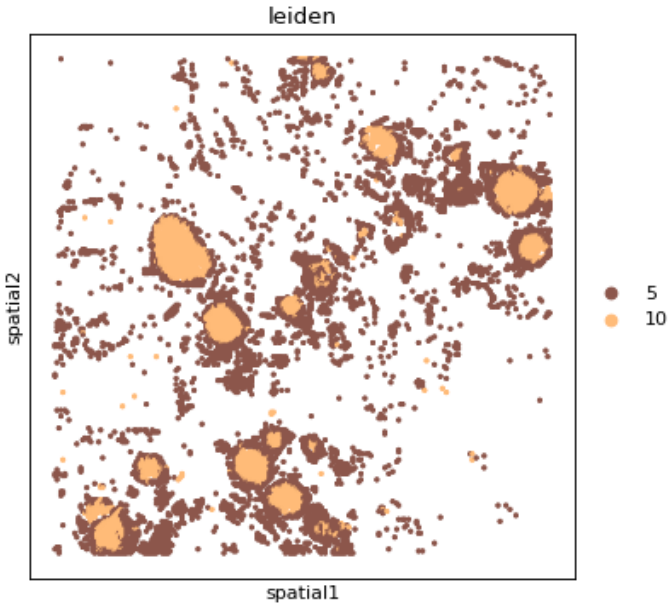
In [53]:
```
sq.gr.spatial_neighbors(idata, coord_type="generic", delaunay=True)
sq.gr.nhood_enrichment(idata, cluster_key="leiden")
fig, ax = plt.subplots(1, 2, figsize=(13, 7))

sq.pl.nhood_enrichment(
    idata,
    cluster_key="leiden",
    library_id="spatial",
    figsize=(8, 8),
    title="Neighborhood enrichment idata",
    ax=ax[0],
)
sq.pl.spatial_scatter(idata, color="leiden", shape=None, size=2, ax=ax[1])
```

```
  0%|          | 0/1000 [00:00<?, ?/s]
WARNING: Please specify a valid `library_id` or set it permanently in `adata.uns['spatial']`
```

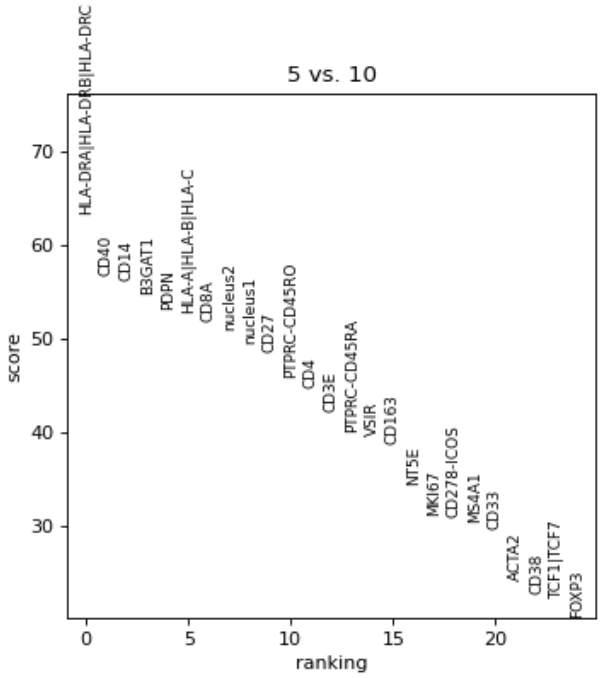


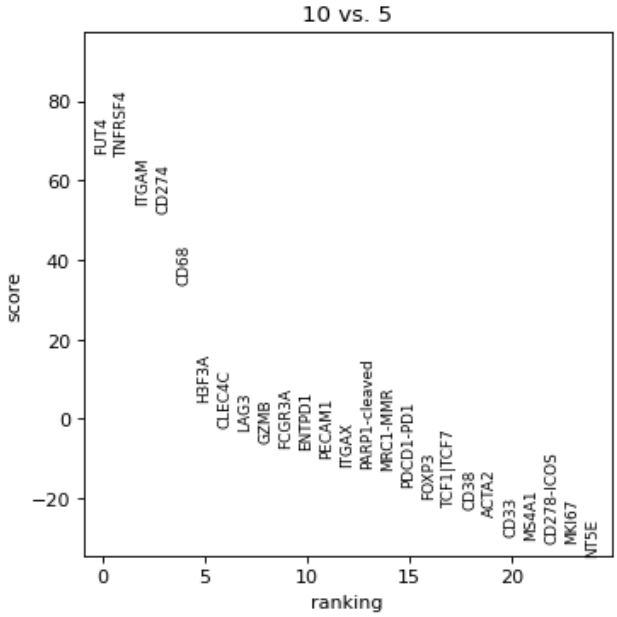
In [54]:
```
# Plot specific leiden clusters
iclustersoi = ["10", "5"]

sq.pl.spatial_scatter(
    idata[idata.obs["leiden"].isin(iclustersoi)],
    library_id="spatial",
    shape=None,
    color=["leiden"],
```

```
    size=12,
)
```

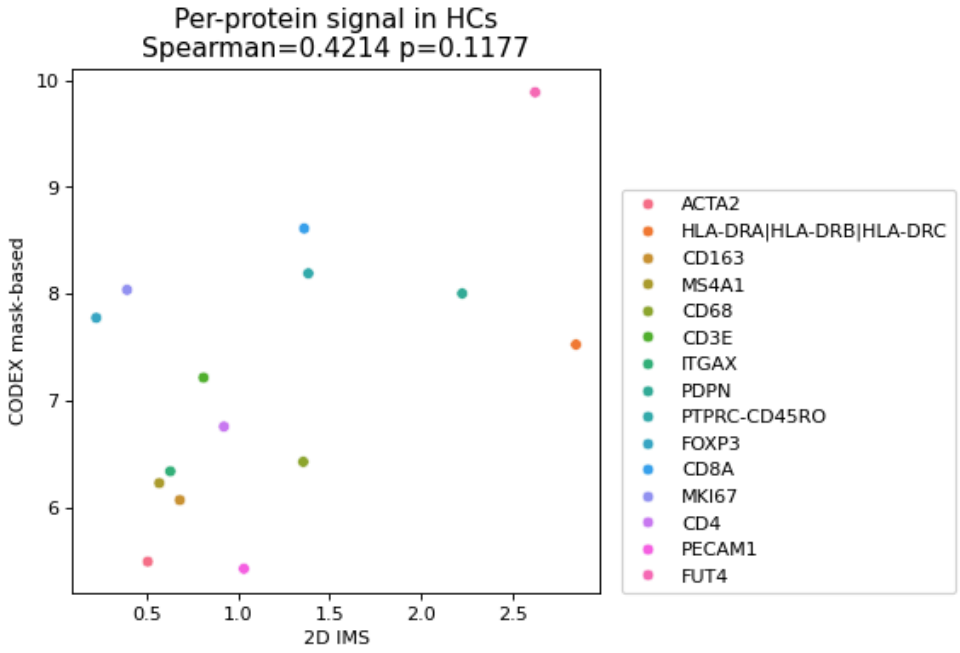


In [55]:
```
# Identify dfferentally expressed genes for specific clusters against a reference
csets = [iclustersoi]

for pair in csets:
    sc.tl.rank_genes_groups(
        idata, "leiden", method="wilcoxon", groups=[pair[1]], reference=pair[0]
    )
    sc.pl.rank_genes_groups(idata, groups=[pair[1]], n_genes=25)

    sc.tl.rank_genes_groups(
        idata, "leiden", method="wilcoxon", groups=[pair[0]], reference=pair[1]
    )
    sc.pl.rank_genes_groups(idata, groups=[pair[0]], n_genes=25)
```

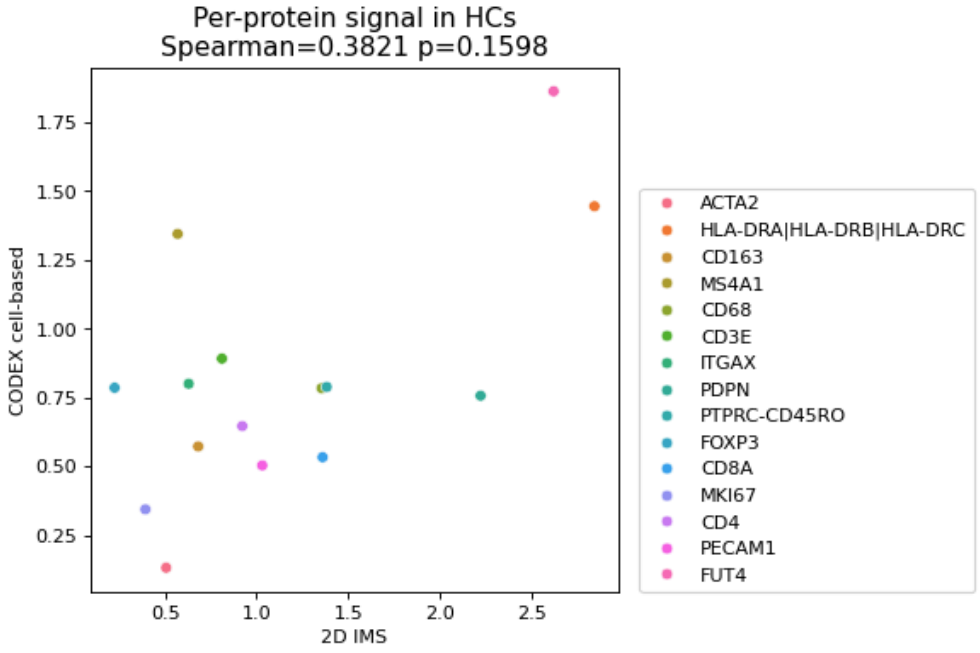




In [56]:
```
# Pseudobulk per-gene comparison normalized by number of ids (structures or cells)
x = idata[idata.obs["leiden"].isin(iclustersoi)].to_df().sum() / len(
    idata[idata.obs["leiden"].isin(iclustersoi)]
)
y = cad.to_df().sum() / len(cad)
ic = pd.concat([x, y], axis=1)
ic.columns = ["2D IMS", "CODEX mask-based"]
icf = ic.dropna()

srs, srsp = scipy.stats.spearmanr(
    icf["2D IMS"], icf["CODEX mask-based"], nan_policy="omit"
)
srs = round(srs, 4)
srsp = round(srsp, 4)

sns.scatterplot(x="2D IMS", y="CODEX mask-based", data=icf, hue=icf.index)
plt.title(f"Per-protein signal in HCs\nSpearman={srs} p={srsp}", size=14)
plt.legend(loc=(1.04, 0))
plt.show()
```



In [57]:
```
# Pseudobulk per-gene comparison normalized by number of ids (structures or cells)
# x = idata[idata.obs['leiden']=='11'].to_df().sum()/len(idata[idata.obs['leiden']=='11'])
x = idata[idata.obs["leiden"].isin(iclustersoi)].to_df().sum() / len(
    idata[idata.obs["leiden"].isin(iclustersoi)]
)
y = cad2.to_df().sum() / len(cad2)
ic = pd.concat([x, y], axis=1)
ic.columns = ["2D IMS", "CODEX cell-based"]
icf = ic.dropna()

srs, srsp = scipy.stats.spearmanr(
    icf["2D IMS"], icf["CODEX cell-based"], nan_policy="omit"
)
srs = round(srs, 4)
srsp = round(srsp, 4)

sns.scatterplot(x="2D IMS", y="CODEX cell-based", data=icf, hue=icf.index)
plt.title(f"Per-protein signal in HCs\nSpearman={srs} p={srsp}", size=14)
plt.legend(loc=(1.04, 0))
plt.show()
```



The masked-based CODEX HC data corresponds slightly better to the 2D IHC data compared to the cell-based CODEX data for the 15 common markers. Keep in mind the patient sample is different for the 2D IHC sample than the CODEX samples.

[3.3] Pseudobulk comparisons: CODEX, 2D IMC and scRNA-Seq

CZI cellxgene hosts single-cell RNA-Seq cell atlas data for human thymus. The study

> A cell atlas of human thymic development defines T cell repertoire formation

offers three data sets.

In [63]:
```
cziurls = {
    "czifull": "c6e08ab6-ab3b-41dc-8058-8e6442e081ec.h5ad",  # 255,901 cells
    "czistem": "59d5b3c5-9a55-44ae-a7fa-c14567e02755.h5ad",  # 27,589 cells
    "czitecs": "5401b83e-2c9b-40a0-b20e-fd9a58c69d92.h5ad",  # 18,524 cells
}
```

```
cgdir = f"{ddir}cellxgene/"
ctlabel = "cell_type"
```

In [64]:
```
for url in cziurls.values():
    !wget --directory-prefix={cgdir} https://datasets.cellxgene.cziscience.com/{url}
```

```
--2024-11-14 18:34:58--  https://datasets.cellxgene.cziscience.com/c6e08ab6-ab3b-41dc-8058-8e
6442e081ec.h5ad
Resolving datasets.cellxgene.cziscience.com (datasets.cellxgene.cziscience.com)... 13.32.151.
108, 13.32.151.10, 13.32.151.37, ...
Connecting to datasets.cellxgene.cziscience.com (datasets.cellxgene.cziscience.com)|13.32.15
1.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2633195386 (2.5G) [binary/octet-stream]
Saving to: ‘datasets/cellxgene/c6e08ab6-ab3b-41dc-8058-8e6442e081ec.h5ad.1’

c6e08ab6-ab3b-41dc- 100%[===================>]   2.45G   110MB/s    in 23s

2024-11-14 18:35:21 (109 MB/s) - ‘datasets/cellxgene/c6e08ab6-ab3b-41dc-8058-8e6442e081ec.h5a
d.1’ saved [2633195386/2633195386]

--2024-11-14 18:35:21--  https://datasets.cellxgene.cziscience.com/59d5b3c5-9a55-44ae-a7fa-c1
4567e02755.h5ad
Resolving datasets.cellxgene.cziscience.com (datasets.cellxgene.cziscience.com)... 13.32.151.
10, 13.32.151.37, 13.32.151.100, ...
Connecting to datasets.cellxgene.cziscience.com (datasets.cellxgene.cziscience.com)|13.32.15
1.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 458816316 (438M) [binary/octet-stream]
Saving to: ‘datasets/cellxgene/59d5b3c5-9a55-44ae-a7fa-c14567e02755.h5ad.1’

59d5b3c5-9a55-44ae- 100%[===================>] 437.56M   112MB/s    in 4.0s

2024-11-14 18:35:25 (110 MB/s) - ‘datasets/cellxgene/59d5b3c5-9a55-44ae-a7fa-c14567e02755.h5a
d.1’ saved [458816316/458816316]

--2024-11-14 18:35:25--  https://datasets.cellxgene.cziscience.com/5401b83e-2c9b-40a0-b20e-fd
9a58c69d92.h5ad
Resolving datasets.cellxgene.cziscience.com (datasets.cellxgene.cziscience.com)... 13.32.151.
10, 13.32.151.37, 13.32.151.100, ...
Connecting to datasets.cellxgene.cziscience.com (datasets.cellxgene.cziscience.com)|13.32.15
1.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 265561295 (253M) [binary/octet-stream]
Saving to: ‘datasets/cellxgene/5401b83e-2c9b-40a0-b20e-fd9a58c69d92.h5ad.1’

5401b83e-2c9b-40a0- 100%[===================>] 253.26M   109MB/s    in 2.3s

2024-11-14 18:35:28 (109 MB/s) - ‘datasets/cellxgene/5401b83e-2c9b-40a0-b20e-fd9a58c69d92.h5a
d.1’ saved [265561295/265561295]
```

In [58]:
```
def anndataToPseudobulk(
    url=str,
    slabel=None,
    flabel="feature_name",
    clabel="cell_type",
):
    adata = ad.read_h5ad(url)
    adata.var.index = adata.var[flabel]
    cts = set(adata.obs[clabel])
    bulk = []
    for ct in cts:
        sub = adata[adata.obs[adata.obs[clabel] == ct].index].to_df()
        sub2 = pd.DataFrame(sub.sum() / len(sub))
        sub2.columns = [ct]
```

```
        bulk.append(sub2)

    return bulk
```

In [66]:
```
# czibulkraw = {}
# czibulkraw['czitecs'] = ad.read_h5ad('datasets/cellxgene/5401b83e-2c9b-40a0-b20e-fd9a58c69
# czibulkraw['czistem'] = ad.read_h5ad('datasets/cellxgene/59d5b3c5-9a55-44ae-a7fa-c14567e02
# czibulkraw['czifull'] = ad.read_h5ad('datasets/cellxgene/c6e08ab6-ab3b-41dc-8058-8e6442e08

# czibulk = {}
# czibulk['czitecs'] = anndataToPseudobulk(czibulkraw['czitecs'])
# czibulk['czistem'] = anndataToPseudobulk(czibulkraw['czistem'])
# czibulk['czifull'] = anndataToPseudobulk(czibulkraw['czifull'])
```

In [59]:
```
cziurls = {
    "czifull": "c6e08ab6-ab3b-41dc-8058-8e6442e081ec.h5ad",
    "czistem": "59d5b3c5-9a55-44ae-a7fa-c14567e02755.h5ad",
    "czitecs": "5401b83e-2c9b-40a0-b20e-fd9a58c69d92.h5ad",
}

# Read in AnnData objects and convert to pseudobulk counts for each cell_type label
czibulk = {}

# This works on my laptop but in HuBMAP Workspaces gives an 'unable to sychronously' error.
# Reading in individually kills the kernel. I think these may be too unweildy. More cores an
for name, url in cziurls.items():
    print(f"{ddir}cellxgene/{url}")
    pseudo = anndataToPseudobulk(f"{ddir}cellxgene/{url}")
    czibulk[name] = pseudo
```

```
datasets/cellxgene/c6e08ab6-ab3b-41dc-8058-8e6442e081ec.h5ad
datasets/cellxgene/59d5b3c5-9a55-44ae-a7fa-c14567e02755.h5ad
datasets/cellxgene/5401b83e-2c9b-40a0-b20e-fd9a58c69d92.h5ad
```

In [60]:
```
# Wrangle gene names from the CODEX and IMC data and find common genes to CZI
onames = set()
nameslist = list(channames["name"]) + list(codexgenemap["codex"])

for cname in nameslist:
    if cname == "na":
        continue
    elif str("|") in str(cname):
        mnames = cname.split("|")
        for mname in mnames:
            onames.add(mname)
    elif str("-") in str(cname):
        mnames = cname.split("-")
        for mname in mnames:
            onames.add(mname)
    else:
        onames.add(cname)

commongenes = list(set(list(czibulk["czitecs"][0].index)).intersection(onames))
len(commongenes)
```

Out[60]: 53

In [61]:
```
# Read in previously saved anndata objects
cad = ad.read_h5ad("cad.h5ad")
cad2 = ad.read_h5ad("cad2.h5ad")
idata = ad.read_h5ad("idata.h5ad")
```

In [62]:
```
lclustersoi = ["2", "5", "10", "11", "14", "16"]
iclustersoi = ["9", "6"]
```

In [63]:
```
x = cad.to_df().sum() / len(cad)
y = cad2[cad2.obs["leiden"].isin(lclustersoi)].to_df().sum() / len(
    cad2[cad2.obs["leiden"].isin(lclustersoi)]
)
z = idata[idata.obs["leiden"].isin(iclustersoi)].to_df().sum() / len(
    idata[idata.obs["leiden"].isin(iclustersoi)]
)

xyz = pd.DataFrame(pd.concat([x, y, z], axis=1))
xyz.columns = ["CODEX mask-based", "CODEX cell-based", "2D IMC"]
```

In [64]:
```
pd.concat(czibulk["czifull"], axis=1)
```

Out[64]:

feature_name	natural killer cell	monocyte	fibroblast	naive B cell	medullary thymic epithelial cell	dendritic cell	erythrocyte	
TSPAN6	0.021578	0.000000	0.719379	0.047346	0.869268	0.069640	0.214231	0.12
TNMD	0.001432	0.000000	0.045375	0.000000	0.008097	0.000000	0.006956	0.00
DPM1	0.514863	0.523218	0.613272	0.401953	0.535895	0.769474	1.000435	0.52
SCYL3	0.162699	0.048003	0.110275	0.159971	0.123444	0.110124	0.122967	0.1
C1orf112	0.086081	0.041251	0.079711	0.114051	0.079478	0.151801	0.294090	0.10
...	...	...	...	...	...	...	...	
ENSG00000283096.1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
ENSG00000283103.5	0.416208	0.265153	0.384515	0.206016	0.425892	0.272247	0.613991	0.3
MGC4859	0.000000	0.000000	0.001824	0.000000	0.009011	0.002347	0.000000	0.00
ENSG00000283118.1	0.000000	0.000000	0.000713	0.000000	0.000000	0.000000	0.000000	0.00
ENSG00000283125.1	0.001432	0.000000	0.001775	0.003539	0.000982	0.002209	0.002747	0.0

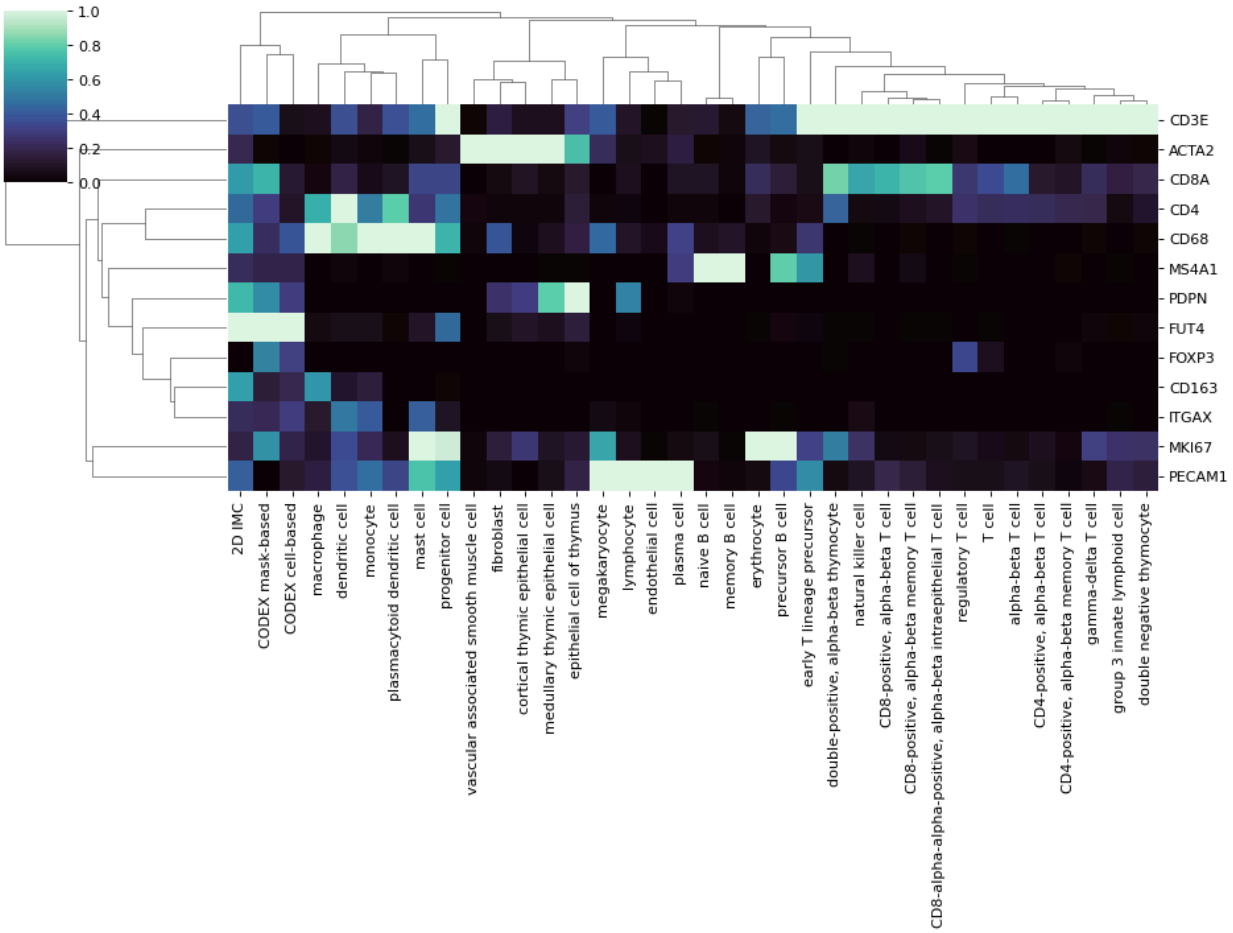
32839 rows × 33 columns

In [65]:
```
commondfs = {}

for name, mylist in czibulk.items():
    df = pd.concat(mylist, axis=1)
    czisub = df.loc[commongenes]
    czisub = czisub.merge(xyz, how="outer", left_index=True, right_index=True)

    commondfs[name] = czisub

print(commondfs.keys())
```

```
dict_keys(['czifull', 'czistem', 'czitecs'])
```

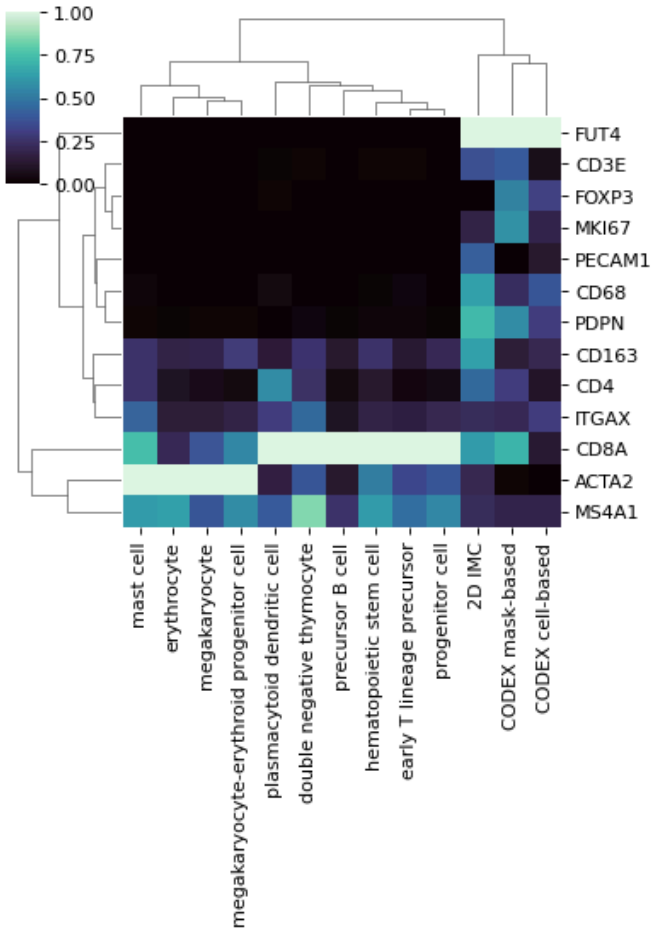
In [66]:
```
sns.clustermap(
    commondfs["czifull"].dropna(),
    standard_scale=1,
    cmap="mako",
    figsize=(12, 9),
    row_cluster=True,
    col_cluster=True,
)
```

Out[66]: <seaborn.matrix.ClusterGrid at 0x14d4415b9100>



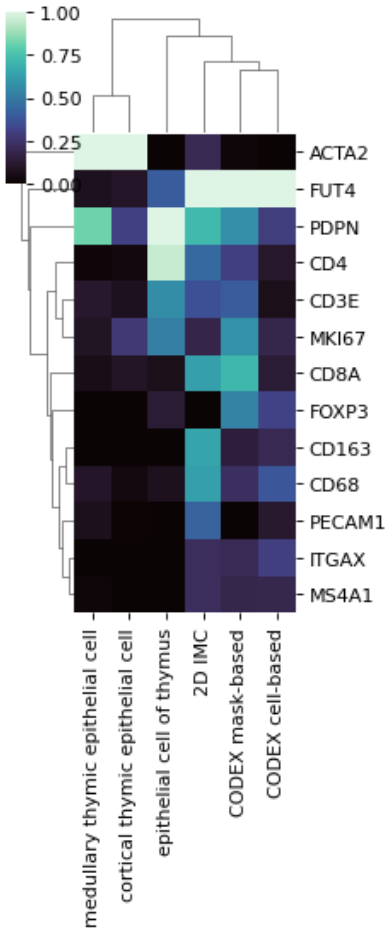
In [67]:
```
sns.clustermap(
    commondfs["czistem"].dropna(),
    standard_scale=1,
    cmap="mako",
    figsize=(5, 7),
    row_cluster=True,
    col_cluster=True,
)
```

Out[67]: <seaborn.matrix.ClusterGrid at 0x14d4731f8790>



In [68]:
```
sns.clustermap(
    commondfs["czitecs"].dropna(),
    standard_scale=1,
    cmap="mako",
    figsize=(3, 7),
    row_cluster=True,
    col_cluster=True,
)
```

Out[68]: <seaborn.matrix.ClusterGrid at 0x14d440c56fa0>



Literature states HC cells are akin to medullary TECs. RNA expression does not always correlate with protein expression and technology differences can make data noncomparable.