multiple-labels

October 19, 2023

1 Ultrack I2K 2023 - Multiple hypotheses tracking

This tutorial shows the multiple hypotheses tracking capabilities of Ultrack.

Here, rather than searching for an optimal segmentation parameter, we sampled multiple segmentations with different parametrizations and used Ultrack to find the best segments, obtaining more accurate cell tracking.

1.1 Setting up Colab runtime

If you are using Colab, we recommend to set up the runtime to use a GPU. To do so, go to `Runtime > Change runtime type` and select `GPU` as the hardware accelerator.

1.2 Setup Dependencies

This step is only necessary if you are on Colab or don't have the required packages.

IMPORTANT: The runtime must be initialized.

Uncomment and run the following commands to install all required packages.

[1]:
```
# !pip install stackview cellpose 'napari[all]' ultrack ipycanvas==0.11 cucim
# !pip install git+https://github.com/Janelia-Trackathon-2023/traccuracy
```

1.3 Download Dataset

Download the Fluo-C2DL-Huh7 dataset from the Cell Tracking Challenge, which contains fluorescence microscopy images for cell tracking.

The dataset will be used for demonstrating the segmentation and tracking workflow.

[2]:
```
!wget -nc http://data.celltrackingchallenge.net/training-datasets/
  ↪Fluo-C2DL-Huh7.zip
!unzip -n Fluo-C2DL-Huh7.zip
```

```
File 'Fluo-C2DL-Huh7.zip' already there; not retrieving.

Archive:  Fluo-C2DL-Huh7.zip
```

1.4 Import Libraries

Import the libraries needed for reading images, processing them, cell segmentation, tracking, and performance metrics.

[3]:
```
from pathlib import Path
from typing import Dict

import pandas as pd
import numpy as np
import stackview
from dask.array.image import imread
from numpy.typing import ArrayLike
from rich import print

from traccuracy import run_metrics
from traccuracy.loaders import load_ctc_data
from traccuracy.matchers import CTCMatched
from traccuracy.metrics import CTCMetrics

from ultrack import track, to_tracks_layer, tracks_to_zarr, to_ctc
from ultrack.utils import labels_to_edges
from ultrack.config import MainConfig
from ultrack.imgproc import normalize
from ultrack.imgproc.segmentation import Cellpose
from ultrack.utils.array import array_apply
```

1.5 Colab or Local

Change the `COLAB` variable to `True` or `False` depending on whether you are running this notebook on Colab or locally.

When running locally napari will be used a the image viewer, while on Colab the images will be displayed using `stackview`.

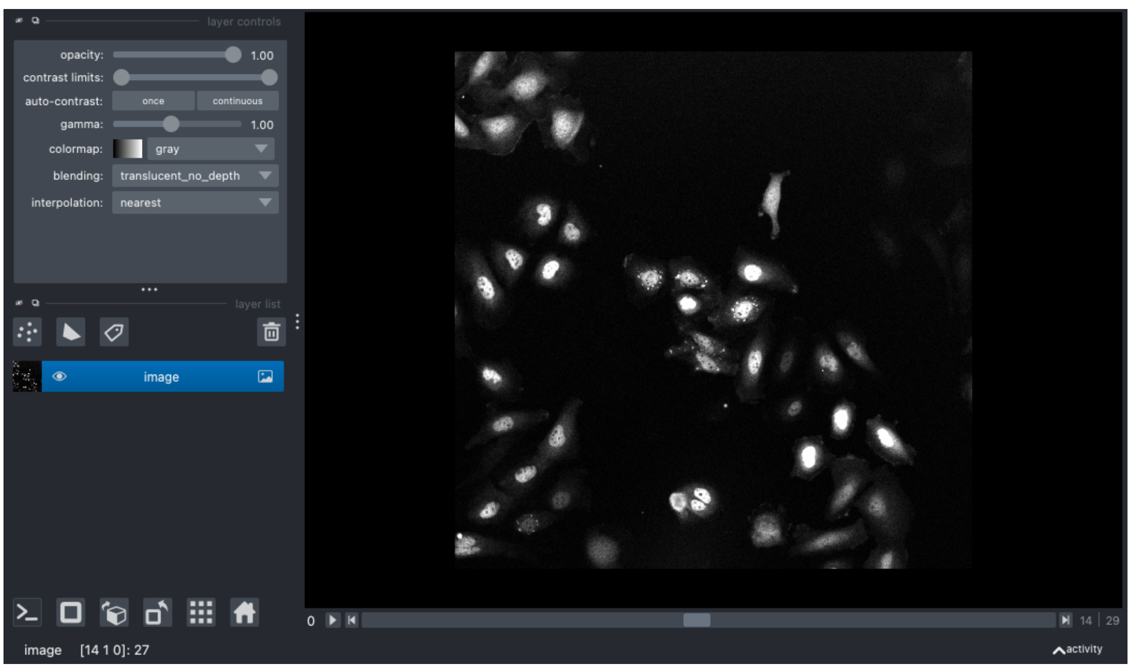
[4]:
```
# COLAB = True
COLAB = False

if COLAB:
    viewer = None

    # fixes colab encoding error
    import locale
    locale.getpreferredencoding = lambda: "UTF-8"

    # enabling colab output
    try:
        from google.colab import output
        output.enable_custom_widget_manager()
```

```
    except ModuleNotFoundError as e:
        print(e)
else:
    import napari
    from napari.utils import nbscreenshot

    viewer = napari.Viewer()

    def screenshot() -> None:
        display(nbscreenshot(viewer))
```

1.6 Load Data

Load the Fluo-C2DL-Huh7 dataset.

[5]:
```
dataset = "02"
path = Path("Fluo-C2DL-Huh7") / dataset
image = imread(str(path / "*.tif"))

if COLAB:
    display(stackview.slice(image))
else:
    viewer.add_image(image)
    screenshot()
```



1.7 Configuration

We'll use the same configuration as in the previous example, except for `config.segmentation_config.min_frontier` which had its value decreased.

The `min_frontier` merges regions with an average contour lower than the provided value. Since the contours are combined by averaging, the previous value of 0.1 removed relevant segments from the candidate hypotheses.

As a reminder, the configuration parameters documentation can be found here.

[6]:
```python
config = MainConfig()

# Candidate segmentation parameters
config.segmentation_config.n_workers = 8
config.segmentation_config.min_area = 2500
config.segmentation_config.min_frontier = 0.05  # NOTE: this parameter is not␣
↪the same as in intro.ipynb

# Setting the maximum number of candidate neighbors and maximum spatial␣
↪distance between cells
config.linking_config.max_neighbors = 5
config.linking_config.max_distance = 100
config.linking_config.n_workers = 8

# Adding absurd weight to division because there's no diving cell
config.tracking_config.division_weight = -100
# Very few tracks enter/leave the field of view, increasing penalization
config.tracking_config.disappear_weight = -1
config.tracking_config.appear_weight = -1

print(config)
```

```
MainConfig(
    data_config=DataConfig(working_dir=PosixPath('.'), database='sqlite',␣
↪address=None, n_workers=1),
    segmentation_config=SegmentationConfig(
        threshold=0.5,
        min_area=2500,
        max_area=1000000,
        min_frontier=0.05,
        anisotropy_penalization=0.0,
        max_noise=0.0,
        ws_hierarchy=<function watershed_hierarchy_by_area at 0x15e1a4360>,
        n_workers=8
    ),
    linking_config=LinkingConfig(
        n_workers=8,
        max_neighbors=5,
```

```
        max_distance=100,
        distance_weight=0.0,
        z_score_threshold=5.0
    ),
    tracking_config=TrackingConfig(
        appear_weight=-1,
        disappear_weight=-1,
        division_weight=-100,
        dismiss_weight_guess=None,
        include_weight_guess=None,
        window_size=None,
        overlap_size=1,
        solution_gap=0.001,
        time_limit=36000,
        method=0,
        n_threads=-1,
        link_function='power',
        power=4,
        bias=-0.0
    )
)
```

1.8 Cellpose Segmentation

The same function as `intro.ipynb` to segment cells within each frame.

[7]:
```
cellpose = Cellpose(model_type="cyto2", gpu=True)

def predict(frame: ArrayLike, gamma: float) -> ArrayLike:
    norm_frame = normalize(np.asarray(frame), gamma=gamma)
    return cellpose(norm_frame, tile=False, normalize=False, diameter=75.0)
```

1.9 Metrics

Helper function to evaluate tracking score using Cell Tracking Challenge's metrics and annotations.

[8]:
```
gt_path = path.parent / f"{dataset}_GT"
gt_data = load_ctc_data(gt_path / "TRA")

def score(output_path: Path) -> Dict:
    return run_metrics(
        gt_data=gt_data,
        pred_data=load_ctc_data(output_path),
        matcher=CTCMatched,
        metrics=[CTCMetrics],
    )["CTCMetrics"]
```

```
Loading TIFFs: 100%|                    | 30/30 [00:00<00:00,
500.16it/s]
```

1.10 Parameter Search

Here, we evaluate the segmentation and tracking given multiple values of `gamma`, used on the normalization step before the Cellpose prediction.

[9]:
```
all_labels = []
metrics = []
gammas = [0.1, 0.25, 0.5, 1]
sigma = 5.0

for gamma in gammas:

    # cellpose prediction
    cellpose_labels = np.zeros_like(image, dtype=np.int32)
    array_apply(
        image,
        out_array=cellpose_labels,
        func=predict,
        gamma=gamma,
    )
    all_labels.append(cellpose_labels)

    name = f"{dataset}_labels_{gamma}"
    if not COLAB:
        viewer.add_labels(cellpose_labels, name=name, visible=False)

    # cell tracking using `labels` parameter, it's the same as using␣
 ↪`labels_to_edges`.
    track(
        config,
        labels=cellpose_labels,
        sigma=sigma,
        overwrite=True
    )

    # exporting to CTC format
    output_path = Path(name.upper()) / "TRA"
    to_ctc(output_path, config, overwrite=True)

    # computing tracking score
    metric = score(output_path)
    metric["gamma"] = gamma
    metrics.append(metric)

print(metrics)
```

```
Applying predict …: 100%|              | 30/30 [02:32<00:00,
5.09s/it]
Converting labels to edges: 100%|           | 30/30 [00:01<00:00,
15.70it/s]
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
Adding nodes to database: 100%|            | 30/30 [00:06<00:00,
4.97it/s]
Linking nodes.: 100%|                  | 29/29 [00:03<00:00,
8.70it/s]
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2024-08-17
Using GRB solver
Solving ILP batch 0
Constructing ILP …
Set parameter TimeLimit to value 36000
Solving ILP …
Set parameter NodeLimit to value 1073741824
Set parameter SolutionLimit to value 1073741824
Set parameter IntFeasTol to value 1e-06
Set parameter Method to value 3
Set parameter MIPGap to value 0.001
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 8824 rows, 14838 columns and 33513 nonzeros
Model fingerprint: 0x28978f57
Variable types: 0 continuous, 14838 integer (14838 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e-19, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -0.0000000
Presolve removed 5366 rows and 7735 columns
Presolve time: 0.06s
Presolved: 3458 rows, 7103 columns, 15984 nonzeros
Found heuristic solution: objective 271.6120410
Variable types: 0 continuous, 7103 integer (7103 binary)
Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only…

Root barrier log…
```

```
Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 1.357e+04
 Factor NZ  : 7.968e+04 (roughly 5 MB of memory)
 Factor Ops : 2.393e+06 (less than 1 second per iteration)
 Threads    : 8

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -2.19437633e+05  4.89972790e+05  1.87e+01 3.58e+01  1.69e+02     0s
   1  -7.91809998e+04  2.89102971e+05  7.28e+00 1.47e+00  5.80e+01     0s

Barrier performed 1 iterations in 0.07 seconds (0.08 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex

Root relaxation: objective 5.369321e+02, 1515 iterations, 0.01 seconds (0.02
work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0     536.9321057  536.93211  0.00%     -    0s

Explored 1 nodes (1515 simplex iterations) in 0.08 seconds (0.08 work units)
Thread count was 10 (of 10 available processors)

Solution count 3: 536.932 271.612 -0

Optimal solution found (tolerance 1.00e-03)
Best objective 5.369321057463e+02, best bound 5.369321057463e+02, gap 0.0000%
Saving solution …
Done!
Exporting segmentation masks: 100%|          | 30/30 [00:00<00:00,
122.68it/s]
Loading TIFFs: 100%|                 | 30/30 [00:00<00:00,
878.16it/s]
Matching frames: 100%|                 | 30/30 [00:00<00:00,
125.64it/s]
Evaluating nodes: 100%|           | 1138/1138 [00:00<00:00,
1081849.04it/s]
Evaluating FP edges: 100%|           | 1008/1008 [00:00<00:00,
1282.57it/s]
```

```
Evaluating FN edges: 100%|           | 1563/1563 [00:00<00:00,
3225.37it/s]
Applying predict …: 100%|              | 30/30 [02:32<00:00,
5.09s/it]
Converting labels to edges: 100%|           | 30/30 [00:01<00:00,
16.09it/s]
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
Adding nodes to database: 100%|            | 30/30 [00:06<00:00,
4.85it/s]
Linking nodes.: 100%|                 | 29/29 [00:03<00:00,
8.51it/s]

Set parameter Username
Academic license - for non-commercial use only - expires 2024-08-17
Using GRB solver
Solving ILP batch 0
Constructing ILP …
Set parameter TimeLimit to value 36000
Solving ILP …
Set parameter NodeLimit to value 1073741824
Set parameter SolutionLimit to value 1073741824
Set parameter IntFeasTol to value 1e-06
Set parameter Method to value 3
Set parameter MIPGap to value 0.001
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 11316 rows, 18294 columns and 42777 nonzeros
Model fingerprint: 0x152eb52f
Variable types: 0 continuous, 18294 integer (18294 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [3e-19, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -0.0000000
Presolve removed 7601 rows and 10446 columns
Presolve time: 0.07s
Presolved: 3715 rows, 7848 columns, 17868 nonzeros
Found heuristic solution: objective 321.5799346
Variable types: 0 continuous, 7848 integer (7848 binary)
Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only…
```

```
Root barrier log…

Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 1.610e+04
 Factor NZ  : 9.810e+04 (roughly 5 MB of memory)
 Factor Ops : 3.626e+06 (less than 1 second per iteration)
 Threads    : 8

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -2.39232786e+05  5.24545060e+05  1.90e+01 3.38e+01  1.67e+02     0s
   1  -7.83731748e+04  3.03734541e+05  7.02e+00 1.79e-01  5.43e+01     0s

Barrier performed 1 iterations in 0.09 seconds (0.10 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex

Root relaxation: objective 6.167215e+02, 1659 iterations, 0.01 seconds (0.02
work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0     616.7215379  616.72154  0.00%     -    0s

Explored 1 nodes (1659 simplex iterations) in 0.10 seconds (0.10 work units)
Thread count was 10 (of 10 available processors)

Solution count 3: 616.722 321.58 -0

Optimal solution found (tolerance 1.00e-03)
Best objective 6.167215378622e+02, best bound 6.167215378622e+02, gap 0.0000%
Saving solution …
Done!

Exporting segmentation masks: 100%|          | 30/30 [00:00<00:00,
115.09it/s]
Loading TIFFs: 100%|                  | 30/30 [00:00<00:00,
865.35it/s]
Matching frames: 100%|                | 30/30 [00:00<00:00,
122.13it/s]
Evaluating nodes: 100%|            | 1265/1265 [00:00<00:00,
1095333.31it/s]
```

```
Evaluating FP edges: 100%|          | 1143/1143 [00:00<00:00,
1252.14it/s]
Evaluating FN edges: 100%|          | 1563/1563 [00:00<00:00,
3142.51it/s]
Applying predict …: 100%|          | 30/30 [02:31<00:00,
5.06s/it]
Converting labels to edges: 100%|          | 30/30 [00:01<00:00,
15.95it/s]
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
Adding nodes to database: 100%|          | 30/30 [00:06<00:00,
4.75it/s]
Linking nodes.: 100%|          | 29/29 [00:03<00:00,
8.43it/s]
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2024-08-17
Using GRB solver
Solving ILP batch 0
Constructing ILP …
Set parameter TimeLimit to value 36000
Solving ILP …
Set parameter NodeLimit to value 1073741824
Set parameter SolutionLimit to value 1073741824
Set parameter IntFeasTol to value 1e-06
Set parameter Method to value 3
Set parameter MIPGap to value 0.001
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 11485 rows, 19050 columns and 44011 nonzeros
Model fingerprint: 0x889609ee
Variable types: 0 continuous, 19050 integer (19050 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [9e-18, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -0.0000000
Presolve removed 7776 rows and 11141 columns
Presolve time: 0.07s
Presolved: 3709 rows, 7909 columns, 17895 nonzeros
Found heuristic solution: objective 386.8293721
Variable types: 0 continuous, 7909 integer (7909 binary)
```

```
Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only…

Root barrier log…

Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 1.564e+04
 Factor NZ  : 9.386e+04 (roughly 5 MB of memory)
 Factor Ops : 3.135e+06 (less than 1 second per iteration)
 Threads    : 8

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -2.16246253e+05  5.19500566e+05  1.96e+01 3.24e+01  1.47e+02     0s
   1  -6.52383470e+04  2.56382204e+05  6.48e+00 2.26e+00  4.19e+01     0s
   2  -8.54440561e+03  5.72268000e+04  1.57e-01 8.53e-14  3.91e+00     0s

Barrier performed 2 iterations in 0.09 seconds (0.10 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex

Root relaxation: objective 6.924227e+02, 1689 iterations, 0.01 seconds (0.02
work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  692.42269    0   34  386.82937  692.42269  79.0%     -    0s
H    0     0                    690.7152383  692.42269  0.25%     -    0s
*    0     0               0    692.2988120  692.29881  0.00%     -    0s

Cutting planes:
  Zero half: 1

Explored 1 nodes (1754 simplex iterations) in 0.11 seconds (0.12 work units)
Thread count was 10 (of 10 available processors)

Solution count 4: 692.299 690.715 386.829 -0

Optimal solution found (tolerance 1.00e-03)
Best objective 6.922988119602e+02, best bound 6.922988119602e+02, gap 0.0000%
Saving solution …
Done!
```

```
Exporting segmentation masks: 100%|          | 30/30 [00:00<00:00,
112.53it/s]
Loading TIFFs: 100%|                  | 30/30 [00:00<00:00,
839.98it/s]
Matching frames: 100%|                 | 30/30 [00:00<00:00,
119.06it/s]
Evaluating nodes: 100%|            | 1323/1323 [00:00<00:00,
1114717.60it/s]
Evaluating FP edges: 100%|            | 1203/1203 [00:00<00:00,
1212.68it/s]
Evaluating FN edges: 100%|            | 1563/1563 [00:00<00:00,
3056.69it/s]
Applying predict …: 100%|              | 30/30 [02:02<00:00,
4.07s/it]
Converting labels to edges: 100%|           | 30/30 [00:01<00:00,
15.94it/s]
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
Adding nodes to database: 100%|            | 30/30 [00:05<00:00,
5.64it/s]
Linking nodes.: 100%|                  | 29/29 [00:03<00:00,
8.60it/s]

Set parameter Username
Academic license - for non-commercial use only - expires 2024-08-17
Using GRB solver
Solving ILP batch 0
Constructing ILP …
Set parameter TimeLimit to value 36000
Solving ILP …
Set parameter NodeLimit to value 1073741824
Set parameter SolutionLimit to value 1073741824
Set parameter IntFeasTol to value 1e-06
Set parameter Method to value 3
Set parameter MIPGap to value 0.001
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 5654 rows, 10695 columns and 22142 nonzeros
Model fingerprint: 0x3a9963a7
Variable types: 0 continuous, 10695 integer (10695 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [8e-18, 1e+02]
```

```
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -0.0000000
Presolve removed 4402 rows and 8160 columns
Presolve time: 0.04s
Presolved: 1252 rows, 2535 columns, 5515 nonzeros
Found heuristic solution: objective 414.6247120
Variable types: 0 continuous, 2535 integer (2535 binary)
Found heuristic solution: objective 415.3483989
Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only…

Root barrier log…

Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 4.075e+03
 Factor NZ  : 2.089e+04 (roughly 2 MB of memory)
 Factor Ops : 3.911e+05 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual          Primal    Dual     Compl     Time
   0  -6.79909688e+04  1.63211754e+05  1.32e+01 3.05e+01  1.30e+02     0s
   1  -1.21903102e+04  7.59184169e+04  2.36e+00 7.11e-14  2.51e+01     0s

Barrier performed 1 iterations in 0.05 seconds (0.05 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex

Root relaxation: objective 5.507780e+02, 524 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0     550.7780352  550.77804  0.00%     -    0s

Explored 1 nodes (524 simplex iterations) in 0.05 seconds (0.05 work units)
Thread count was 10 (of 10 available processors)

Solution count 4: 550.778 415.348 414.625 -0

Optimal solution found (tolerance 1.00e-03)
Best objective 5.507780352111e+02, best bound 5.507780352111e+02, gap 0.0000%
```

```
Saving solution …
Done!

Exporting segmentation masks: 100%|          | 30/30 [00:00<00:00,
126.41it/s]
Loading TIFFs: 100%|                  | 30/30 [00:00<00:00,
841.73it/s]
Matching frames: 100%|                | 30/30 [00:00<00:00,
132.96it/s]
Evaluating nodes: 100%|            | 1056/1056 [00:00<00:00,
1040447.50it/s]
Evaluating FP edges: 100%|            | 940/940 [00:00<00:00,
1306.10it/s]
Evaluating FN edges: 100%|            | 1563/1563 [00:00<00:00,
3323.64it/s]
```


```
[
    {
        'AOGM': 4843.5,
        'fp_nodes': 25,
        'fn_nodes': 363,
        'ns_nodes': 66,
        'fp_edges': 10,
        'fn_edges': 565,
        'ws_edges': 1,
        'TRA': 0.7385354530486653,
        'DET': 0.7537082818294191,
        'gamma': 0.1
    },
    {
        'AOGM': 3550.0,
        'fp_nodes': 50,
        'fn_nodes': 256,
        'ns_nodes': 53,
        'fp_edges': 18,
        'fn_edges': 438,
        'ws_edges': 0,
        'TRA': 0.8083618991065886,
        'DET': 0.8223114956736712,
        'gamma': 0.25
    },
    {
        'AOGM': 2959.5,
        'fp_nodes': 97,
        'fn_nodes': 206,
        'ns_nodes': 49,
        'fp_edges': 7,
        'fn_edges': 367,
        'ws_edges': 0,
```

```
        'TRA': 0.8402386029312532,
        'DET': 0.8515451174289246,
        'gamma': 0.5
    },
    {
        'AOGM': 5798.0,
        'fp_nodes': 36,
        'fn_nodes': 452,
        'ns_nodes': 58,
        'fp_edges': 7,
        'fn_edges': 630,
        'ws_edges': 0,
        'TRA': 0.6870090960619719,
        'DET': 0.7004944375772559,
        'gamma': 1
    }
]
```


1.11 Combined Contours and Detection

The `labels_to_edges` combines multiple segmentation labels into a single detection and contour map.

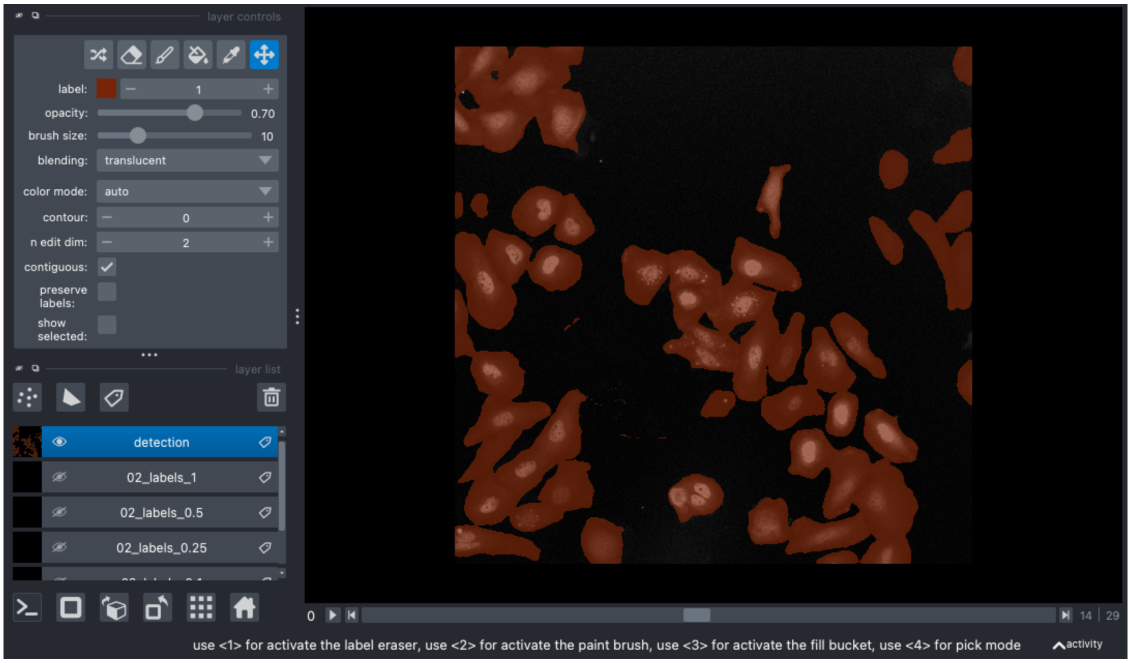
The detection map is the maximum value between the binary masks of each label.

The contour map is the average contour map of the binary contours of each label.

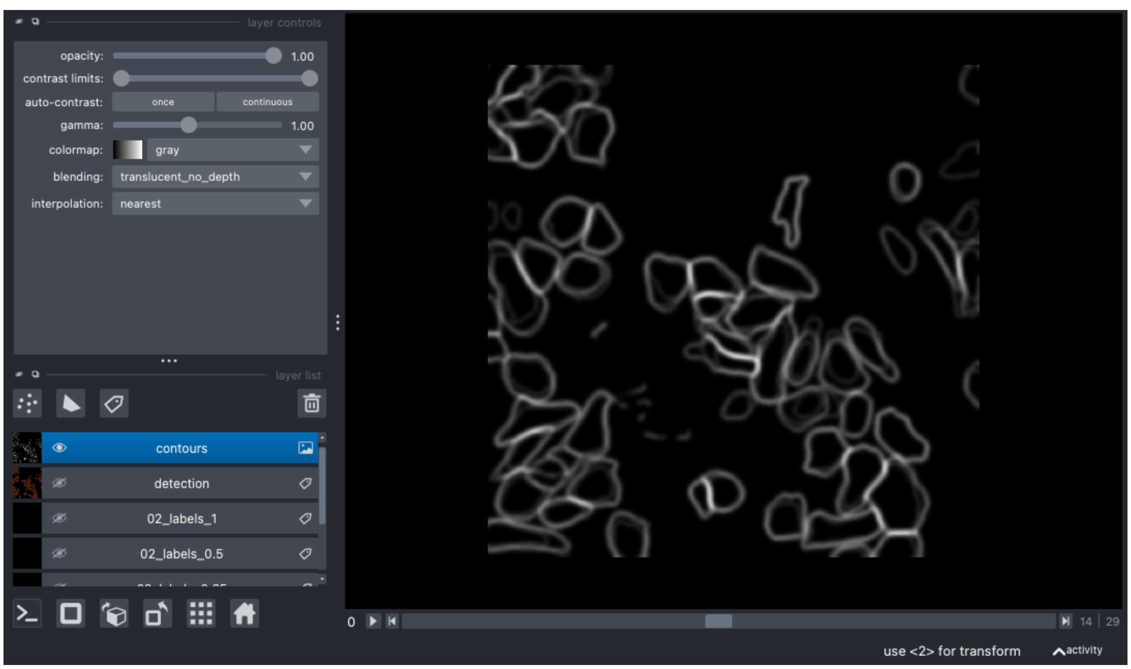
[10]:
```
detection, contours = labels_to_edges(all_labels, sigma=sigma)
```

```
Converting labels to edges: 100%|          | 30/30 [00:05<00:00,
5.36it/s]
```

[11]:
```
if COLAB:
    display(stackview.curtain(image, detection))
else:
    layer = viewer.add_labels(detection)
    screenshot()
    layer.visible = False
```



[12]:
```python
if COLAB:
    display(stackview.curtain(image, contours))
else:
    layer = viewer.add_image(contours)
    screenshot()
    layer.visible = False
```



1.12 Tracking

Run the tracking algorithm on the provided configuration, and combined detections and contours.

[13]:
```python
track(
    config,
    detection=detection,
    edges=contours,
    overwrite=True
)
```

```
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
Adding nodes to database: 100%|          | 30/30 [00:06<00:00,
4.35it/s]
Linking nodes.: 100%|                    | 29/29 [00:03<00:00,
8.46it/s]

Set parameter Username
Academic license - for non-commercial use only - expires 2024-08-17
Using GRB solver
Solving ILP batch 0
Constructing ILP …
Set parameter TimeLimit to value 36000
Solving ILP …
Set parameter NodeLimit to value 1073741824
Set parameter SolutionLimit to value 1073741824
Set parameter IntFeasTol to value 1e-06
Set parameter Method to value 3
Set parameter MIPGap to value 0.001
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 16054 rows, 24412 columns and 59939 nonzeros
Model fingerprint: 0x64652bb9
Variable types: 0 continuous, 24412 integer (24412 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [7e-18, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -0.0000000
```

```
Presolve removed 11468 rows and 14434 columns
Presolve time: 0.09s
Presolved: 4586 rows, 9978 columns, 22944 nonzeros
Found heuristic solution: objective 387.9206864
Variable types: 0 continuous, 9978 integer (9978 binary)
Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only…

Root barrier log…

Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 2.167e+04
 Factor NZ  : 1.395e+05 (roughly 7 MB of memory)
 Factor Ops : 6.349e+06 (less than 1 second per iteration)
 Threads    : 8

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -3.10239587e+05  7.35441630e+05  2.41e+01 3.95e+01  1.92e+02     0s
   1  -1.37653177e+05  4.70084754e+05  1.15e+01 1.08e+01  8.09e+01     0s
   2  -2.40848409e+04  1.38224952e+05  8.15e-01 7.11e-14  9.39e+00     0s

Barrier performed 2 iterations in 0.12 seconds (0.14 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex

Root relaxation: objective 6.995956e+02, 2175 iterations, 0.02 seconds (0.02
work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0     699.5956110  699.59561  0.00%     -    0s

Explored 1 nodes (2175 simplex iterations) in 0.12 seconds (0.14 work units)
Thread count was 10 (of 10 available processors)

Solution count 3: 699.596 387.921 -0

Optimal solution found (tolerance 1.00e-03)
Best objective 6.995956110027e+02, best bound 6.995956110027e+02, gap 0.0000%
Saving solution …
Done!
```

Compute metrics for the multiple hypotheses tracking and compare the scores of the different approaches.

[14]:
```
output_path = Path(f"{dataset}_COMBINED") / "TRA"
to_ctc(output_path, config, overwrite=True)

metric = score(output_path)
metrics.append(metric)

df = pd.DataFrame(metrics)
df.to_csv(f"{dataset}_scores.csv", index=False)
df
```

```
Exporting segmentation masks: 100%|          | 30/30 [00:00<00:00,
111.72it/s]
Loading TIFFs: 100%|                  | 30/30 [00:00<00:00,
852.39it/s]
Matching frames: 100%|                 | 30/30 [00:00<00:00,
115.04it/s]
Evaluating nodes: 100%|            | 1398/1398 [00:00<00:00,
1161116.24it/s]
Evaluating FP edges: 100%|            | 1288/1288 [00:01<00:00,
1184.36it/s]
Evaluating FN edges: 100%|            | 1563/1563 [00:00<00:00,
2994.83it/s]
```

[14]:

	AOGM	fp_nodes	fn_nodes	ns_nodes	fp_edges	fn_edges	ws_edges	\
0	4843.5	25	363	66	10	565	1	
1	3550.0	50	256	53	18	438	0	
2	2959.5	97	206	49	7	367	0	
3	5798.0	36	452	58	7	630	0	
4	1879.0	93	114	46	1	276	1	

	TRA	DET	gamma
0	0.738535	0.753708	0.10
1	0.808362	0.822311	0.25
2	0.840239	0.851545	0.50
3	0.687009	0.700494	1.00
4	0.898567	0.909580	NaN

1.13 Exporting and Visualization

The intermediate tracking data is stored on disk and must be exported to your preferred format. Here we convert the resulting tracks to a DataFrame and Zarr to visualize using napari if running locally.

[15]:
```
tracks_df, graph = to_tracks_layer(config)
tracks_df.to_csv(f"{dataset}_tracks.csv", index=False)
```

```python
segments = tracks_to_zarr(
    config,
    tracks_df,
    overwrite=True,
)

if COLAB:
    display(stackview.curtain(image, segments))
else:
    viewer.add_tracks(
        tracks_df[["track_id", "t", "y", "x"]],
        name="tracks",
        graph=graph,
        visible=True,
    )

    viewer.add_labels(segments, name="segments").contour = 2
    screenshot()
```

```
Exporting segmentation masks: 100%|          | 30/30 [00:00<00:00,
153.60it/s]
```