intro

October 19, 2023

1 Ultrack I2K 2023 - Introduction

This tutorial will introduce the basic concepts of Ultrack and how to use it to track cells when segmentation is already available.

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]:
```python
from pathlib import Path

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.

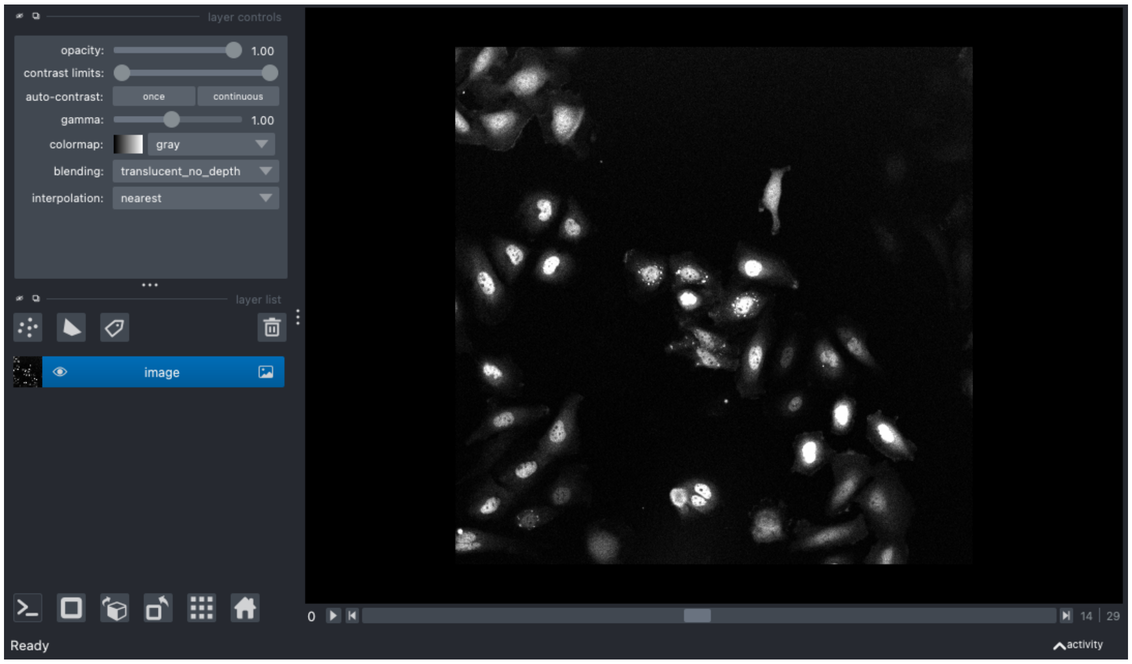
When running locally napari will be used a the image viewer, while on Colab the images will be displayed using `stackview`.

[4]:
```python
# 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]:
```python
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()
```


opacity: 1.00
contrast limits:
auto-contrast: once continuous
gamma: 1.00
colormap: gray
blending: translucent_no_depth
interpolation: nearest

1.7 Cellpose Segmentation

Use the Cellpose model to segment cells within each frame. The function predict applies Cellpose segmentation to each frame after normalizing it.

[6]:
```python
cellpose = Cellpose(model_type="cyto2", gpu=True)
```

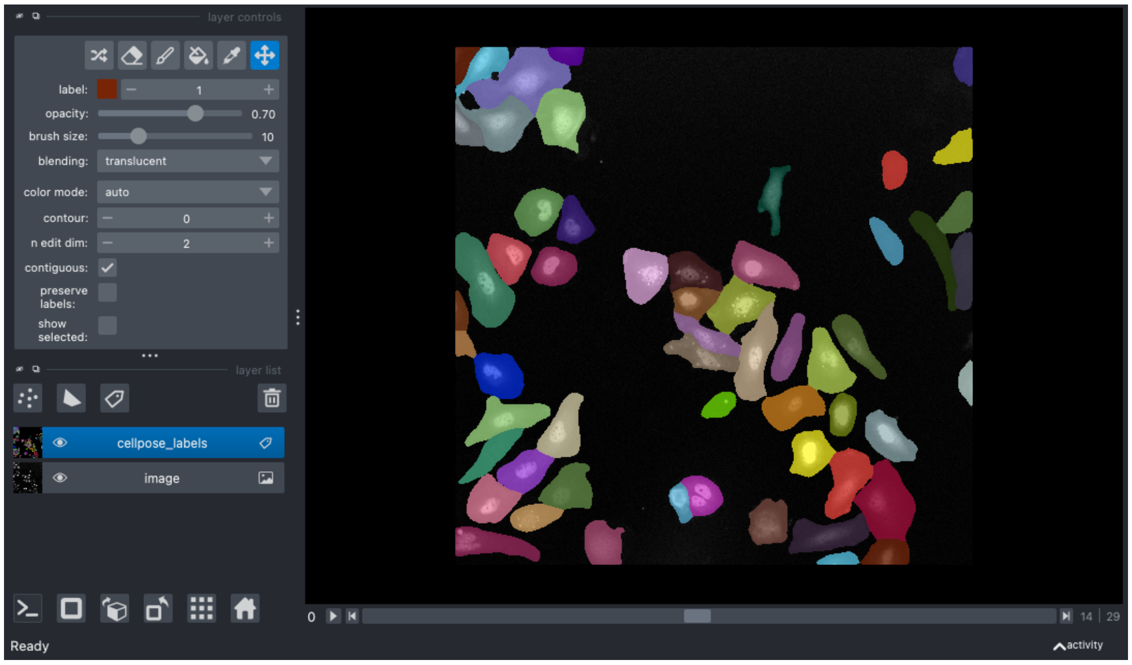
```
def predict(frame: ArrayLike, gamma: float) -> ArrayLike:
    norm_frame = normalize(np.asarray(frame), gamma=gamma)
    return cellpose(norm_frame, tile=False, normalize=False, diameter=75.0)

cellpose_labels = np.zeros(image.shape, dtype=np.int32)
array_apply(
    image,
    out_array=cellpose_labels,
    func=predict,
    gamma=0.5,
)
```

```
Applying predict …:
100%|                            | 30/30 [02:39<00:00,
5.31s/it]
```

1.8 View Segmentations

After obtaining segmented labels, visualize them alongside the original images. This helps to inspect the quality of the segmentation.

[7]:
```
if COLAB:
    display(stackview.curtain(image, cellpose_labels))
else:
    layer = viewer.add_labels(cellpose_labels)
    screenshot()
    layer.visible = False
```



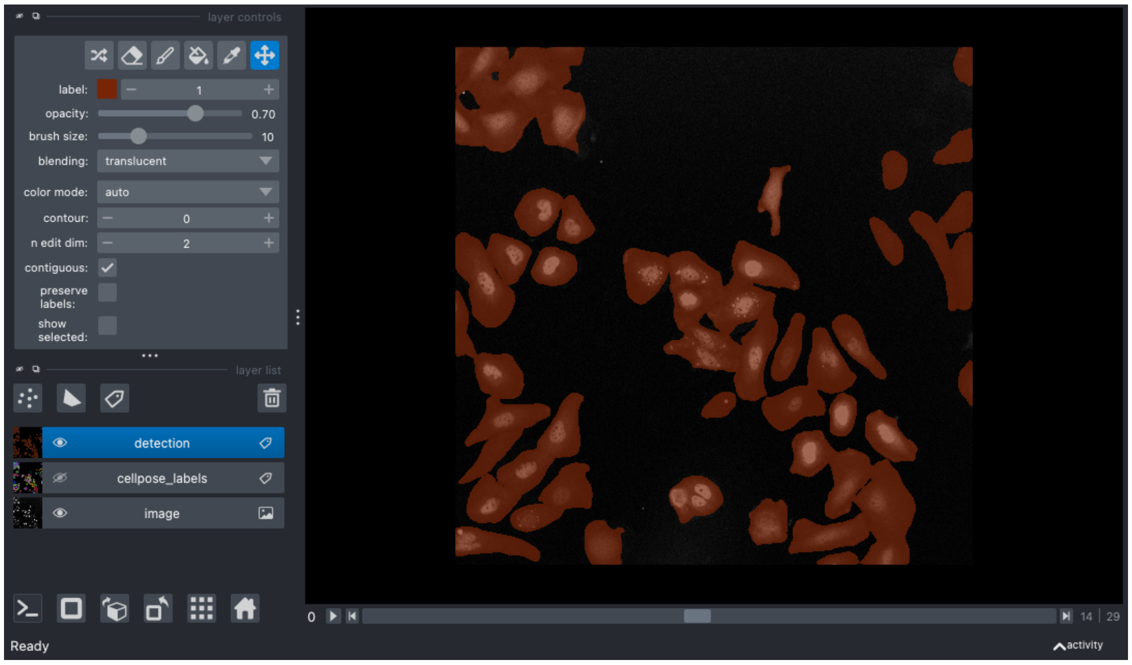
1.9 Extract Contours and Detection

We converted the segmentation labels to contour and detection maps. These maps are the intermediate representation of used by Ultrack.

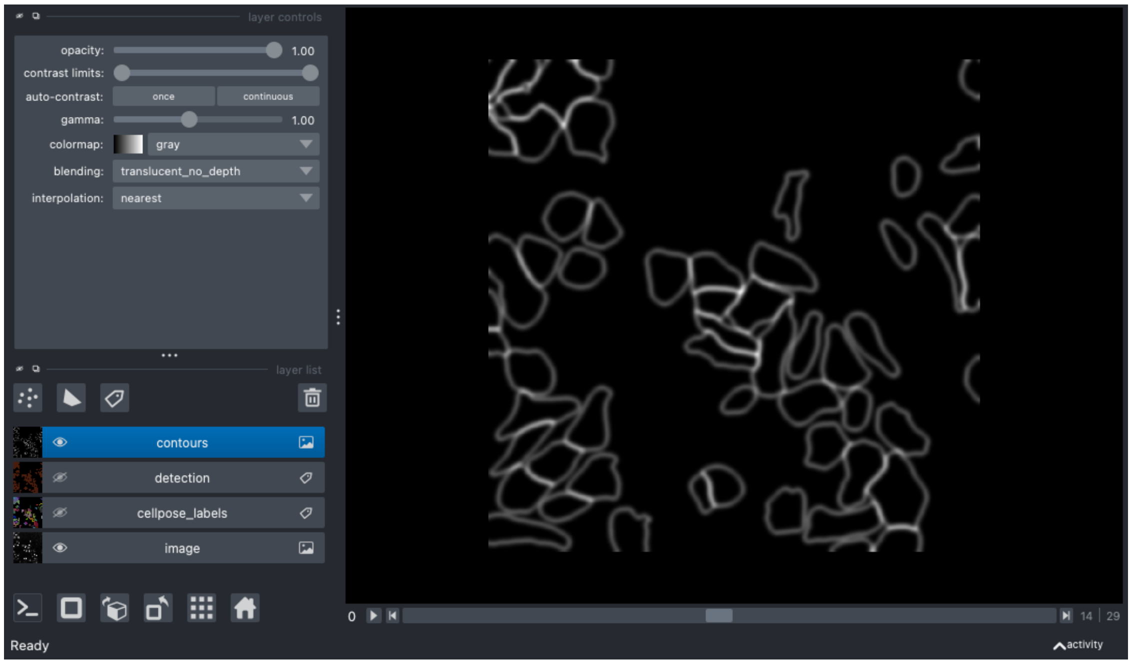
[8]:
```
detection, contours = labels_to_edges(cellpose_labels, sigma=5.0)
```

```
Converting labels to edges:
100%|                              | 30/30 [00:01<00:00,
15.76it/s]
```

[9]:
```
if COLAB:
    display(stackview.curtain(image, detection))
else:
    layer = viewer.add_labels(detection)
    screenshot()
    layer.visible = False
```



[10]:
```
if COLAB:
    display(stackview.curtain(image, contours))
else:
    layer = viewer.add_image(contours)
    screenshot()
    layer.visible = False
```



1.10 Configuration

Set tracking parameters with ultrack's `MainConfig`.

The `track` procedure is composed of three steps that can also be called individually: - `segment`: Computes the segmentation hypotheses for tracking; - `link`: Links and assign edge weights to the segmentation hypotheses; - `solve`: Solves the tracking problem by selecting the strongly connected segmentation hypotheses.

Each of these steps requires its own configuration, which we'll set up below. Its documentation can be found here.

[11]:
```
config = MainConfig()

# Candidate segmentation parameters
config.segmentation_config.n_workers = 4
config.segmentation_config.min_area = 2500
config.segmentation_config.min_frontier = 0.1

# 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 = 4

# Adding absurd weight to division because there are few dividing cells
config.tracking_config.division_weight = -100
```

```python
# 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=0.1,
        anisotropy_penalization=0.0,
        max_noise=0.0,
        ws_hierarchy=<function watershed_hierarchy_by_area at 0x1602ff060>,
        n_workers=4
    ),
    linking_config=LinkingConfig(
        n_workers=4,
        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
    )
)
```

1.11 Tracking

Run the tracking algorithm based on the provided configuration, detected regions, and contours.

[12]:
```python
track(
    config,
    detection=detection,
    edges=contours,
    overwrite=True
)
```

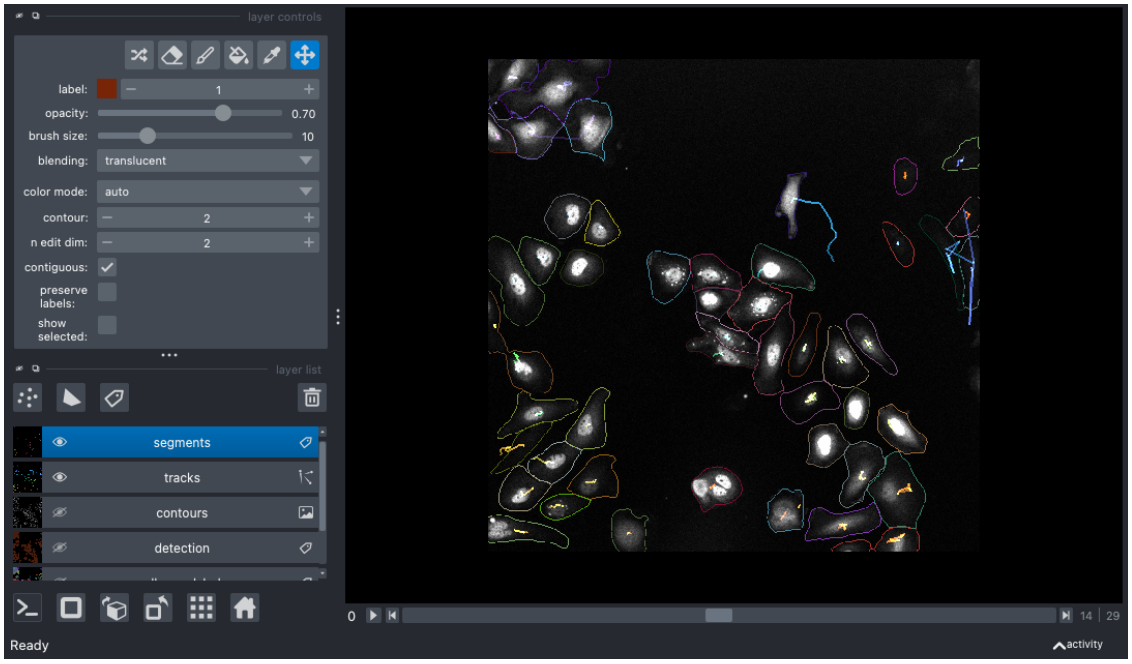
```
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
WARNING:ultrack.core.segmentation.processing:Found zarr with MemoryStore. Using
an zarr with MemoryStore can lead to considerable memory usage.
Adding nodes to database:
100%|                           | 30/30 [00:06<00:00,
4.81it/s]
Linking nodes.:
100%|                                | 29/29
[00:02<00:00, 11.35it/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 11231 rows, 18664 columns and 43018 nonzeros
Model fingerprint: 0xbfead154
Variable types: 0 continuous, 18664 integer (18664 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 7644 rows and 11057 columns
Presolve time: 0.07s
Presolved: 3587 rows, 7607 columns, 17190 nonzeros
Found heuristic solution: objective 386.1439665
Variable types: 0 continuous, 7607 integer (7607 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     : 1.490e+04
 Factor NZ  : 8.872e+04 (roughly 5 MB of memory)
 Factor Ops : 2.850e+06 (less than 1 second per iteration)
 Threads    : 8

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -2.05859668e+05  4.98132126e+05  1.76e+01 3.23e+01  1.44e+02     0s
   1  -6.07232402e+04  2.40392361e+05  5.68e+00 1.49e+00  3.99e+01     0s

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


Solved with dual simplex

Root relaxation: objective 6.916134e+02, 1517 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     691.6134063  691.61341  0.00%     -    0s

Explored 1 nodes (1517 simplex iterations) in 0.10 seconds (0.10 work units)
Thread count was 10 (of 10 available processors)

Solution count 3: 691.613 386.144 -0

Optimal solution found (tolerance 1.00e-03)
Best objective 6.916134063371e+02, best bound 6.916134063371e+02, gap 0.0000%
Saving solution …
Done!
```

1.12 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.

[13]:
```
tracks_df, graph = to_tracks_layer(config)
tracks_df.to_csv(f"{dataset}_tracks.csv", index=False)

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,
162.76it/s]
```



1.13 Run Metrics

Finally, we evaluate the tracking performance using `traccuracy` with the metrics and annotations from the Cell Tracking Challenge.

[14]:
```python
name = f"{path.parent.name}_{path.name}".upper()
output_path = Path(name) / "TRA"
to_ctc(output_path, config, overwrite=True)

gt_path = path.parent / f"{dataset}_GT" / "TRA"

run_metrics(
    gt_data=load_ctc_data(gt_path),
    pred_data=load_ctc_data(output_path),
    matcher=CTCMatched,
    metrics=[CTCMetrics],
)["CTCMetrics"]
```

```
Exporting segmentation masks:
100%|                             | 30/30 [00:00<00:00,
113.46it/s]
Loading TIFFs:
100%|                                     | 30/30
[00:00<00:00, 462.28it/s]
Loading TIFFs:
100%|                                     | 30/30
[00:00<00:00, 904.92it/s]
```

```
Matching frames:
100%|                                    | 30/30
[00:00<00:00, 118.69it/s]
Evaluating nodes: 100%|                              |
1324/1324 [00:00<00:00, 1124824.49it/s]
Evaluating FP edges: 100%|                              |
1203/1203 [00:00<00:00, 1222.70it/s]
Evaluating FN edges: 100%|                              |
1563/1563 [00:00<00:00, 3078.62it/s]
```

```
[14]: {'AOGM': 2948.5,
        'fp_nodes': 96,
        'fn_nodes': 205,
        'ns_nodes': 49,
        'fp_edges': 7,
        'fn_edges': 367,
        'ws_edges': 0,
        'TRA': 0.8408324111312047,
        'DET': 0.8522249690976514}
```