A. Appendix

A. Appendix

A.1. Self-supervised pretraining

For self-supervised pre-training, we use a method based on the contrastive loss formulation of [9]. Consider a graph $G = (V, E)$, where the nodes in $V = \{1, 2, ..., n\}$ correspond to the individual superpixels and the edges in $E = \{(i, j)|i \neq j \,\text{and}\, i, j \in V\}$ connect nodes with adjacent superpixels. In addition, consider edge weights $W \in \mathbb{R}^{|E|}$ associated with each edge. Here, we infer the weights from pixel-wise boundary probability predictions and normalize the weights such that $\sum_{w \in W} w = 1$ holds. We train a 2D UNet to predict embeddings for each node in V by pulling together pixel embeddings that belong to the same superpixel and pushing apart pixel embeddings for *adjacent* superpixels. The intensity of the push force is scaled by the weight of the respective edge. With pixel embeddings x_n and node embeddings $f_i = \frac{1}{m_i} \sum_{k \in s_i} x_k$, where m_i is the mass of the superpixel for node i and s_i is the set of indices for all pixels of the corresponding superpixel, and in accordance with [9] we formulate the loss as

$$\mathcal{L}_{var} = \frac{1}{|N|} \sum_{i=1}^{|N|} \frac{1}{m_i} \sum_{n=1}^{m_i} \left[d(f_i, x_n) - \delta_v \right]_+^2 \tag{8}$$

$$\mathcal{L}_{dist} = \sum_{(i,j) \in E}^{|E|} w_{(i,j)} \left[2\delta_d - d(f_i, f_j) \right]_+^2 \tag{9}$$

$$\mathcal{L}_{feat} = \mathcal{L}_{var} + \mathcal{L}_{dist} \tag{10}$$

Here $[\cdot]_+$ refers to selecting the max value from the argument and 0. The forces are hinged by the distance limits δ_{var} and δ_{dist}. $d(\cdot)$ refers to the distance function in the embedding space. Since the feature extractor is trained self-supervised, we give it a smooth edge map of the superpixels as well as the raw data as an input, see Fig. 6.

The training of the feature extractor happens prior to training the agent for *Method 2*.

A.2. Reward Generation

We seek to express the rewards based on prior rules derived from topology, shape, texture, etc. Rules are typically formulated per-object. Section A.7 describes the object-to-sub-graph reward mapping. The reward function is part of the environment and the critic learns to approximate it via Q-learning, enabling the use of non-differentiable functions.

This approach can also be extended to semantic instance segmentation where in addition to the instance labeling a semantic label is to be predicted. To this end, each predicted object is softly assigned to one of the possible classes and the reward is generated specifically for the predicted class. We make use of this extension by separating the objects into a foreground and background class in our experiments.

In addition to the sub-graph rewards our approach can also be extended to global rewards by global pooling of the output of the critic GNN and adding the squared difference of global action value and reward to Equation 4. Alternatively, the global reward can be distributed onto the sub-graph rewards via a weighted sum of sub-graph reward and global reward. In the second approach a different global reward can be specified per class in the case of the semantic instance segmentation formulation. We make use of the per class global reward to encode a reward for the correct number of predicted objects in the experiments for synthetic data (Subesection 4.1).

The biggest challenge in designing the reward function is to avoid local optima. Since the reward is derived from each predicted object, we define the reward by extracting shape features, position, orientation and size of objects and compare them with our expectation of the true object's features. This similarity score should be monotonically increasing as the objects fit our expectation better. All used similarity functions are to a certain extend linear, however an exponential reward function can speed up learning significantly. Consider an object level reward $r \in [0, 1]$, which is linear. We calculate the exponential reward by

$$r_{exp}(r) = \frac{exp(r\theta)}{exp(\theta)} \tag{11}$$

where the factor θ determines the range of the gradient in the output. We also find that it is better to compute the reward as a "distance function" of all relevant features rather than decomposing it into the features and simply summing up the corresponding rewards. In our experiments the latter approach behaved quite unpredictably and often generated local optima which the agent could not escape.

A.3. Object level rewards

We have tested generating the rewards based directly on the object scores instead of using the subgraph decomposition described in Section A.7. Since rewards are mainly derived from the features of the predicted objects it seems reasonable to formulate the supervision signal directly for objects. To this end we calculate a scalar reward per object as sketched in Figure 7. In this case, the critic uses a second GNN to predict the per-object action values. It is applied to an object's subgraph, which is composed of all edges that have at least one node in common with the respective object. The graph convolutions are followed by a global pooling operation which yields the scalar action value. This GNN replaces the MLPs used in the case of the reward subgraph decomposition. After extensive testing, we found that this approach is always inferior to the subgraph decomposition.

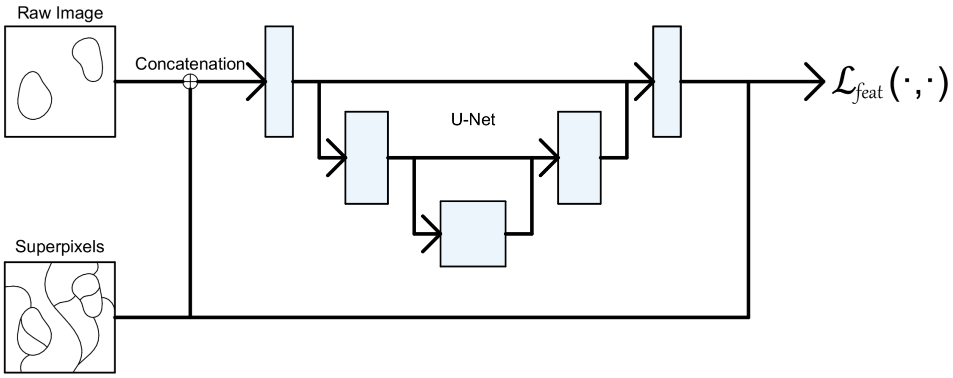


Figure 6: Training setup of the feature extractor. The input is a concatenation of the raw data and a smoothed edge map of the superpixels. The superpixel over-segmentation is used in the loss again as the supervision for learning the embedding space.

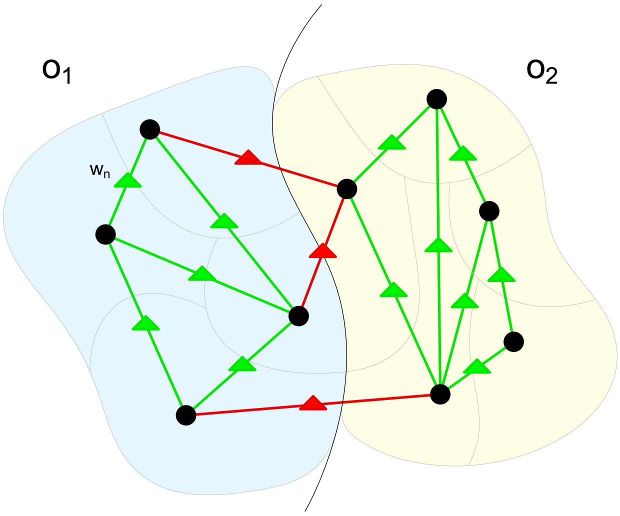


Figure 7: Object level rewards. We accumulate edge rewards over each object where we consider all edges that have at least one node within the respective object. E.g. for o_1 we consider all edges that are covered by the light blue object as well as all the red ”split” edges.

A.4. Impact of different feature space capacities

In Tab 3 we compare the performance of our method, using different dimensionalities of the learned feature space (the number of channels in the output of the feature extractor UNet). We find that the reduced capacity of small feature spaces improves the agents performance. Here we train and evaluate on the fruit fly embryo dataset from Subsection 4.3.

A.5. Random seed evaluation

Figure 8 shows the evolution of the average subgraph reward furing training for different random seeds. The model performance depends on the chosen seed and for the final comparisons we select the runs based on the best score. The seed is generated randomly for each run.

n channels	VI	VI merge	VI split
4	1.712	0.889	0.823
12	2.062	0.821	1.241
16	2.212	0.838	1.374

Table 3: Quantitative evaluation of our method using different feature space dimensionality. We use the same metrics for evaluation as in Tab.2 and compare all results on the validation set.

A.6. Gaussian weighting scheme

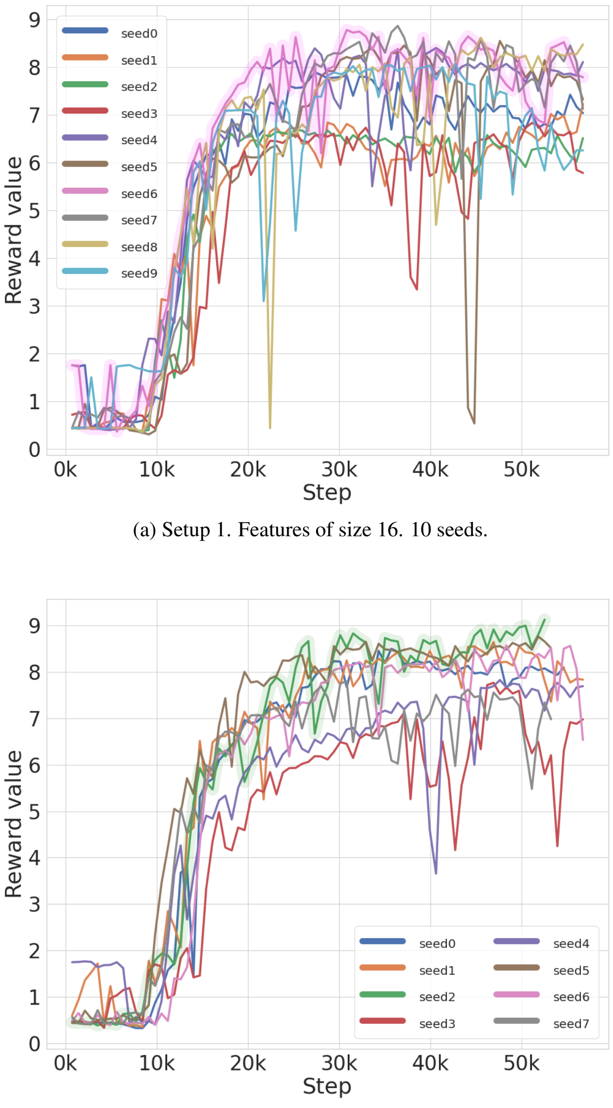
Fig. 9 shows the weighting scheme which was used to generate the rewards for the fruitfly embryo data (Subsection 4.3). It can be seen as a very approximate semantic segmentation and serves the purpose of generating a reward maximum at the approximate foreground locations.

A.7. Randomly generated subgraphs

We select subgraphs using Alg. 1. Subgraphs are selected randomly starting from random nodes and continuously adding edges to the subgraph until the desired size is reached. The size of the subgraph is defined by the number of edges in the graph. The algorithm selects edges such that the subgraphs are connected and such that their density is high (low number of nodes in the subgraph).

A.8. Multistep Reinforcement Learning

We tested several methods that use multiple steps within one episode. In this formulation we predict the changes starting from an initial state rather than predicting absolute values for the edge weights. For example, we can start from



(a) Setup 1. Features of size 16. 10 seeds.

(b) Setup 2. Features of size 12. 8 seeds.

Figure 8: Running the same setup for different random seeds reveals a stable trend towards larger rewards. We select the model for comparison based on the best achieved reward (magenta line in Fig. 8a and green line in Fig. 8b).

Algorithm 1: Dense subgraphs in a rag

```
Data: $G = (V, E)$, $l$
Result: subgraphs by sets of $l$ edges
Initialization:$SG = \emptyset$;
while $E \backslash SG \neq \emptyset$ do
    pq = PriorityQueue;
    prio = 0;
    n_draws = 0;
    $sg = \emptyset$;
    $sg_{vtx} = \emptyset$;
    $i, j = (ij)$ s.t. $(ij) \in E \backslash SG$;
    pq.push($i$, prio);
    pq.push($j$, prio);
    $sg = sg \cup (ij)$;
    $sg_{vtx} = sg_{vtx} \cup i$;
    $sg_{vtx} = sg_{vtx} \cup j$;
    while —$sg$— ¡ $l$ do
        $n$, n_prio = pq.pop();
        n_draws ++;
        $adj = \{(nj) | \exists (nj) \in E$ and $\exists j \in sg_{vtx}\}$;
        forall $(nj) \in adj$ do
            $sg = sg \cup (nj)$;
            n_draws = 0;
        if $|adj| < deg(n)$ then
            n_prio -= $(|adj| - 1)$;
            pq.push($n$, n_prio);
        if pq.size() $\leq$ n_draws &
          $\exists j | (nj) \in E, j \notin sg_{vtx}$ then
            $j \in \{j | (nj) \in E, j \notin sg_{vtx}\}$;
            prio ++;
            pq.push($j$, prio);
            $sg = sg \cup (nj)$);
            $sg_{vtx} = sg_{vtx} \cup j$;
    $SG$ = $sg \cup SG$
return $SG$
```

a state defined by edge weights derived from a boundary map. Given that this state should be somewhat close to the desired state we expect that a few small steps within one episode should be sufficient. In our experiments, we have typically used three steps per episode and used actions that can change the weight per edge by the values in $[-0.1, 0.1]$. This approach generates an action space that is exponentially larger than in the stateless formulation. A priori this setup might still be more stable because it is not possible to diverge from a given solution so fast due to the incremental changes per step.

Let us first consider the case with groundtruth edge weights. In this case, we can give an accurate reward not only for the final segmentation but for every step. Hence, the path to the optimal state is linear. Take for example an initial edge with weight 0.3 and its respective ground truth edge with weight 0. We can give a reward that mirrors the correct confidence of the action by using the negative value of the predicted action: $r = -a$. This allows us to set the discount factor γ of the RL setup to 0, because the path to the correct edge weight is linear and the correct direction will be encoded in the reward at every step. Therefore the rewards for the following steps are not needed. Setting the discount to 0 generates a problem of equal size as the stateless RL method.

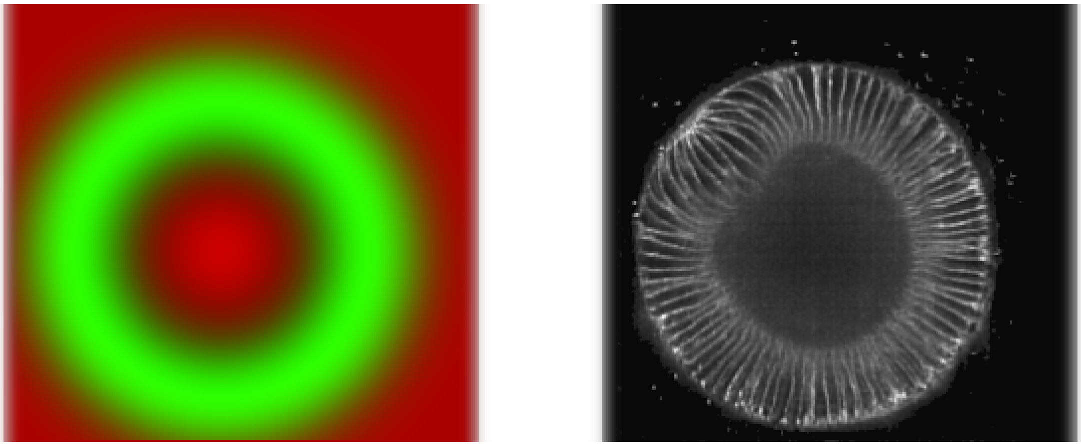


Figure 9: Weighting scheme for object rewards and merge affinity rewards, roughly encoding foreground location in 4.3. Left: weights for object rewards in green and for merge affinity rewards in red, both have a gaussian profile and are concentric. Right: corresponding light-sheet image.

However, for this approach the ground truth direction of the path must be known for each edge, so it can only be used in the case of full supervision.

To generalize the multistep approach to the rule-based rewards we need to choose a different setup where a constant reward is given at each non terminal step and the rule-based one is given at the terminal step. This setup requires a discount factor $\gamma > 0$ and has an action space more complex than the stateless approach, because future steps are necessary to compute the reward. We tested this setup extensively against the stateless approach and found that it was not competitive.

A.9. Network architectures, hyperparameters and experiment details

The U-Net used for feature extraction is based on a slightly modified implementation of the standard 2D U-Net. It uses max-pooling to spatially downsample the features by a factor of 2 in the encoder and transposed convolutions to upsample them in the decoder. All convolutions have a 3x3 kernel size, followed by ReLU activations and group normalization. It has four levels, each level consisting of 2 convolutional blocks followed by downsampling (encoder) / upsampling (decoder), the features from the encoder are concatenated to the decoder inputs on the same level using skip connections. For the experiments we use feature maps of size 32, 64, 128, 256 for the different levels; except for the nucleus segmentation experiment where we use 16, 32, 64, 128.

Both actor and critic are based on GNNS: the GNN for the actor predicts the action per edge and the critic has a GNN that also predicts per edge outputs. Both GNNs use an architecture of depth two and each layer consists of an MLP with three hidden layers and 2028 units per hidden layer. In the case of the critic we have an additional MLP per sub-graph size that takes the edge outpus of the GNN as input and predicts the action value for a given sub-graph. Each of these MLPs has two hidden layers with 2028 hidden units each.

We use the Adam optimizer with a learning rate of 0.0001 and the pytorch default values for all other hyperparameters. We have trained the networks on different GPUs, depending on the problem size (which is defined by the number of superpixels): Nvidia Geforce GTX 2080 TIs for the synthetic data experiments, Nvidia Geforce RTX 3090s for the nucleus segmentation experiments and Nvidia A100s for the cell segmentation experiments. A single training run always used a single GPU.

The cell segmentation dataset is available on a general-purpose open-access repository Zenodo [1].

A.10. Direct supervision

We have also implemented a set-up for direct supervision that can be applied if any of the images have a groundtruth pixelwise segmentation. We investigated both full supervision (Fig. 10) and mixed supervision, where one fully segmented image was used in addition to the prior rules (Fig. 11). Under full supervision with a set of groundtruth edge weights, we compute the Dice Score DBLP:journals/corr/abs-1906-11031 of the predicted edge weights a and the ground-truth $\hat{a}$ for each sub-graph and use it as the reward. We find this approach to be robust against class imbalance. In both cases, the agent learns to segment the circles correctly, demonstrating fast and robust convergence. Note that learned pixel features converge to a state which strongly resembles a semantic segmentation of the

[1] https://zenodo.org/record/4899944#.YORWq0xRVEZ

image. We have also used the mixed-supervision approach in Subsection 4.3.

A.11. Synthetic Circle Experiments

For the synthetic circular data we implement a reward function based on the circular hough transform (CHT). The object rewards r_{fg} are computed as follows: we define a threshold γ for the CHT value (Fig. 3 shows the reward surface for $\gamma = 0.8$). Let $c \in [0, 1]$ be the CHT value for to the given object, let k be the number of expected objects and n be the number of predicted objects. We then define the local and global reward per object as follows.

$$r_{locacl} = \begin{cases} \sigma\left((\frac{c-\gamma}{1-\gamma} - 0.5)\, 6 \right) 0.4, & \text{if } c \geq \gamma \\ 0, & \text{otw} \end{cases} \tag{12}$$

$$r_{global} = \begin{cases} 0.6 \, \left(\frac{k}{n}\right), & \text{if } n \geq k \\ 0.6, & \text{otw} \end{cases} \tag{13}$$

$$r_{fg} = r_{local} + r_{global} \tag{14}$$

Here $\sigma(\cdot)$ is the sigmoid function. The input to $\sigma(\cdot)$ is normalized to the interval $[-3, 3]$. The rewards are always in $[0, 1]$, the local reward is in range $[0, 0.4]$ and the global reward is in range $[0, 0.6]$.

We assume that the largest object is the background and assign it the background reward:

$$r_{bg} = \begin{cases} \left(\frac{n}{k}\right), & \text{if } n \leq k \\ 1, & \text{otw} \end{cases} \tag{15}$$

A.12. DSB Experiments

We compute the per-object rewards based on the following properties: the eccentricity, the extent, the minor axis length, the perimeter, the solidity as well as the mean, maximal and minimal intensity. The properties are computed using the "regionprops" functionality from skimage. Using these properties p_i and their expected values $\hat{p}_i$ determined for simplicity from ground-truth objects, we compute the reward r_o as

$$r_o = \frac{1 + cs(\frac{p_i}{n_i}, \frac{\hat{p}_i}{\hat{n}_i})}{2}. \tag{16}$$

Here, n_i and $\hat{n}_i$ are normalization factors to bring each property into the range $[0, 1]$ and $cs(\cdot, \cdot)$ is the cosine similarity. Note that objects that are identified as background, using a simple size criterion, do not receive any reward. Projecting this reward to edges using the maximum (see Section 3.2) yields r_{oe}, and the final edge-level reward r_e is constructed by the sum

$$r_e = \alpha \, r_{oe} + \beta \, \hat{r}_e. \tag{17}$$

Here α and β are scaling factors and $\hat{r}_e$ is a reward based on the distance of the action to the difference of the mean intensity μ of the incident superpixels of edge (i, j)

$$\hat{r}_{e,ij} = |(1 - a_{ij}) - (|\mu_i - \mu_j|)|. \tag{18}$$

If the action for an edge is close to 1, i.e. strongly favoring a split, and the difference in the mean intensity is high, there will be a large reward and vice versa.

For the DSB experiments we use three sets of superpixels:

- "GT": these superpixels take into account ground-truth information. They are used to judge the performance of our method without being limited by the quality of the underlying superpixels. For these superpixels we compute a height map based on the gradient of the input image and then perform a seeded watershed transform with seeds at the minima of the height map. The resulting superpixels are intersected with the groundtruth, further breaking into pieces the superpixels which cover more than one object or that spill into the background.

- "UNET": these superpixels are based on predictions from a pre-trained U-Net and allow the performance of our method when having access to such a pre-trained network. The U-Net predicts foreground and boundary probabilties and the superpixels are computed via a watershed that uses the minima of the boundary probabilities as seeds and also uses these probabilties as heightmap. Befor computing the seeds the boundary probability are smoothed, using a higher smoothing factor in the background than in the foreground, which is determined based on the foreground predictions.

- "RAW": these superpixels are computed based on raw image data only. They are computed similar to the "UNET" superpixels, but the gradient image of the input is used to compute seeds and as heightmap instead of the boundary predictions. To determine foreground vs. background the otsu thresholded intensity image is used.

Furthermore, we use groundtruth objects to determine expected values of the priors. In practice, these are usually known from biology (shape priors) or can be determined from bulk measurements without segmentation (intensity priors).

Fig. 12 shows segmentation results for the methods from Tab. 1 for several images from the test set. Here, red arrows mark segmentation errors that wrongly split a nucleus, purple arrows mark segmentation errors that wrongly merge nuclei and yellow arrows mark segmentation errors that either omit nuclei or segment them with a completely wrong shape. Note that these errors were manually annotated and are not exhaustive for the shown images. We observe that different methods suffer from different systematic errors: Our proposed method suffers from merges, sometimes omits

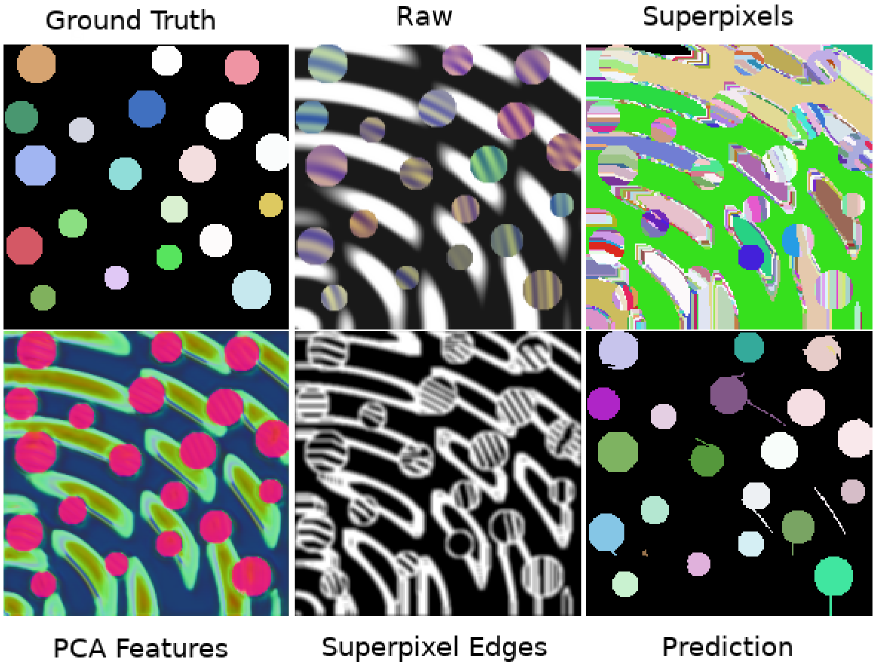


Figure 10: An example of a fully supervised prediction on the synthetic dataset. Here, we use the edge-based Dice score over subgraphs and make use of *Method 1*, i.e. joint training of the feature extractor, but we initialized it using the weights pretrained by self-supervision. The features for the circles are significantly more pronounced after training.

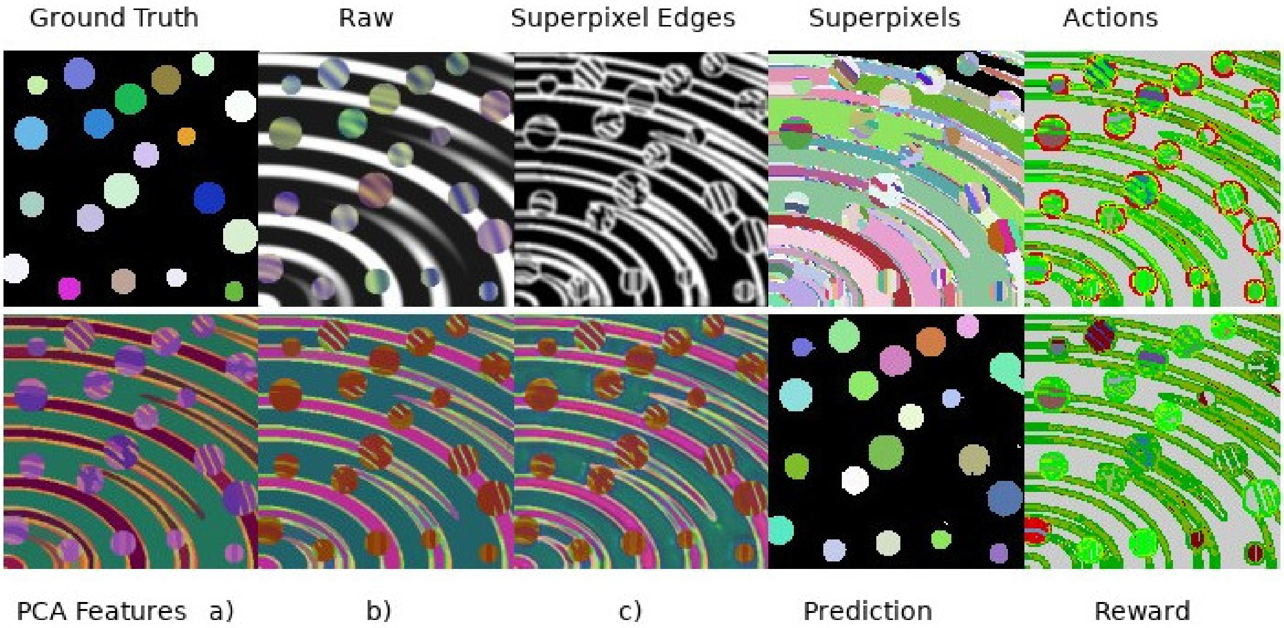


Figure 11: An example for prediction with mixed supervision on the synthetic dataset. The reward was defined as follows: for all but one image we use the unsupervised CHT reward and for one image we make use of ground truth and the Dice score. We find that this mixed reward setting leads to improved performance compared to the unsupervised CHT reward.

nuclei and in some cases wrongly splits of a small superpixel from a nucleus. The UNet predominantly suffers from merges. The combination of UNet and Multicut, which uses the same superpixels as our method, suffers from merges and omissions, it also systematically splits off superpixels located on the boundary of nuclei, which can especially be seen for the images in the third and fourth row. Stardist and Cellpose, which use use a strong shape prior, do not suffer from merges. Instead, they often wrongly split up nuclei that do not adhere to the shape prior and sometimes omit nuclei or predict them with a very wrong shape. Note that both methods with shape priors result in round shapes that may not match the ground-truth objects for high intersection thresholds, even if they are visually matching well.