A. Supplementary materials for "ColorRL: Reinforced Coloring for End-to-End Instance Segmentation"

A.1. PixelRL

In this work, we employed the method introduced by Furuta *et al.* [3], which uses an efficient technique within a multi-agent system (PixelRL) for image processing. The technique works well with the asynchronous actor-critic (A3C) algorithm [5] and other reinforcement learning algorithms for a discrete environment like deep Q-network [6]. In the PixelRL problem setting, an image I has a set of pixels $V = \{v_1, v_2, ..., v_N\}$. Each v_i has a corresponding state $s_i^{(t)}$ at time step t. A pixel-level agent with policy $\pi(a_i^{(t)} \mid s_i^{(t)})$ is assigned to each pixel v_i. State $s^{(t+1)} = (s_1^{(t+1)}, s_2^{(t+1)}, ..., s_N^{(t+1)})$ and reward $r^t = (r_1^t, r_2^t, ..., r_N^t)$ are obtained from the environment by taking action $a^t = (a_1^t, a_2^t, ..., a_N^t)$, $a_i^t \in \mathcal{A}$. In our work, $\mathcal{A}$, which represents the binary segmentation map of multiple objects at a time step, consists of two values: 0 and 1. The agents try to learn a policy that maximize the mean of their total expected reward:

$\pi^* = \arg\max_\pi E_\pi(\sum_{t=0}^{\infty} \gamma^t \bar{r}^t), \qquad \bar{r}^t = \frac{1}{N}\sum_{i=1}^{N} r_i^t$

where $\bar{r}^{(t)}$ is the mean of all the rewards r_i^t received by the agents at time step t. For more information, see Furuta *et al.* [3]. At each time step t, with state $s^{(t)}$, each pixel-level agent computes the value function $\mathcal{V}(s^{(t)})$ and policy function $\pi(s^{(t)})$. $\mathcal{V}(s^{(t)})$ estimates the expected reward an agent can get from the state $s^{(t)}$, which implies how good the state $s^{(t)}$ is. Loss functions L_{value}^i of $\mathcal{V}$ and L_{policy}^i of π for a single agent at pixel v_i are computed as follows:

N-step return:

$$R_i^{(t)} = r_i^{(t)} + \gamma r_i^{(t+1)} + \gamma^2 r_i^{(t+2)} + \dots + \gamma^{n-1} r_i^{(t+n-1)} + \gamma^n \mathcal{V}(s_i^{(t+n)}))$$

Value loss:

$$L_{value}^i = (R_i^{(t)} - \mathcal{V}(s_i^{(t)}))^2$$

Policy loss:

$$L_{policy}^i = -log(\pi(a_i^{(t)} \mid s^{(t)}) A(s_i^{(t)}))$$

where $A^t = R_i^{(t)} - \mathcal{V}(s_i^{(t)})$ is the advantage function, which shows how good the action a_i^t at step t is compared with the expected return. γ is the discounting factor. At each time step t, gradients for value loss and policy loss are computed and used to update the parameters of $\mathcal{V}$ and π. In PixelRL, a convolutional neural network is used to compute $\mathcal{V}$ and π; $\mathcal{V}$ and π have the same dimensions as the input image $s^{(t)}$.

A.2. Coloring algorithm

To further explain Eq. 1 in the main text and the coloring process of ColorRL, we provide pseudo code for the coloring process of the agent in Algorithm 1.

Algorithm 1: Agent's coloring algorithm

input: I: Input image of shape H×W
$\mathcal{F}$: Agents's network
T: number of coloring steps
$C \leftarrow$ Zero-valued array of shape H×W
$C_{bin} \leftarrow$ Zero-valued array of shape H×W×T
$t \leftarrow 0$
while $t < T$ **do**
  $action \leftarrow 2^t \mathcal{F}(I, C_{bin})$
  for $i \leftarrow 1$ **to** H **do**
    for $j \leftarrow 1$ **to** W **do**
      $C[i,j] \leftarrow C[i,j] + 2^t action[i,j]$
      $C_{bin}[i,j,t] \leftarrow action[i,j]$
    end
  end
end
return C

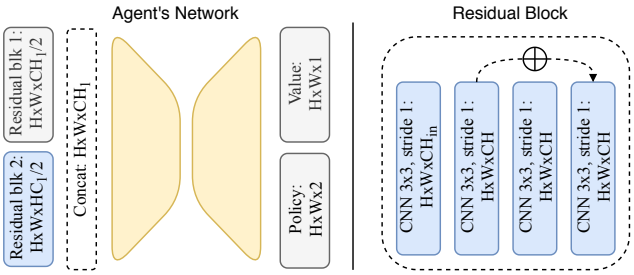

Figure 1. Our agent's network architecture for 2D images (3D version is similar). The two input modules are residual blocks (shown on the right).

A.3. Implementation Details

A.3.1 A3C Implementation

We modified an open source implementation[1] so that it can work with the PixelRL setting. This A3C implementation allows worker agents to run parallel on graphics processing units (GPUs). Because in our coloring algorithm, at a step, the reward from the agent's current action is as important as future rewards, we let discounting factor $\gamma = 1$. A generalized advantage estimation [10] was also used in the implementation. We set n=T in the n-step return setting during training.

A.3.2 Agent's Network Architecture

Our agent's network for 2D images is shown in Fig. 1. For the instance segmentation of 2D images, we used the Attention UNet architecture [7] with five levels of encoding for the core architecture. The number of features are doubled after each level: [32, 64, 128, 256, 512]. For 3D images, we used the original UNet architecture [9], with the number of features for each layers being [8, 16, 32, 64, 128]. Input image I and current color map

[1] https://dgriff777/rl_a3c_pytorch

$C^{(t)}$ are processed by two different residual modules. The two modules have the same architecture, which is composed of four CNN layers. The output of the two layers are concatenated before being processed by the core architecture. During training, the network produce policy map π of shape H×W×2 and value map $\mathcal{V}$ of shape H×W×1 (H×W×D×2 and H×W×D×1, respectively, for 3D data). We use only the output of policy map π for inference after the training is finished.

A.4. Additional Ablation Study

We continue the ablation study experiment with only one training sample and plot the rewards of the agent over the training iterations in Fig. 2. As the agent explores for better decisions (better sum of all rewards) during early training iterations (iterations 5–10), the merging reward reduces while the splitting reward increases. The behaviors of the rewards function is fully expected when the agent is affected by the early affluence of R_{BF}. During later iterations (iterations 10–40), both reward components increase together, and reach achieves the highest reward sum. This experiment suggests that the splitting weight w_s should not be less than the merging weight w_m for the agent to quickly get over the local minimum (like in iterations 0–5).

A.5. Experimental Details

A.5.1 Common Training Details

In this section, we will go over the training details of our agent and the training details of the compared methods. We also provide data preparation details and parameter values for each dataset.

ColorRL: We trained our A3C agent with 8 workers on 4 GTX1080 GPUs. The number of coloring steps that the agent perform for each data set is determined by the maximum number of neighbor instances of a single objects. An instance is a neighbor of another instance if their distance is smaller than a splitting radius r (e.g. when $r_1 = 10$ and $r_2 = 20$, two instances are each other's neighbor if their distance is smaller than 20). If the maximum number of neighboring objects of an instance is N_b, then we choose number of coloring steps T such that $2^{T-1} + 1 \geq N_b$. For 3D data set, we found that during the first few training iterations, overfitting the agent with a single volume will help the agent learn the full data set better. For the 3D Zebrafish dataset, we let ColorRL learn to color a single data volume for 300 steps (for each workers) before it tries to learn on the full dataset. To reduce the reward calculation time in the datasets of many objects and large image/volume size such as MoNuSeg and Zebrafish, we only calculate the rewards and the gradient update for a subset of the instances for each image/volume. To choose the instances for rewards calculations, we randomly choose an instance and chose a third of all the instances that are nearest to the first chosen instance. We used Adam optimizer to optimizer the network with a learning rate of 10^-4 (first 600k steps for each worker) and 10^-5 (last 800k steps for each worker). For post-processing, after the instances are separated using a connected component analysis, we remove the predicted segments that have small area (the best threshold for small area is chosen using validation set)

Other methods: For a comparison, we used Matterport's Mask R-CNN implementation [1]. For each data set, we modified the anchor scales, the number of maximum instances in the Mask R-CNN setting to get the best score on the validation set. As for ACIS [2] and E2E [8], we used the authors' implementations[2][3]. Both ACIS and E2E try to find a good order to predicting instances via the Hungarian matching algorithm. Their methods, however, consume a large amount of memory and processing time during training. Due to memory limitations, the maximum number instances per image was set to 90 for the training of E2E. For ACIS, we attempted to train the agent with different settings. We find that in order for ACIS to work with large number of objects (above 60), the number of prediction steps should gradually increases from 5 to 25 during training. With a greater starting value (above 5) or ending value (above 25) of the prediction step, ACIS will have trouble learning to perform segmentation. Both ACIS and E2E have no limitation on the maximum number of predictions during inference. For the 3D instance segmentation task, we train a U-Net model [9] for cell probability map and get the instance labeling results using two different methods: connected component analysis and watershed. With the watershed algorithm, we chose markers by using inverse distance transform on the cell probability map and find the local peaks on the maps. The minimum distance between peaks and maximum number of peaks was chosen so that the watershed algorithm yields the highest score on the validation set. With connected component analysis, with the validation set, we chose the best threshold value that transforms the probability map to a binary map. Evaluations of our methods and other methods were done using the same hardware for a fair comparison in processing measurement. To compute the average processing time measurement for each image, we measure the time from the time when an input image is finished loading to the point when the prediction is fully made (post-processing time is computed)

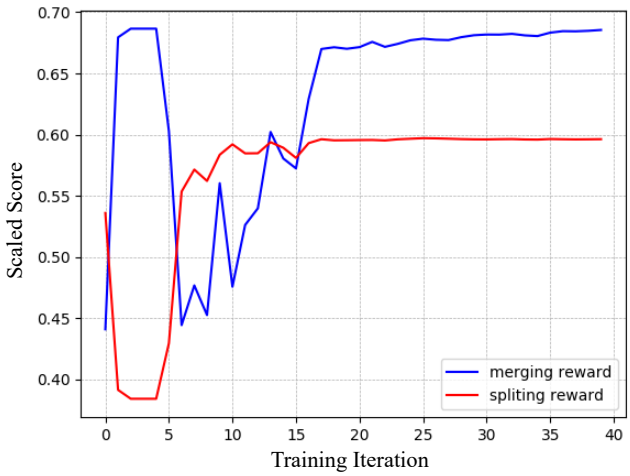

Figure 2. Illustration for the changing of scaled values of merging and splitting rewards over the training iterations.

A.5.2 CVPPP

We used the A1 dataset, which consists of 128 labeled images and 33 testing images. All the images are downsampled to 176 × 176 pixels (the original size is 530 × 530 pixels). Our

[2]https://github.com/renmengye/rec-attend-public
[3]https://github.com/visinf/acis

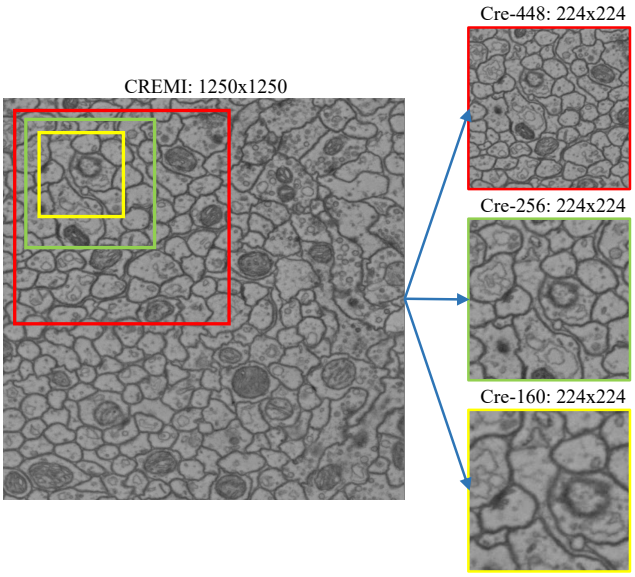

Figure 3. Illustration for preparation of the three datasets: Cre-160, Cre-256, Cre-448. The name and image size of each dataset are noted above each image and separated by a colon. Blue arrows indicate crop and resize.

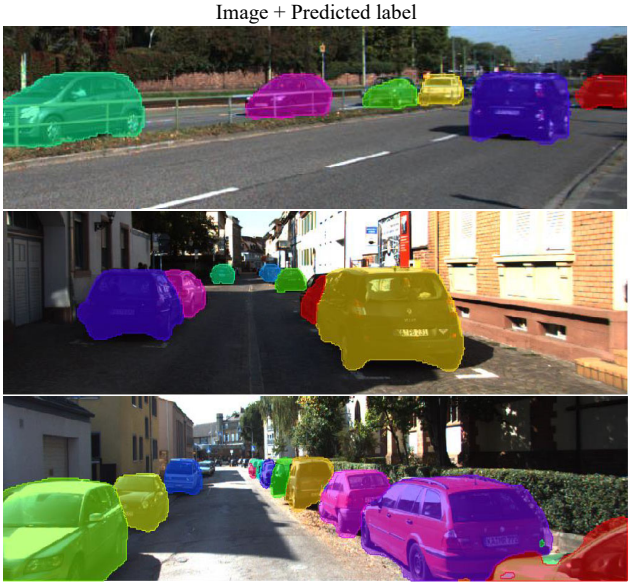

Figure 4. Coloring results KITTI test dataset. In this figure, we show predictions of KITTI after post-processing with connected component.

agent takes downsampled input images and produces output images of 176 × 176 pixels. We upsampled the prediction results to the original size (530 × 530 pixels) and then evaluated the test set. We empirically choose $r_1 = 12$, $r_2 = 28$, $w_m = 1.0$, and $w_s = 1.5$. The number of color steps T is 6.

A.5.3 KITTI

Table 1. Segmentation quality on KITTI testset. The metrics are mean weighted (MWCov) and unweighted (MUWCov) coverage, average false positive (AvgFP), and false negative (AvgFN) rates.

Model	MWCov↑	MUCov↑	AvgFP↓	AvgFN↓
DepthOrder [13]	70.9	52.2	0.597	0.736
DenseCRF [12]	74.1	55.2	0.417	0.833
AngleFCN+D [11]	79.7	75.8	0.201	0.159
E2E [8]	80.0	66.9	0.764	0.201
AC-BL-Trunc [2]	72.2	50.7	0.393	0.432
AC-IoU [2]	75.6	57.3	0.338	0.309
ColorRL	77.0	68.5	0.249	0.128

We let our agent perform coloring in 4 steps with $r_1 = 8$, $r_2 = 32$, $w_s = 1.5$, $w_m = 1.0$. The agent processes images of size 160×480 (the images are downsampled from the original size of 256×1024). We use 3712 training images, 144 validation images, and 120 testing images as in [8]. We also assess the performance of our method on the KITTI car segmentation dataset. Unlike CVPPP, cars in KITTI have a higher variance in size and position with noisy and low-quality training labels. Fig. 4, Fig. 5 and Table 1 shows that our agent can generalize well from noisy labels and is on par with the state of the art while outperforms existing sequential methods (E2E, AC-BL-Trunc, and AC-IoU). [2] (AC-BL-Trunc, and AC-IoU) is the most similar method to our method (they use an actor-critic agent to segment one object at a time). In E2E, AC-BL-Trunc, and AC-IoU, an object has to discriminate itself from all other objects in the image at a time step. Therefore, the single-object-per-step scheme requires the neural network to learn and process a lot of spatial information, making the segmentation problem more difficult when the size and position of objects vary. By letting the agent segment multiple objects at a time, the difficulty of the segmentation problem is reduced since each object only has to discriminate itself from the nearby objects that have the same color.

A.5.4 CREMI

CREMI is an EM image dataset in which many cell objects are densely packed and have very irregular shapes. From dataset A of CREMI, which consists of 125 1250×1250 images, we prepared three different datasets: Cre-160, Cre-256, and Cre-448. We randomly sampled 160×160 patches from the original dataset for Cre-160 and then resized all the patches to 224×224. We did the same for Cre-256 and Cre-448 (patches of size 256×256 and 448×448 are sampled respectively). Thus, each dataset has images of 224×224 but with different scales and average number of objects per image. Fig. 3 illustrates the generation of the three datasets. Each dataset has 123 training images and 25 testing images. We empirically choose $r_1 = 12$, $r2 = 24$ for Cre-160, $r_1 = 12, r_2 = 18$ for Cre-256, $r_1 = 8, r_2 = 12$ for Cre-448, and $T = 6, w_s = 1.5$ and $w_m = 1.0$ for all the datasets. We show additional results of Cre-256 in Fig. 6

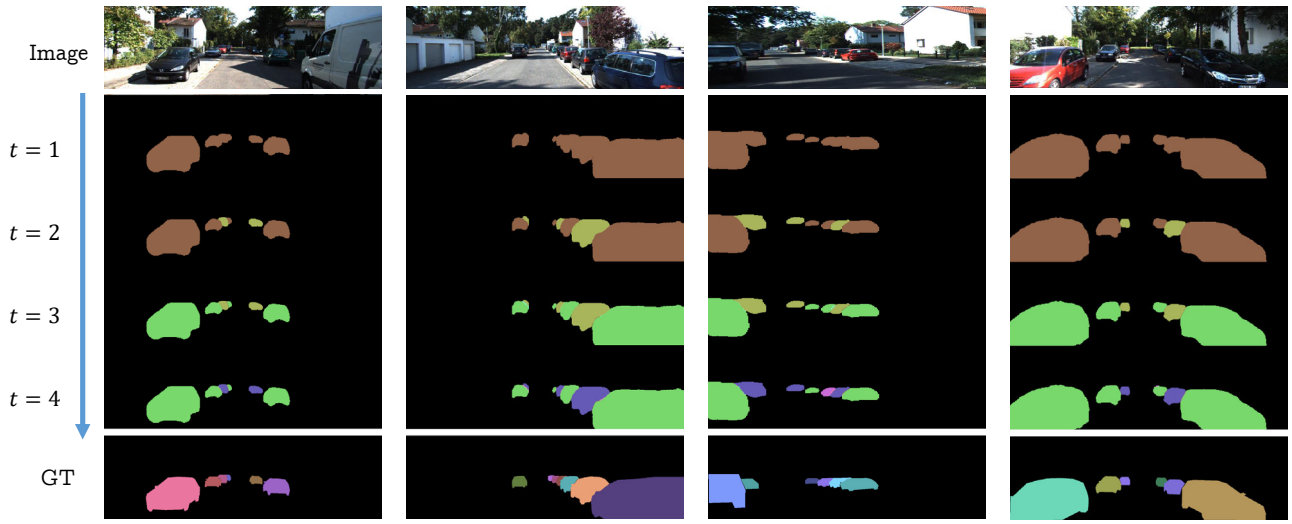

Figure 5. ColorRL's step-by-step results on KITTI.

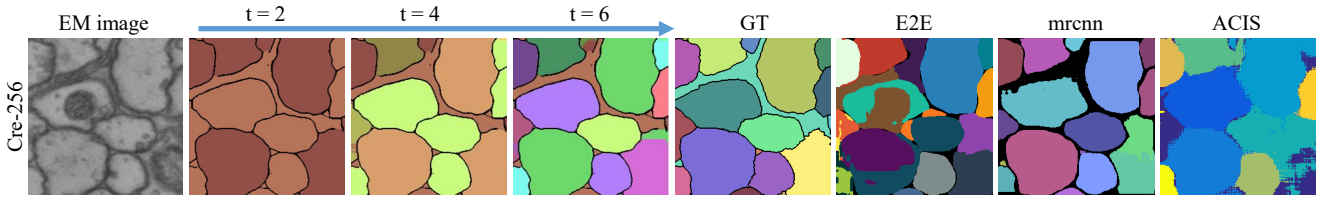

Figure 6. Segmentation results of Cre-256.

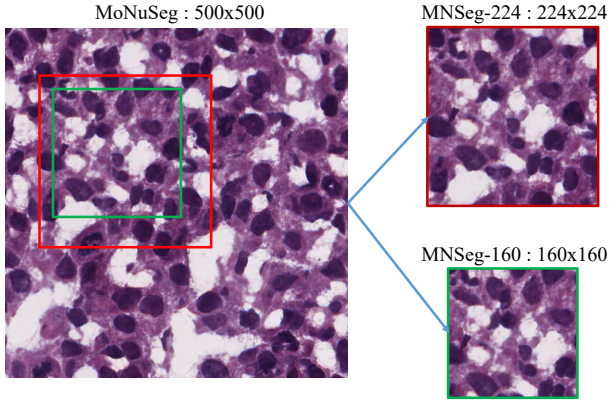

Figure 7. Illustration for preparation of the two datasets: MNSeg-160 and MNSeg-224. The name and image size of each dataset are noted above each image and separated by a colon. Blue arrows indicate crop.

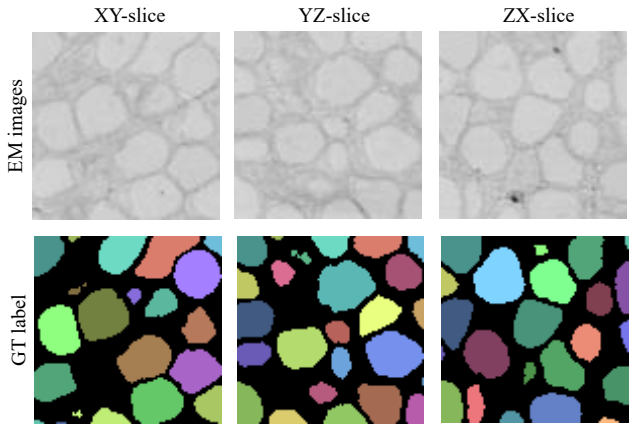

Figure 8. Visulization for XY, YZ, and ZX slice of Zebrafish data.

A.5.5 MoNuSeg

The dataset consists of 30 1000×1000 histopathology images acquired from multiple sites covering diverse nuclear appearances. The nuclei in MoNuSeg appear sparsely in each image; however, they are vastly different in size and shape from image to image. Before we created two versions of the data: MNSeg-160 and MNSeg-224, we downsampled all 30 images to 500×500. To create MNSeg-160, we sampled 600 (100) 160×160 patches from the downsampled images for the training (testing). Likewise, MNSeg-224 has 600 (100) 224×224 cropped images for trainning (testing). Fig. 7 illustrates the preparation steps for MNSeg-160 and MNSeg-224. The two versions of the data have the same scale but with different average number of objects per image and different image sizes. For both datasets, we chose $T = 7, w_s = 1, w_m = 1$, and $r = 10$. Due to memory limitations, E2E was trained and evaluated with a maximum of 90 instances. Therefore, there is not much difference in E2E's average inference time between MNSeg-160 and MNSeg-224.

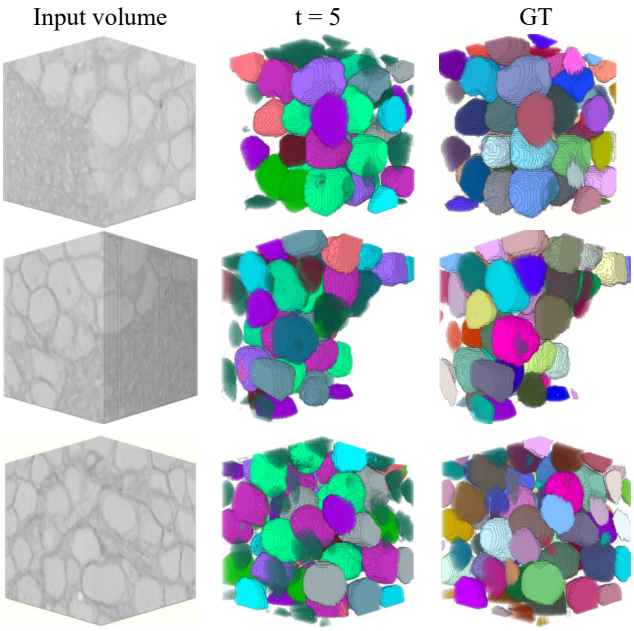

Figure 9. ColorRL's coloring results at $t = 5$.

A.5.6 Zebrafish

The larval zebrafish serial-section EM data were captured from a 5.5 day post-fertilization larval zebrafish. This specimen was cut into 18000 serialsections and collected onto a tape substrate with an ATUM. A series of images spanning the anterior quarter of the larval zebrafish was next acquired at a nearly isotropic resolution of 56.4×56.4× 60 nm^3 vx^{-1} from 16000 sections in the resulting serial section library using scanning EM [4]. We extracted 120 512×512×512 volumes that were manually labeled. We divided the 120 volumes into 80 training volumes and 40 testing volumes. All the volumes are down-sampled to 96×96×96 voxels. Fig. 8 show an example of slices of the 3D zebrafish data. Our agent processes full volume of 96×96×96 voxels with $T = 5, r = 12$, $w_m = 1.0$ and $w_s = 1.5$. Fig. 9 shows our additional results.

References

[1] Waleed Abdulla. Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. https://github.com/matterport/Mask_RCNN, 2017. 2

[2] Nikita Araslanov, Constantin A Rothkopf, and Stefan Roth. Actor-Critic Instance Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8237–8246, 2019. 2, 3

[3] Ryosuke Furuta, Naoto Inoue, and Toshihiko Yamasaki. Fully convolutional network with multi-step reinforcement learning for image processing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3598–3605, 2019. 1

[4] David Grant Colburn Hildebrand, Marcelo Cicconet, Russel Miguel Torres, Woohyuk Choi, Tran Minh Quan, Jungmin Moon, Arthur Willis Wetzel, Andrew Scott Champion, Brett Jesse Graham, Owen Randlett, et al. Whole-brain serial-section electron microscopy in larval zebrafish. *Nature*, 545(7654):345–349, 2017. 5

[5] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. 1

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 1

[7] Ozan Oktay, Jo Schlemper, Folgoc, et al. Attention U-Net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018. 1

[8] Mengye Ren and Richard S Zemel. End-to-end instance segmentation with recurrent attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6656–6664, 2017. 2, 3

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 1, 2

[10] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. 1

[11] Jonas Uhrig, Marius Cordts, Uwe Franke, and Thomas Brox. Pixel-level encoding and depth layering for instance-level semantic labeling. In *German Conference on Pattern Recognition*, pages 14–25. Springer, 2016. 3

[12] Ziyu Zhang, Sanja Fidler, and Raquel Urtasun. Instance-level segmentation for autonomous driving with deep densely connected MRFs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 669–677, 2016. 3

[13] Ziyu Zhang, Alexander G Schwing, Sanja Fidler, and Raquel Urtasun. Monocular object instance segmentation and depth ordering with CNNs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2614–2622, 2015. 3