Master's Thesis

Reinforced Proofreading of Image Segmentation for Connectomics

Khoa Nguyen-Tuan

Department of Computer Science and Engineering

College of Information-Bio Convergence Engineering

2021

Reinforced Proofreading of Image Segmentation for Connectomics

Khoa Nguyen-Tuan

Department of Computer Science and Engineering

College of Information-Bio Convergence Engineering

Reinforced Proofreading of Image Segmentation for Connectomics

A thesis submitted to
Ulsan National Institute of Science and Technology
in partial fulfillment of the
requirements for the degree of
Master of Science

Khoa Nguyen-Tuan

12/15/2020 of submission

Approved by

Advisor

Se Young Chun

Reinforced Proofreading of Image Segmentation for Connectomics

Khoa Nguyen-Tuan

This certifies that the thesis of Khoa Nguyen-Tuan is approved.

12/15/2020 of submission

Signature

Advisor: Se Young Chun

Signature

Committee Member: Won-Ki Jeong

Signature

Committee Member: Jae-Young Sim

Abstract

Manual connectome reconstruction is a challenging task because of large-scale image data, therefore, an automatic pipelines are needed. Recently, with the usage of deep learning in computer vision, automatic segmentations of electron microscopy (EM) image data are acquired but have the high error rates including merge and split errors, which means it still requires correction through human proofreading. In this thesis, I propose a novel fully automatic proofreading system for 2D segmentation base on reinforcement learning. By mimicking the human proofreading process, the proposed system uses Locator, Merger and Splitter agents for error detection and correction tasks. With an input segmentation image, the Locator agent detects erroneous patches on the input image and then feeds them to Merger and Splitter for correcting split and merge errors respectively. To showcase my system performance, I evaluate it on CREMI data set.

Contents

List of Figures

I Introduction

A brain is an organ that serves as the center of the nervous system in all vertebrate and most invertebrate animals [1]. In a human brain, the cerebral cortex is the largest part which contains approximately 14 $\sim$ 16 billion neurons; moreover, the second largest part, the cerebellum, is estimated number of neurons is 55 $\sim$ 70 billion [2, 3].

In particular, each neuron is connected by synapses to other neurons forms a network within the brain, which might be a hypothetical explanation for the functions of the brain – how the brain stores the memories, learns things and makes decisions in circumstances [4,5]. Additionally, these neural connections is called a wiring diagram or connectome; and the field of science, which is the production and study of connectomes, is called connectomics [6, 7]. Therefore, one of the goal of connectomics is fully accomplish the reconstruction map of neural connections to understand the meaningful which is hidden inside the brain structures [8–12]. The connectome reconstruction pipelines [12, 13] is done after finished proofreading process as shown in Fig. 1.

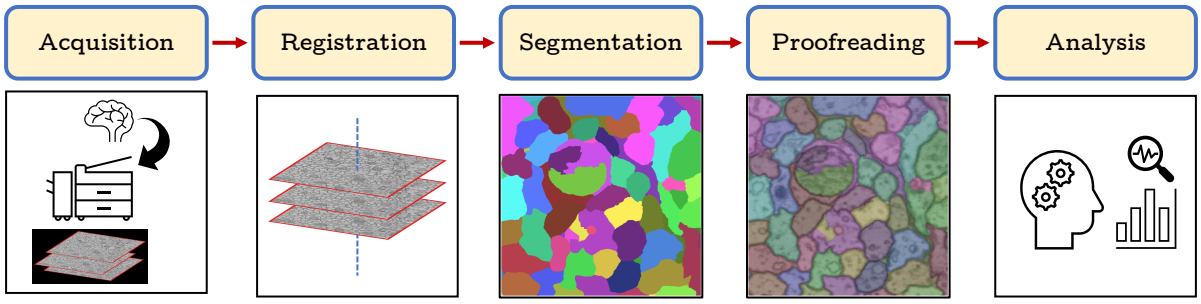


Figure 1: The typical connectomics workflow [12, 13].

To achieve this goal, neuroscientists produce high resolution (HR), nanoscale level, serial-section electron microscopy (EM) images that can capture the single neurons and single synapses. The brain tissues are cut into extremely thin sections, as small as 30 nanometers in thickness, and each pixel represents a 3 $\sim$ 5 nanometers [14–16]. With this pixel density, a cubic millimeter volume may costs approximately one petabyte of data; therefore, manual annotating of neurons is infeasible and automatic connectome reconstruction pipelines in Fig. 1 are required [9,13,17–19].

Recently, the convolutional neural networks (CNNs) have been widely used for automatic cell segmentation and classification [20,21], and then, base on the predicted boundary, using watershed transformation to generate the oversegmentation result [22–26]. Flood-Filling Networks combine two steps above into one by using predicted result as new input, which is the same as a recurrent model [27, 28]. However, even with those state-of-the-art methods, the segmentation result still has errors, especially merge error and split error (as shown in Fig. 3) because of various artifacts in EM images, such as noise, folding, and tearing due to the nature of the tissue preparation in acquisition process, which means the human proofreading process is still required [12].

With a massive data set, manual proofreading by visual inspection causes the bottleneck in

the automatic connectome reconstruction pipelines. Many works have been proposed to accelerate this process, the first approach provides interactive user interface to browse segmentation data in 2D and 3D, and to detect and manually correct errors [29–34]. The second approach is a purely automatic proofreading in 3D [35]. All the approaches above focus on correcting merge errors and split errors which are common errors occur in segmentation result.

1.1 Background

Reinforcement Learning

Two main characters in Reinforcement Learning (RL) are the agent and the environment. The agent here is a learner and takes action to the environment. The environment is the world that the agent exists and interacts with [36, 37]. At time step t, the agent in a state $s = S_t$ of the environment. If the agent is not allowed to observe the environment completely, then its partial observation is denoted as $o = O_t$; otherwise, the agent observe s [37]. After observed, the agent makes an action $a = A_t$. In the next time step $t + 1$, the agent receive a numerical reward R_{t+1} according to its preceding action, which means R_{t+1} denotes the reward for the action A_t. The goal of the agent is getting the total reward it collects over an episode which is all the things happening in between the initial state and terminate state. Therefore, we give a positive reward for a good action and negative reward for bad action through the feedback from the environment to reinforce the agent to develops a rule or policy. However, some reward require a long-term actions to get the final reward, which means it may has zero rewards in between and the agent have to plan to get the immediate rewards or wait for the big shot. The interaction from the agent and the feedback from environment forms a sequential decision making which can be modeled as the Markov Decision Process (MDP). In summary, the training process in RL is the interaction between agent and environment as shown in Fig. 2.

Markov Decision Process (MDP)

The MDP is basicaly defined by:

- A set of environment's states $\mathcal{S}$ ($S_t \in \mathcal{S}$). And $\mathcal{O}$ ($O_t \in \mathcal{O}$), if need.
- A set of action or an action space $\mathcal{A}$ ($A_t \in \mathcal{A}$).
- The probability of transition to state s', from state s taking action a : $p(s' \,|\, s, a) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\}$ with $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$.
- Reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$: $R_t = r(s, a, s')$. Or can be simply as state-action pair $r(s, a)$ or only state $r(s)$.
- A policy $A_t = \pi(S_t)$ is a mapping from states to actions. The agent uses this rules to selection action. If the agent is not fully observation, then $A_t = \pi(O_t)$.

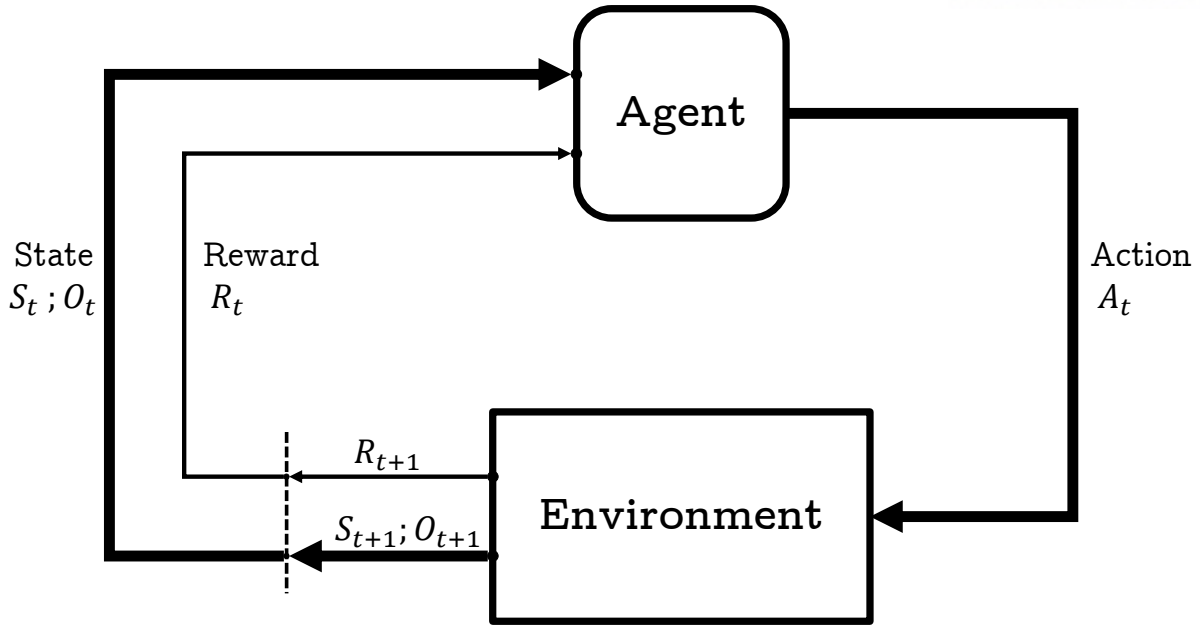


Figure 2: The agent-environment interaction in a Markov decision process (MDP). At time step t, given a current state S_t or an observation O_t, the agent selects an action A_t. In the next time step, $t+1$, the agent receives a reward R_{t+1} as a consequence of its previous action [36]. The observation O_t is used if the agent only can see a limited observation, if not, the agent fully observes the state S_t [37].

1.2 Problem and Motivation

Problem:

In general, as you can see in Fig. 3, the actions required for correcting the error are Merging and Splitting. Merging means we choose a pair of segments and then they will be assigned as a same label, and Splitting means we draw a boundary on a segment to cut it into two fragments. The way to correct the error is not hard; even a novice can correct it easily with a short training beforehand. However, with a huge dataset [14], manually proofreading is time consuming and costly. In fact, many works design applications [33, 34] for accelerating this task or using the community [38], hence, human are still required.

Motivation:

In recent years, the achievements of AI such as in gaming: AlphaGo [39], AlphaZero [40], AlphaStar [41], hide and seek [42] and Agent57 [43] ; or AlphaFold in biology [44], in autonomous driving [45, 46], and in robotics [47] are shown that AI can perform tasks as same as human-level. Inspired by these trends, I proposed a novel automatic proofreading system by mimicking human proofreading process which can correct merge errors and split errors.

The human proofreading process consist of two important tasks. The first one is identifying the errors in the overall context. Secondly, once an error is spotted, we focus on the error area

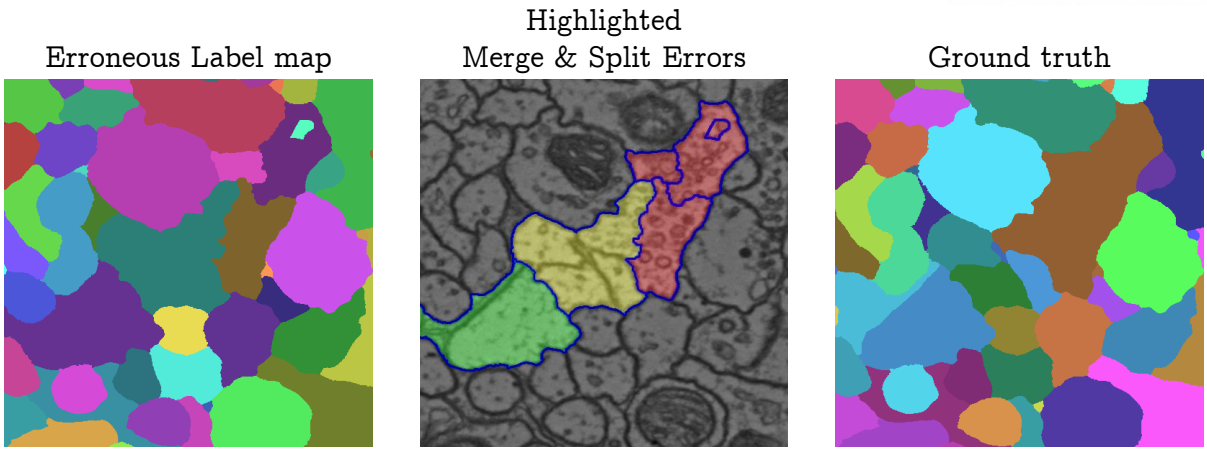


Figure 3: Illustration of merge and split errors. The merge errors are correct segments have been merged together in green regions, split errors are fragments of one or many correct segments highlighted as red regions, and yellow are the overlapped regions between two kinds of errors.

to figure out what is the problem and do either Merging or Splitting. Moreover, the process of locating error spots and applying updates on them requires decisions, which means these tasks can be naturally adapted to the reinforcement learning framework as follows:

- An agent for the detecting task.
- Two sub-agents for doing the Merging or Splitting actions.

In summary, given a good segmentation method, there is no guarantee that the result doesn't have any errors, and hence proofreading task is the one cannot avoid. However, proofreading by human slows the reconstruction pipeline so I would like to make it fully automatic by applying the reinforcement learning which can fulfil the simple human-level task.

1.3 Contribution

From the motivation above, I design three agents:

- Locator agent: with a low-resolution input image can locate the error locations.
- Merger agent: can detect and correct split error.
- Splitter agent: can detect and correct merge error.

In addition, the Locator agent tries to find the erroneous patches from a coarse resolution image just like we humans detect error spot in the overall context. Next, the erroneous patches are cropped from fine resolution image, and fed to Merger and Splitter agents for correcting split and merge errors respectively. Fig. 4 shows the overview of the proposed proofreading system.

Therefore, my contribution consists of three agents which built a fully automatic proofreading system.

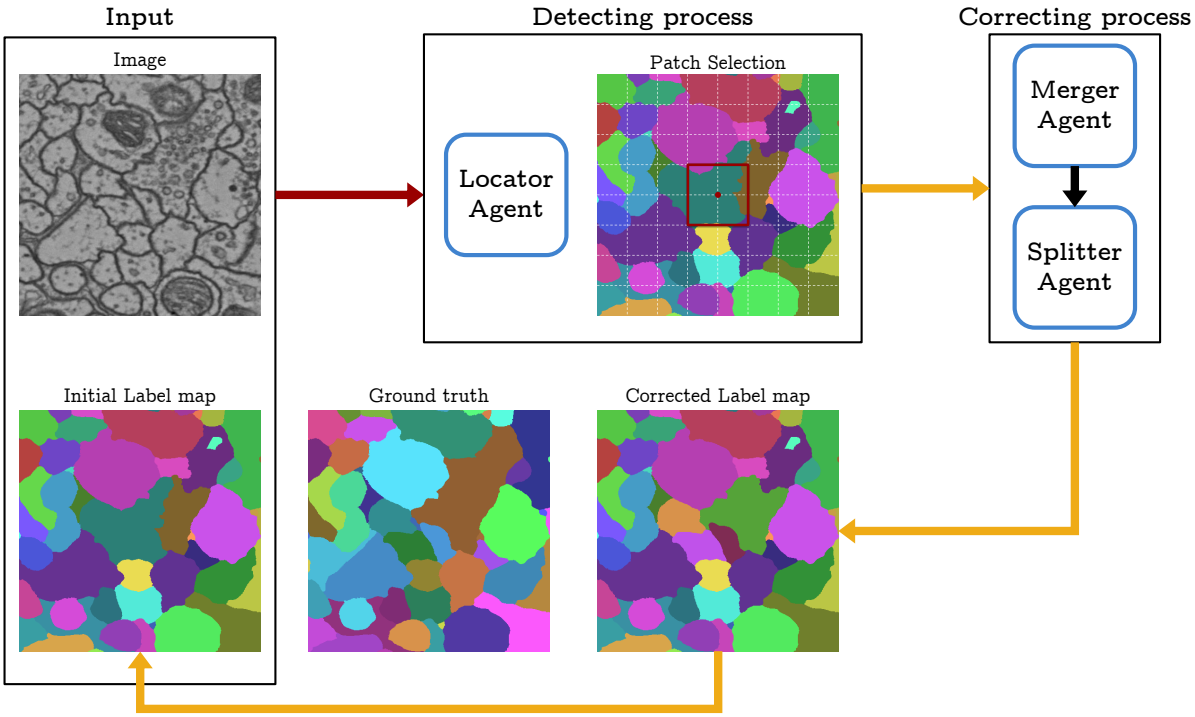


Figure 4: The proposed proofreading system diagram. From the beginning, an input consists of an initial label map and an EM image, the Locator agent choose a coordinate of a patch containing error regions base on an action grid. The chosen patches cropped from image and label map are fed to Merger agent to correct split errors, after that Splitter agent corrects merge errors on the result adopted from Merger agent. The correct label patch is pasted back into the label map, and those segments outside of the patch which connected to prior erroneous segments inside are also relabeled by watershed transformation. The initial label map is updated from the corrected label map for the next iteration.

I'd like to verify this, I don't create new method in the reinforcement learning. I only use reinforcement learning as an approach for my application by adapting the Markov Decision Processes

II Related Work

2.1 Automatic Segmentation

General automatic segmentation methods [22, 23, 48, 49] are used for reconstructing from vast EM brain volumes. Also, a decade ago, CNNs have been applied to detect the boundary of neurons [50,51]; after few years, the U-Net [20], as known as a baseline network for segmentation task till now, is proposed and led to many variant of its designs [21, 52–54]. Then, in the last few years, these state-of-the-art methods [24–27] came with the approaches using both CNNs, to predict the affinity or to classify boundary, and agglomeration to refine after watershed transformation. However, the challenges [16, 55] showed that, because of the uncertain distorted areas on the images, it's still hard to achieve a flawless result without human proofreading process.

2.2 Human Proofreading

In fact, with multi-terabyte EM image data set, it's time consuming to proofread physically.

Interactive Proofreading. In order to speed up the procedure, many interactive tools have been developed to contain a visualization and interaction system, which support user to interact the data in 2D and 3D. Sicat et al. [29] built an AVIZO plugin, which generates a graph abstraction for simplifying proofreading task. A skeleton constructed from the initial segmentation volumes is used to identify the potential erroneous regions and guide users to it. Raveler [56, 57] is one of the tools in Janelia Farm FlyEM project, which supports annotation and proofreading on large data set. Nevertheless, with many parameters for fine-tuning, it's only compatible for professional users. A game platform, EyeWire [38], which adapts the proofreading task as a puzzle game and participants earn virtual rewards after accomplished the task. Mojo [30] and Dojo [31] are simple tools which use scribbles to correct the errors. Dojo is an extended version of Mojo which aims to effortless interaction by running on a minimalistic web-based user interface.

Computer-aided Proofreading. Uzunbas et al. [58] detected potential erroneous labels by training a conditional random field to track the underlying uncertainty in the merge tree of an automatic segmentation method. But this work is only applied to isotropic volumes that certainly needed to be improved for anisotropic volumes. Zung et al. [35] introduce detection and correction framework by an error-detecting net and an error-correcting net. The error-detecting net detects merge and split errors given a binary mask of candidate object, then its output is a map containing the locations of merger and split errors; moreover, the output map is considering as an advice to the error-correcting net by taking all the union of all erroneous segments have been found by the error-detecting net which made the framework fully automatic. However, the error-correcting net only does the object mask pruning task, which means it can correct only merge errors while the error-detecting net detects both merger and split errors; therefore, there

are constraints in their framework and it is not as same as human correction behavior. On top of this, their framework works on 3D volumes, while my system works on 2D slices.

Currently, the work done by Haehn et al. [34] is the state-of-the-art in proofreading task. The authors showed a human-guided proofreading approach, which is called guided proofreading, to correct merge and split errors base on the boundary of segmentation on an error patch, and then displays the suggested corrections as a yes/no decision for users. In order to detect split error, they proposed a CNN-base boundary classifier or split error detection, which outputs the probability of the given boundary mask of two adjacent segments in the error patch. Likewise, to detect merger error, they reused the trained split error detection and inverted the probability result given a set of boundaries generated from a segment. In other words, given a segment, they randomly placed pairs of symmetric seed points on the boundary of the dilated the segment to generate a set of potential boundaries by watershed transformation; then, they used the inverse probability result to rate individually boundary. To get the error patches, they inspect all the segments, to find merge error, and boundaries of neighbouring segments, to find split error, on the segmentation in a brute-force manner; next, all the probability results from two detectors are sorted into a list of rankings which will be looped from top by users. Besides, they also defined a threshold to run automatic selection. However, the result showed the amount of correct corrections is lower than the total. After all, their approach is encapsulated in human correction task and still requires human labor. Actually, I'm inspired a lots from [34]. For merge error correction, dynamic decision-making can take part in, which means instead of placing randomly a pair of points, which creates a boundary and cuts 2 segments have been merged, my Splitter agent can decide the optimal locations to place seeds; therefore, my Splitter agent not only reduces a large amount of effort but also can separate more than 2 segments.

2.3 Reinforcement Learning

Song et al. [59] suggested an interactive segmentation method called SeedNet. In terms of human interaction, it can be useful in the process of connectomics. On top of that, the authors considered the step-by-step segmentation as a decision process, so the idea is implemented on the Reinforcement Learning framework. In contrast, my work is different from the perspective of methodology and goals. However, we shared some ideas such as the definition of current status, and modifying reward function for the Reinforcement Learning framework.

In order to follow the human detection task, it is natural to make a series of decisions such as zoom-in and zoom-out, which leads to the use of the reinforcement learning framework. Utilizing selective zooming-in [60–62] has been suggested in order to perform object detection or segmentation efficiently on large sized images. Trained policy works efficiently on choosing which area to zoom in, which makes these approaches not only reduce computational complexity but also can be considered as human identify actions. However, those approaches are for the semantic segmentation or classification, while mine is for proofreading on the segmentation slices.

To the best of my knowledge, this is the first fully automatic proofreading method mimicking

the behavior of human annotators using a reinforcement learning approach for connectomics.

III Method

From Fig. 4, all the agents observe and take action on image. If I define each pixel as an action, it may requires a huge action space containing pixel-wise actions; and also, the scope of this thesis is limited in discrete action space so I don't use continuous action space. Therefore, action space of the agent is the set of the intersections on a designed grid and an additional terminate action. In other words, I proposed a grid-action map method for applying into all agents. It's similar with [59] but my agents have a terminate action while theirs don't. The details is defined later in the section below.

3.1 Merger agent

The goal of the Merger agent is correcting the split errors highlighted as red regions ,and also the overlapped regions in Fig. 3 . In general, the human correcting split error process has two basic steps. The first step is find and pick a split fragment. The second step is find and pick another split fragment corresponding with the preceding fragment. After that, a pair of split fragments have been merged into a new segment which is a correct segment or still a fragment of a larger correct segment. By repeating this process, till the end, we will fix all the split errors. I design Merger agent following this process as a MDP (as shown in Fig. 7).

State: The state contains the representations of the environment, which consists of an EM image I, a label map L and a point map P. Our agent interacts with the environment base on coordinates so it needs a map representing the chosen locations. Therefore, a Point map is proposed ($P \in [0, 1]$), through all episodes, after the agent taken an non-terminate action A_t, the Point map at time step $t+1$, P_{t+1}, is inserted a Gaussian kernel $g \in [0, 1]^{16\times16}$ corresponding to the area surrounding the chosen coordinate. The label map ($L \in \mathbb{N} \setminus 0$) is the result after applied the watershed transformation into the probability map which is the result of prior automatic segmentation process. The Merger agent observes the normalized label map $\hat{L} = \frac{L}{l_{max}}$ which is divided with a maximum index on label map. The size of both the state and the observation are 128×128, in this case, after measured the maximum number of segments which can be contained, and excluded the background's value 0, I choose $l_{max} = 99$. And also, at the initial state $t = 0$, all the index of segments in label map L_0 are shuffled with range $[1, l_{max}]$ in order to prevent the model observing the same index over a region repeatedly which makes the model more robust. However, observing the $\hat{L}$ may not enough information for the agent to distinguish the segments. In other words, I want the agent observe the label map as a cell instance segmentation map from the index value. But after normalized, the index value in L has been scaled down from range $[1, l_{max}]$ to $[0, 1]$ and the step between two indices are scaled down l_{max} times. There for one label map may not fully represents a dense cells. The position encoding method in natural language processing [63] suggested enhancing the model's input by equipping each word with its position in a sentence, which means the position is encoded into a wavelength d-dimensional vector. Inspired by the position encoding method, I encode the label map $L \in \mathbb{N}^{H\times W} \rightarrow \ddot{L} \in \mathbb{N}^{H\times W\times d}$

and the value in L is calculated by a base B as follows:

$$\begin{aligned} \sum_{i=1}^{H}\sum_{j=1}^{W} L_{i\times j} &= \sum_{i=1}^{H}\sum_{j=1}^{W}\sum_{k=1}^{d} \ddot{L}_{i\times j\times k} * B^{k-1} \\ or\ L &= \sum_{k=1}^{d} \ddot{L}_k * B^{k-1} \end{aligned} \tag{1}$$

where k denotes index of the last axis in $\ddot{L}$. For Merger agent and Splitter agent, I choose $d = 2$ and $B = 10$, Eq. 1 becomes:

$$\begin{aligned} L &= \ddot{L}_2 * 10^1 + \ddot{L}_1 * 10^0 \\ &= \ddot{L}_2 * 10 + \ddot{L}_1 \end{aligned} \tag{2}$$

In this case, the maximum index in $\ddot{L}$ is $l_{max} = B^d - 1 = 10^2 - 1 = 99$. Fig. 5 shows how this encoding method operates on a 3×3 label map.

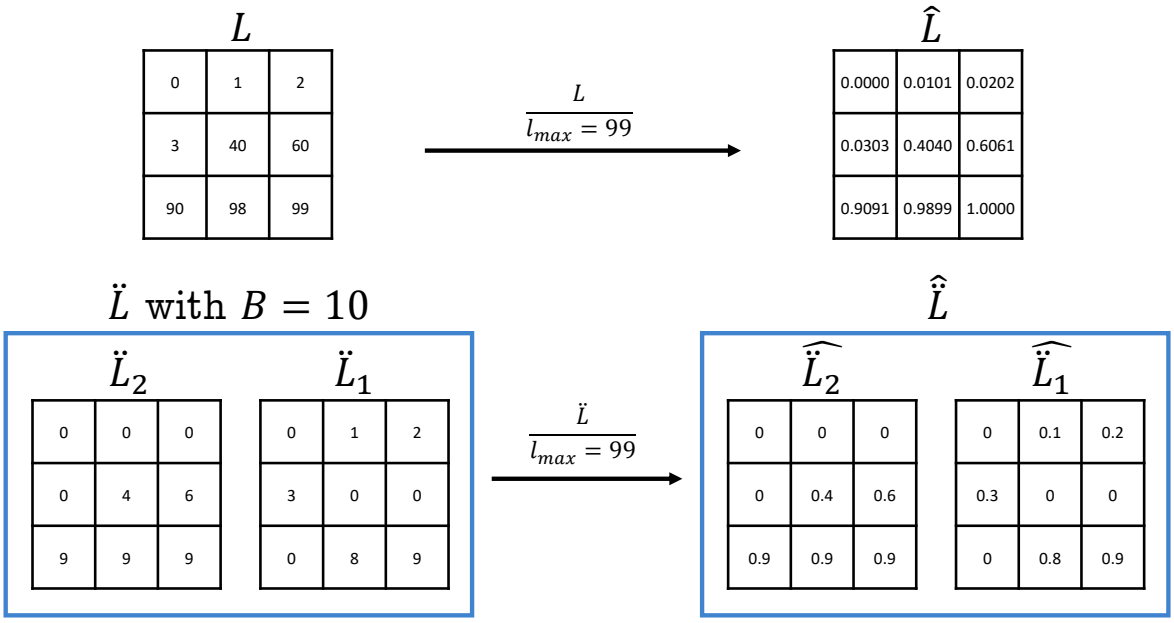
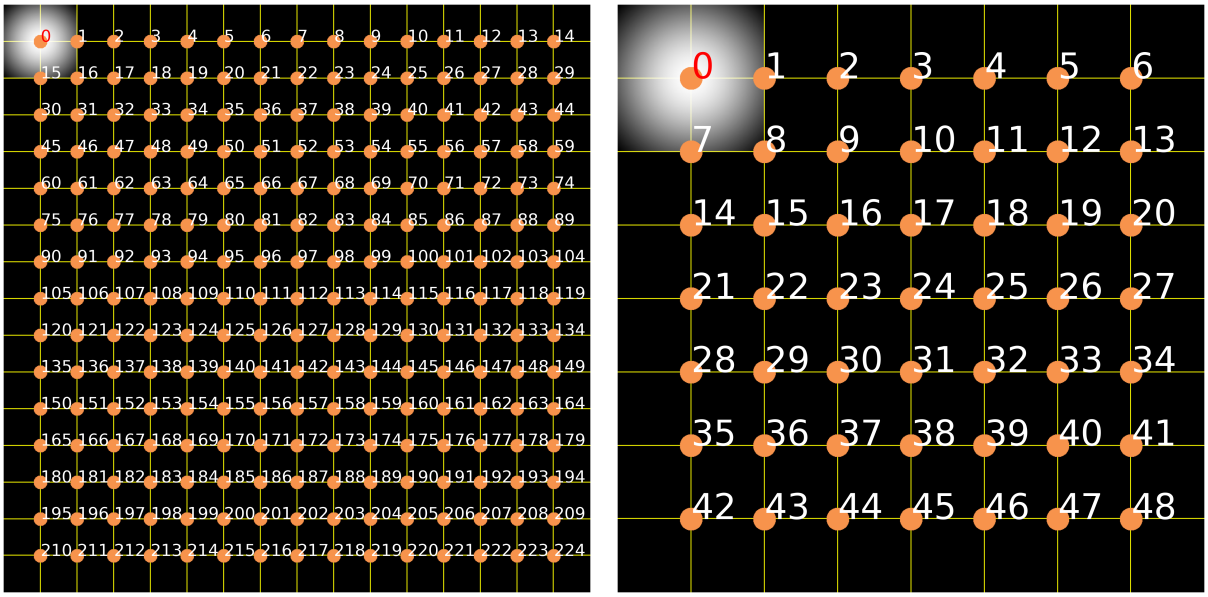


Figure 5: Illustration of encoding and normalizing result on a 3×3 label map. The $\hat{L}$ is the normalized map of L and $\ddot{L}$ is encoded map of L.

In the end, the Merger agent observes $\hat{L}$ or $\hat{\ddot{L}}$, the EM image $I \in [0, 1]$ and Point map P to detect and correct split errors.

Action: Given a state, in order to pick a fragment on L, an non-terminate action is defined as coordinates of intersections on a defined 2D grid excluding the intersections on boundary, which means, from a $(n+2) \times (n+2)$ grid, the action space consists of inner $n \times n$ actions and an additional terminate action. To make it's more precisely, I choose 15×15 actions, thus, the grid-action is dense and the total number of actions is 226 (see Fig. 6a).

Additionally, in the time step t, with action A_t, the chosen coordinate is c_t. Then, the label on the chosen coordinate is $L(c_t)$.



(a) The overlap of grid-action map 15×15 and the Point map in Merger, Splitter agents. The terminate action has index 225 and the size of the Gaussian kernel is 16×16 pixels.

(b) The overlap of grid-action map 7×7 and a Point map in Locator agent. The terminate action has index 49 and the size of the Gaussian kernel is 32×32 pixels.

Figure 6: Illustration of the grid-action map (yellow lines, orange dot and white numbers), and how a Gaussian kernel is inserted at action $a = 0$ (the red number) in the Point map. The orange dots indicate where the agent can do action and the number is the index of non-terminate action in the action space (zero-based numbering). The image has size 128×128 pixels.

Reward: The reward is the feedback of the environment. By designing a correct reward function, we can guide the agent to learn our desire task which is improving the metric value. In this case, for a simple task, given only a correct segment has been split into multiple fragments, I want the Merger agent merge all the split fragments into the correct segment again. Thus, I can simply design the Merger agent receive positive reward if it picked a erroneous fragment and negative reward if the picked. The reward function is as follows:

$$R_{t+1} = \begin{cases} 1 & \text{if } L(c_t) \in E_t \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

where E_t is the binary mask of erroneous fragments at time step t, and R_{t+1} is the reward receive in time step $t+1$, which is 1 if the picked label is matched with the error binary mask and vice versa, for the action c_t at time step t . In fact, at the odd time step, the picked label indicates as a base label; then, in the next time step, or the even time step, the next picked label is changed its label to base label, which is as same as the way human pick and merge a pair of segments (see Fig. 7). Given a correct segment has been split to n fragments, the number of time step need to fix it is $2n-2$ steps. Follow this scheme, a pair of segments is merged without any awareness. Therefore, the agent have to learn not only how to correct split error but also learn how to avoid penalty from merging a correct pair of segments. All of the change-makings at each time step are updated directly on L. Since I have the ground truth ($GT \in \mathbb{N} \setminus 0$), I can use it as a reference for checking picked pairs; thus, a pair is merged only if it's a pair of erroneous fragments. In this additional option, the Merger agent only see the combining of two erroneous fragments and nothing changes when it receives a penalty. In summary, I design 4 Merger agents with follow configuration:

	Observe L	Observe $\ddot{L}$
Environment only merges erroneous segments	M1	M2
Environment merges a pair without any awareness	M3	M4

3.2 Splitter agent

The task of the Splitter agent is correcting the merger errors highlighted as green regions, and the overlapped regions in Fig. 3. The human correcting merge error process is drawing the boundary on the merge error segment, which means we cut a segment into smaller segments by drawing boundaries. In fact, one simple approach is using a refine model to re-segment the label map again. However, a merge errors happened due to ambiguous areas on the EM image which makes the prior segmentation process can't generate complete boundaries . Therefore, there is still no guarantee that using new segmentation model can achieve a better result if the prior segmentation process is already at the best condition. To overcome this problem, [34] proposed generating random boundaries by the watershed transformation base on the EM image, which

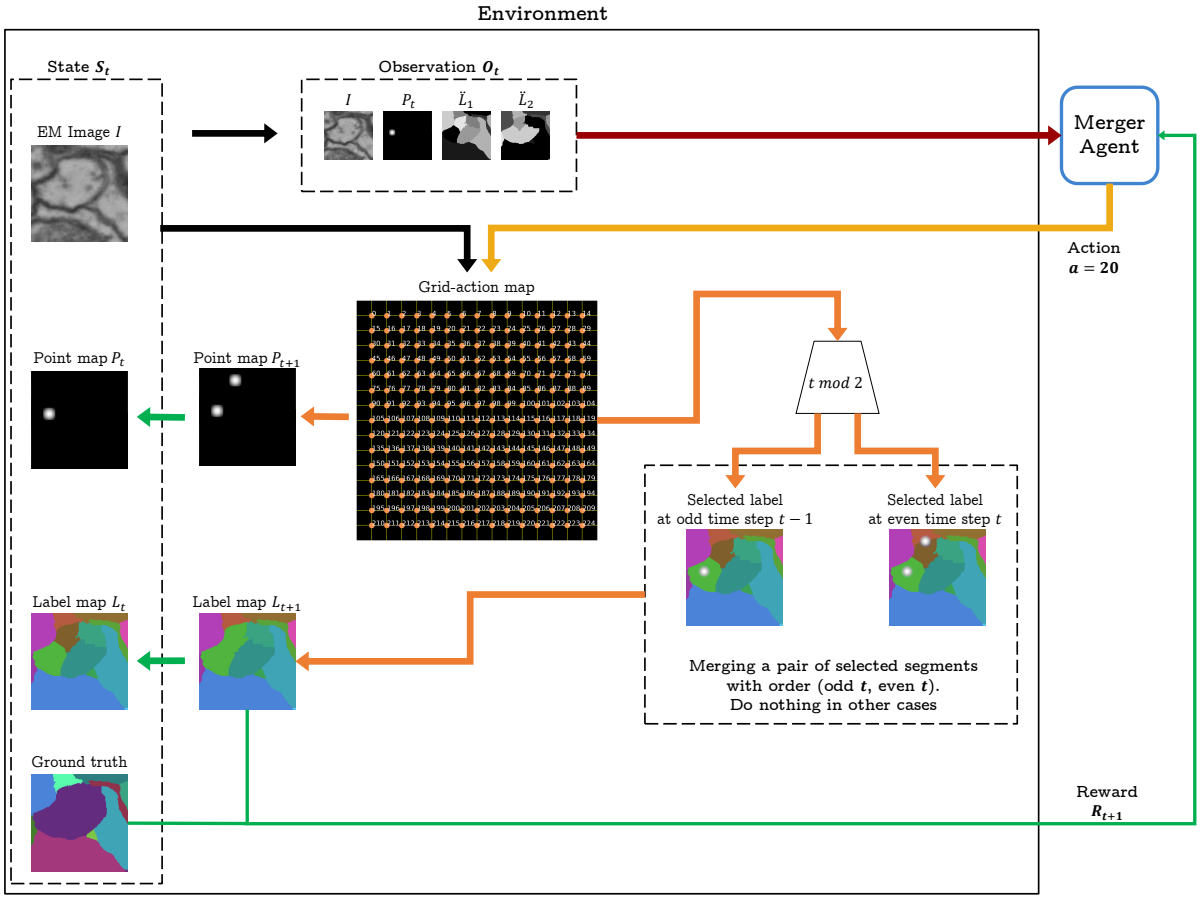


Figure 7: The interaction in Merger agent. In a time step t, The black arrow is the flow of the state S_t; the red arrow is the input observation O_t to the agent; the yellow arrow indicates the action of the agent for the corresponding O_t, hence, $a = 20$. The grid-action map gets the state S_t and read the action a; after that, the Point map P_{t+1} is inserted a Gaussian kernel on a coordinate of the action index. The orange is the image processing flow of the environment. In this case, a gateway module is checking whether this time step is odd or even. If it's odd, then the environment saves the chosen label. If it's even, the environment looks up the previous chosen label in the odd time step and change the current chosen label to the previous chosen label respectively. The green arrow indicated the updating flow for the next time step and also calculating the reward for the action a. The reward calculation Eq. 3 in both kinds of time step is the same, the reward is only either this chosen label is correct or not.

means we use the natural shape of the cells to to get the boundaries on ambiguous areas. I follow [34]'s method to split a segment, but instead of using random locations as markers for running the watershed transformation and relying on a rating model, my Splitter agent will learn and find the optimal locations respectively.

State: The state in Splitter agent is the same with Merger agent, which means the Splitter agent observes $\hat{L}$ or $\hat{\hat{L}}$, the EM image $I \in [0, 1]$ and Point map P to detect and correct merge errors. Even they observe the same view, but their environment operates different with each other. While the Merge agent doesn't create new cell instance or new index value in the label map L, the Splitter agent learns to choose optimal coordinates to generate precise boundaries and new cell instances or new indices are added to the label map L after the watershed transformation.

Action: Given a state, the Splitter agent chooses coordinates to be markers for the watershed transformation process, which means the coordinates are represent as a grid-action 15×15. Therefore, the action space of Splitter agent is the same with Merger agent. To operating the watershed transformation method, a marker map $M \in [0, 1]$ is created from the initial state. All the chosen pixels on M, corresponding with chosen coordinates, are set value, which means $M(c_t) = t$ with $(t > 0)$. Another one needed for the watershed transformation is the value map V. The the watershed transformation using V as a topographic surface and flood this surface from its minima. In general, $V = invert(I)$, which means the high values are considering as catchment basin and low values like background are watershed line, but due to the artifact on EM image, I pass I into a Gaussian filter ($\sigma = 2$) and normalize the result so that $V \in [0, 1]$. However, only using Gaussian filter is not enough blurring the boundary of the cell body inside the neuron, while we are considering the cell body is the same part of the neuron here. Therefore, V is added the Gaussian kernel g on the area surrounding the chosen coordinate: $V(c_t) = g$ By this scheme, the Splitter agent not only learns to find the split errors but also it learns placing the point in order cut the merged segments precisely. Fig. 8 shows how it corrects the merge errors.

Reward: Base on the task of the Slitter agent, the total reward comprise two kind of reward: detecting reward and metric reward. The detecting reward R_d is similar with the reward function in Merger agent:

$$R_d = \begin{cases} 1 & \text{if } L(c_t) \in E_t \\ -1 & \text{otherwise} \end{cases} \tag{4}$$

where R_d is the detecting reward at time step $t+1$ for the action c_t at time step t and E_t is the error map contains the binary mask of merge errors at time step t. To encourage the Splitter agent to find the optimal coordinates, the metric reward can represent as the improvement metric value between two time steps. According to [59], the gap between two metric values are not enough. Since our data set is CREMI so I use the CREMI score metric, which is defined as a geometric mean of $(VOI_{merge} + VOI_{split})$ and the $ARAND$ [16, 64]:

$$CREMI = \sqrt{(VOI_{merge} + VOI_{split}) * ARAND} \tag{5}$$

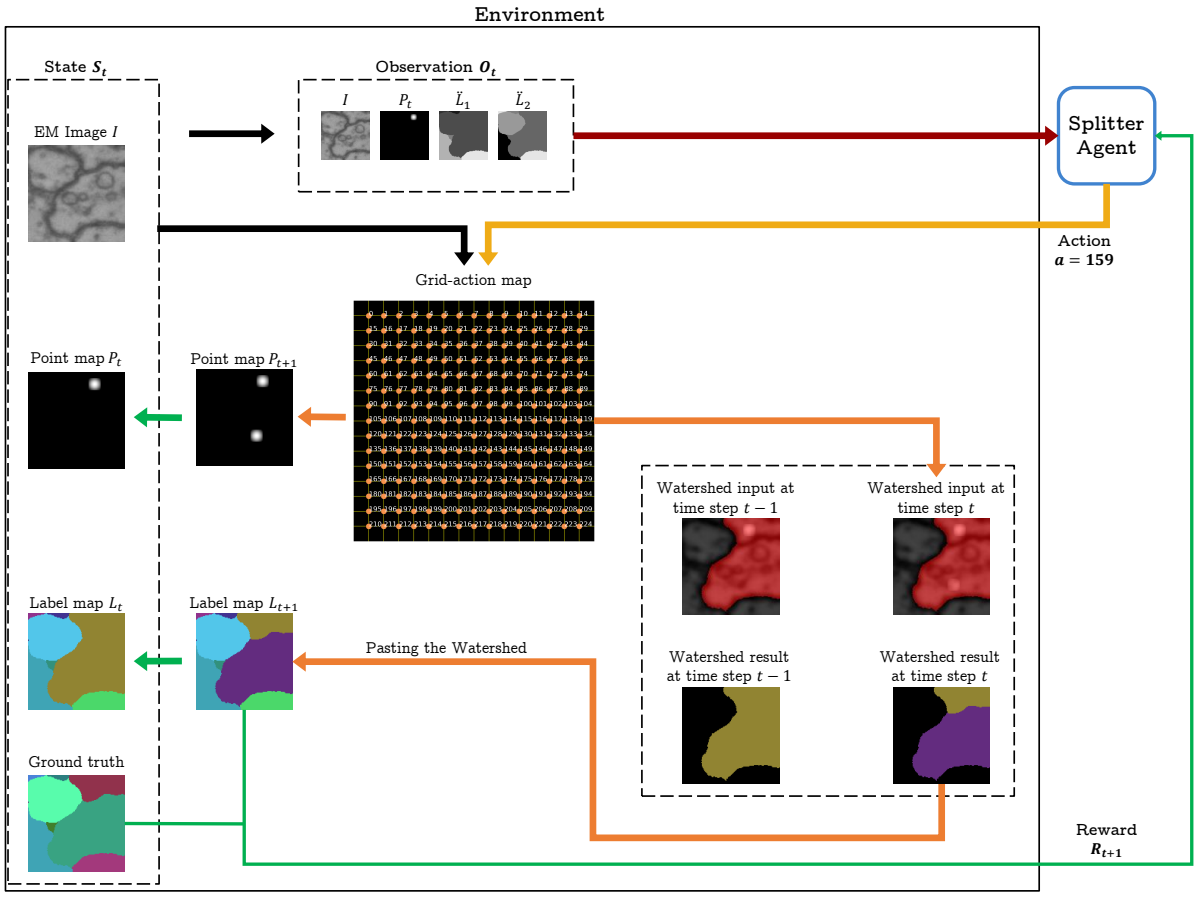


Figure 8: The interaction in Splitter agent. In a time step t, The black arrow is the flow of the state S_t; the red arrow is the input observation O_t to the agent; the yellow arrow indicates the action of the agent for the corresponding O_t, hence, $a = 159$. The grid-action map gets the state S_t and read the action a; after that, the Point map P_{t+1} is inserted a Gaussian kernel on a coordinate of the action index. The orange is the image processing flow of the environment. In this case, the image I is passed through a Gaussian filter and summed with the Point map P_{t+1}, then the inverted result and the mask of chosen labels ,the red color, are used as the input for the watershed transformation. After that, the region in label map L_{t+1} is updated base on the watershed result and the red mask. The green arrow indicated the updating flow for the next time step and also calculating the reward Eq. 7 for the action a.

The VOI_{merge} and VOI_{split} are used to measure the merge and split errors respectively, while $ARAND$ computes Adapted Rand error as defined by the SNEMI3D contest [55,64]. The lower the score, the better the result, which meanss it opposite with the IoU metric. Therefore, I design a function for metric reward which makes the agent receive more reward if the CREMI score is lower and lower in each time step:

$$R_{log} = -\log(\frac{CREMI(L_t, GT)}{CREMI(L_{init}, GT)}) * w \tag{6}$$

where R_{log} is the received reward as time step $t+1$ for the action A_t. The L_t and $L_{initial}$ is the label map at time step t and at the initial state respectively. w is the constant value, in this case I choose $w = 2$. The total reward at time step $t+1$ for the action A_t at time step t is as follows:

$$R_{t+1} = \begin{cases} 1 + R_{log} & \text{if } L(c_t) \in E_t \\ 0 & \text{if CREMI has been equal 0 once: } \Pi_{t_i}^{t-1} CREMI(L_{t_i}) = 0 \\ -1 & \text{otherwise} \end{cases} \tag{7}$$

During the training phase, if the agent corrects all the split errors, if it keep does more actions, the label map will have split errors, and also, I can't give the penalty for this case because it makes the agent misleading. Therefore, I set all the later reward to 0 as a signal to prevent the agent doing redundant actions. In summary, I have two kinds of Splitter agents:

- S_1 is a Splitter agent observing $\hat{L}$.
- S_2 is a Splitter agent observing $\hat{\hat{L}}$.

3.3 Locator

In the general case, an error segment is not easily distinguished as a stand-alone merge error or a stand-alone split error, thus it can be both like the region highlighted as yellow in Fig. 3. Running only or both pre-train Merger or Splitter agents on that area is not enough to have a flawless result. There is a case we have to revisit that area again. Therefore, I propose Locator agent, which mimics the human detecting error task. The task of the Locator agent is finding error areas within high resolution label map L from a coarse resolution view. The Locator agent also has to be aware of the improvement of the label map such as if the result from Merger and Splitter is not good, the Locator has to consider to revisit the preceding error area again or not. In fact, by observing the coarse resolution image, the memory for parameters in the Locator agent is saved.

State: The state in Locator agent is the same with Merger and Splitter agent but the observation's size is different, which means the Locator agent still observes $\hat{L}$ or $\hat{\hat{L}}$, the EM image $I \in [0, 1]$ and Point map P, but the $\hat{L}$ or $\hat{\hat{L}}$ are computed from a downscaled label map, in this case, 4 times. The EM image I is also downscaled from its original input size. The

observation's size can be also known as the input to the agent's model. From now, for definition,I will call the observation's size as coarse resolution or low resolution (LR), and the size of the stage in the environment as fine resolution or high resolution (HR); and also, the error patch P, which is the patch cropped from HR image with a size the same in Merger and Splitter agents, hence, the correcting size or cropping size = 128. Additionally, with the larger size of HR label map requiring larger capacity index or larger l_{max}; therefore, I choose the encode base $B = 20$ so the $l_{max} = 399$. The Gaussian kernel in Locator is $g \in [0, 1]^{32\times32}$ because of the different grid-action map.

Action: Given a stage, the Locator observes through its coarse resolution view and try to identify the error area. The action space of Locator is the same grid-action map of Merger and Splitter agents but different size because the task now is viewing from the overall context so I don't need a dense grid anymore, hence, I choose 7×7, so now the total actions of the action space is 50. A patch is cropped from HR image with the correcting size and the chosen coordinate as the center. Then, this patch is firstly fed to a pre-train Merger agent for detecting and correcting split errors. Next, the Slitter agent obtains the result patch from Merger agent and do the rest. The corrected patch is pasted back into to label map L I choose this correcting order because I trained the Merger agent only with one split error case, which means the Merger agent may picks wrong segments and produces merge errors. That is the reason why the Splitter agent is run after, it can cover the Merger agent if Merger agent accidentally generated merge errors. Fig. 4 is the summary of how Locator work and Fig. 9 shows its interaction. Nevertheless, if the Splitter agent generates split errors, the Locator has to know it and revisiting the previous area again. From this scheme, a loop may exists if the two agents keep generate and fixing each other errors; in this case, the Locator still receive the rewards but it made the Locator agent only revisits the same location again and again forever, which is what I want to avoid.

Reward: Again, the task of the Locator is locating the error areas and those chosen areas have to improve the metric of the HR label map. It's the same case with Splitter agent, I want the Locator not only able to find error areas but also fully aware of the improvement metric. However, the R_{log} in Splitter agent encourages the Splitter agent with the lower the CREMI score, the much higher reward, but it is not sensitive with small improvement from the beginning, so I can't apply the same case to the Locator agent. Because the large size of the label map, after corrected the error patch, the improvement score is low so R_{log} return a low immediate reward. Therefore I design the function which is similar with R_{log} but sensitive with the small change in CREMI score.

$$R_{exp} = \frac{\exp(w * (\frac{-CREMI(L_t)}{CREMI(L_{init})} + 1) - 1)}{\exp(w) - 1} \tag{8}$$

where w is a constant. There for the reward at time step $t + 1$ for the action A_t at time step t

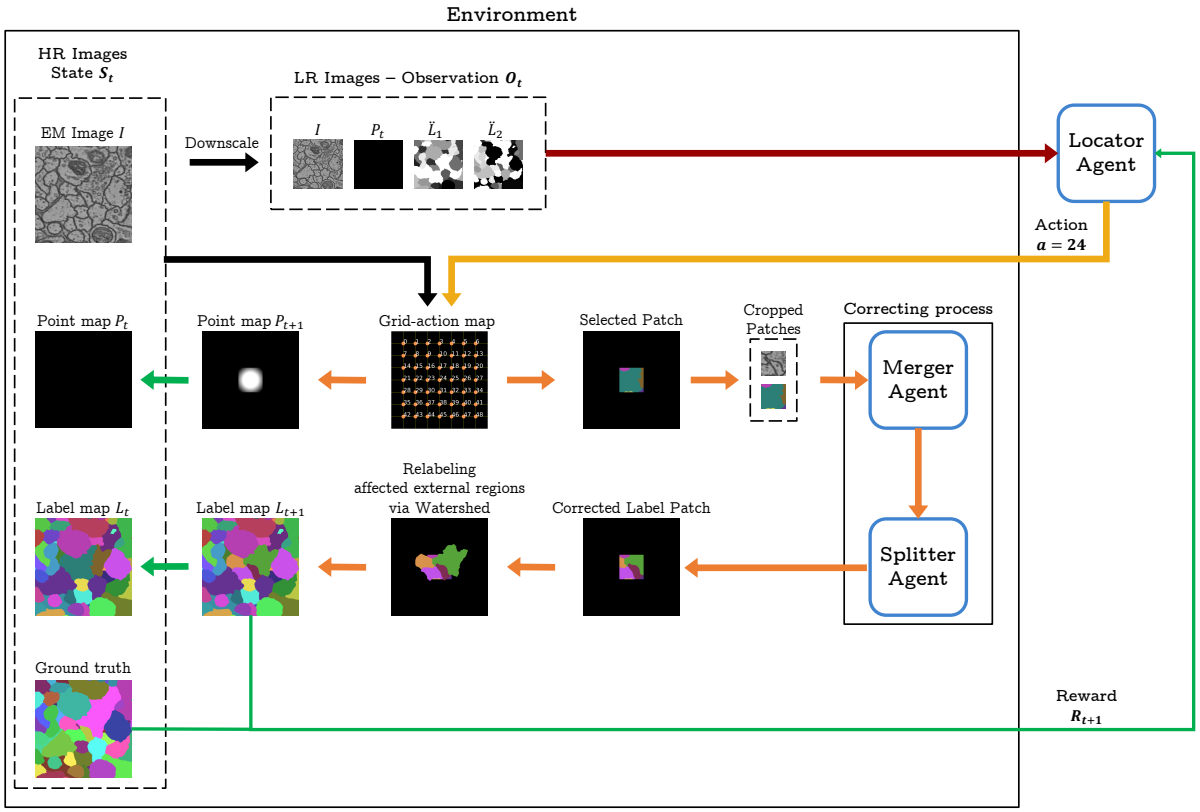


Figure 9: The interaction in Locator agent. In a time step t, The black arrow is the flow of the state S_t; the red arrow is the input observation O_t, which is the downscaled S_t, to the agent; the yellow arrow indicates the action of the agent for the corresponding O_t, hence, $a = 24$. The grid-action map gets the state S_t and read the action a; after that, the Point map P_{t+1} is inserted a Gaussian kernel on a coordinate of the action index, and also, the location of the chosen patch is calculated. The orange is the image processing flow of the environment. In this case, the image I and the label map L_t is cropped according to the location of the chosen patch. The size of the chosen patch is base Merger and Splitter agent, hence, 128×128 as I defined in the previous. A pre-trained Merge agent gets the cropped patches and corrects the split errors if they exist in the cropped label map. Next, the result from the Merger sent to a pre-trained Splitter agent for checking and correcting the merger errors. After the correcting phase, the region in label map L_{t+1} is updated base on the corrected patch. Moreover, all the modified labels inside the patch are tracked on a mask; then this mask is used for relabeling the external labels which are related to modified labels. The green arrow indicated the updating flow for the next time step and also calculating the reward Eq. 9 for the action a.

is as follows:

$$R_{t+1} = \begin{cases} 1 + R_{exp} & \text{if } L(c_t) \in E_t \\ 0 & \text{if CREMI score has been equal 0 once,} \\ & \text{or no improvement in the CREMI score} \\ -1 & \text{otherwise} \end{cases} \tag{9}$$

IV Experiments and Results

4.1 CREMI data set

CREMI is data from the MICCAI Challenge on Circuit Reconstruction from Electron Microscopy Images (CREMI) [16]. The CREMI data set consists of three volumes from an adult Drosophila melanogaster (a common fruit fly) brain tissue, each of volume has size $1250 \times 1250 \times 125$ pixels or $(5\mu m)^3$ in physical size. The three volumes are different in appearance: The neurons in CREMI A are mostly homogeneous in size and shape, the two other volumes (CREMI B and C) are more challenging, with cells that have jagged boundaries and large variations in size and shape. In my experiments, I use CREMI A data set. I use the first 92 slices in z axis for training, 23 slices for validation while training and the last 10 slices for testing. The metric for the CREMI data is the CREMI_score:

$$CREMI\_score = \sqrt{(VOI_{Merge} + VOI_{Split}) * ARand} \tag{10}$$

where VOI_{Merge}, VOI_{Split} and $ARand$ are defined in [64]. The lower the score, the better result.

4.2 Training

Synthetic Error: To train the agents, I need erroneous label maps which can be obtained from the output of a segmentation model. However, training with the result from the segmentation model can make the agents bias on the characteristics of the segmentation model. Therefore, I generate synthetic error while training the agents. Additionally, the data set is for 3D segmentation task, there are cases two cells are shown separated in an EM image but they are merged in the ground truth, which means the data contains noisy labels. But the result shows that my agents work robustly just by training with synthetic error. To generate merge error for training the Splitter agent, given a ground truth, I randomly pick from 1 to 4 segments and set them as the same index. If only 1 segment is chosen, in this case, there are no merge errors. By this scheme, the Splitter agent not only fix a big merged segment but also can fix a far segments which is may generated by Merger agent. For Locator agent, I randomly pick from 0 to 2 segments and then those chosen segments are randomly merged with theirs neighbour segments with the amount from 1 to 4 including itself. To generate split errors for Locator and Merger agents, I randomly pick a segment from the ground truth and run watershed to generate erroneous fragments. The number of fragments is from 2 to 6. Fig. 10 shows how to generating the synthetic error from the ground truth.

Agent settings: All the agents are trained with asynchronous advantage actor-critic (A3C) [65] method and the code is base on the RL framework from [66]. I train the model with Adam optimizer and the learning rate is 10^{-5}. The discount factor γ is 0.9. During training, in Merger and Splitter agents, the ϵ decreases from 1 to 0.1 over $40,000$ steps and fixed. In the Locator, the ϵ decreases from 1 to 0.1 over $10,000$ steps and fixed. The maximum length of the episode

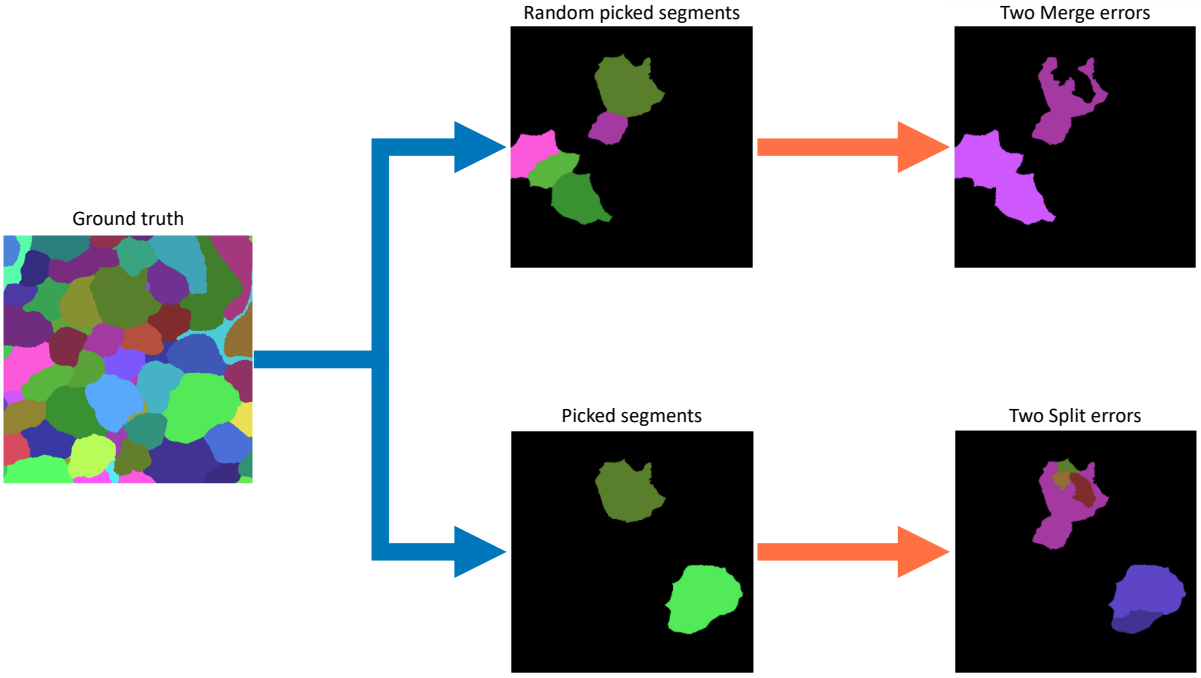


Figure 10: The flow of generating merge and split error from the ground truth. The upper branch is the generating merger error flow

is 6, if go over, I will terminate. The training procedure stop when the agent reaches 1 million steps.

4.3 Testing

Evaluate Merger and Splitter Agents

With Synthetic Error: From test set, I randomly generate synthetic error test set for special test cases:

- Split error test set: contains only split errors. The number of correct segments chosen to be split on each image is in randomly
- Merge error test set: contains only merge errors.
- Mix error test set: contains both split and merge errors, hence including the overlapped errors.

Each data set contains 1000 samples with size 128×128.

With the set of Merger agents (M1, M2, M3, M4) and two Slitter agents, I combine Merger and Splitter agent in this order:

- M1S1, M2S1 is a group of agents observing label map
- M3S2, M4S2 is a group of agents observing encoded label map

From this setup, I can measure the benefit of encoded label map. The Splitter agent are run after Merger agent because the Splitter agent is trained with the metric reward so it's stronger than the Merger agent.

In Fig. 11, M3 and M4 are the best for fixing merge error among Merger agents, especially, the CREMI score in M3 decreased 67.7%. It proofs the benefit from encoded label map in M3 and M4. All Merger agents also slightly produced merge errors (0.01 VOI_{Merge}). When there are no merge error, the Splitter agents only create a small split error (increased 3%VOI_{Split}). Merge error also produced from Splitter agent because it placed a point on a boundary. When combining Merger and Splitter agents, the errors made from Merger agent are corrected by Splitter agent after that.

- M1 reduce 19.7% when the S1 is after it
- M2 reduce 15.4% when the S1 is after it
- M3 reduce 20.6% when the S2 is after it
- Surprisingly, M4S2 is increased. Maybe because of the score from M4 is already slow from the beginning.

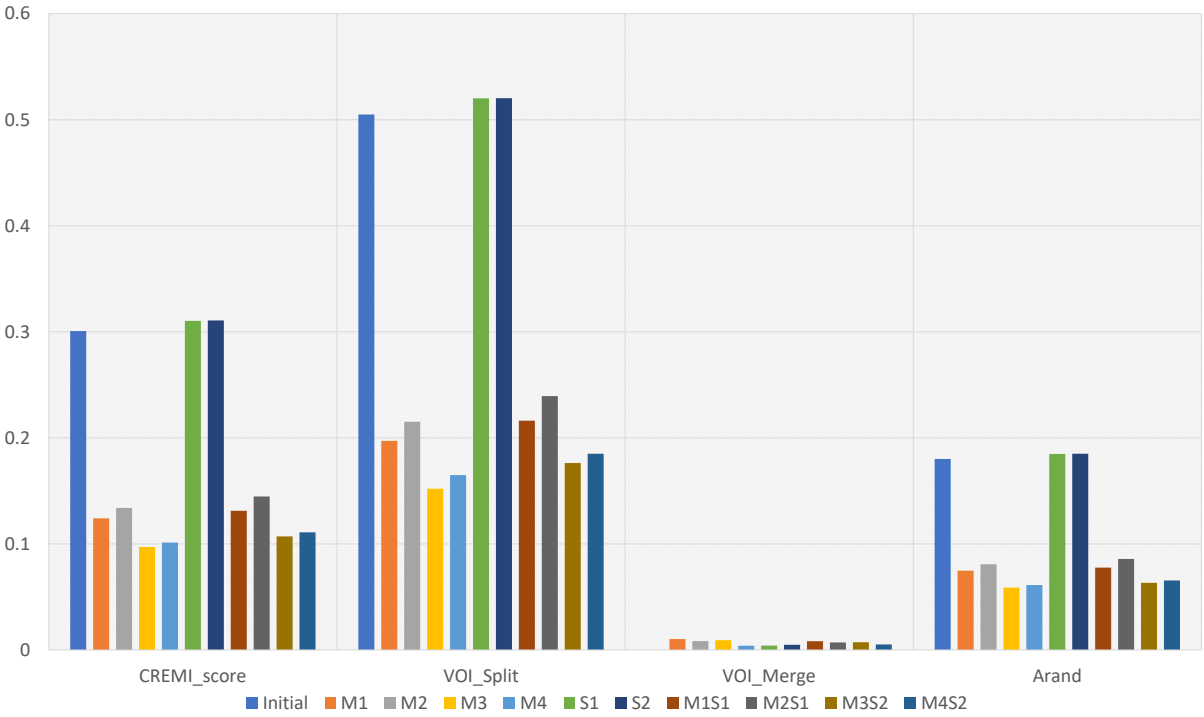


Figure 11: The result on split error test set. The lower the score, the better result.

In Fig. 12, the Splitter agents reduce 88.9% and 90.5% scores on S1 and S2. But they also created split errors. In overall, S2 is better than S1 16.6% The encoded label map shown the improvement in both Merger and Splitter agents. When there are no split error, the Merger

agents only create around 1% Merge error. From the function of Merger, merging a pair of segments, it can't create split error. In overall, the stand-alone merger agent is more stable than splitter agent.

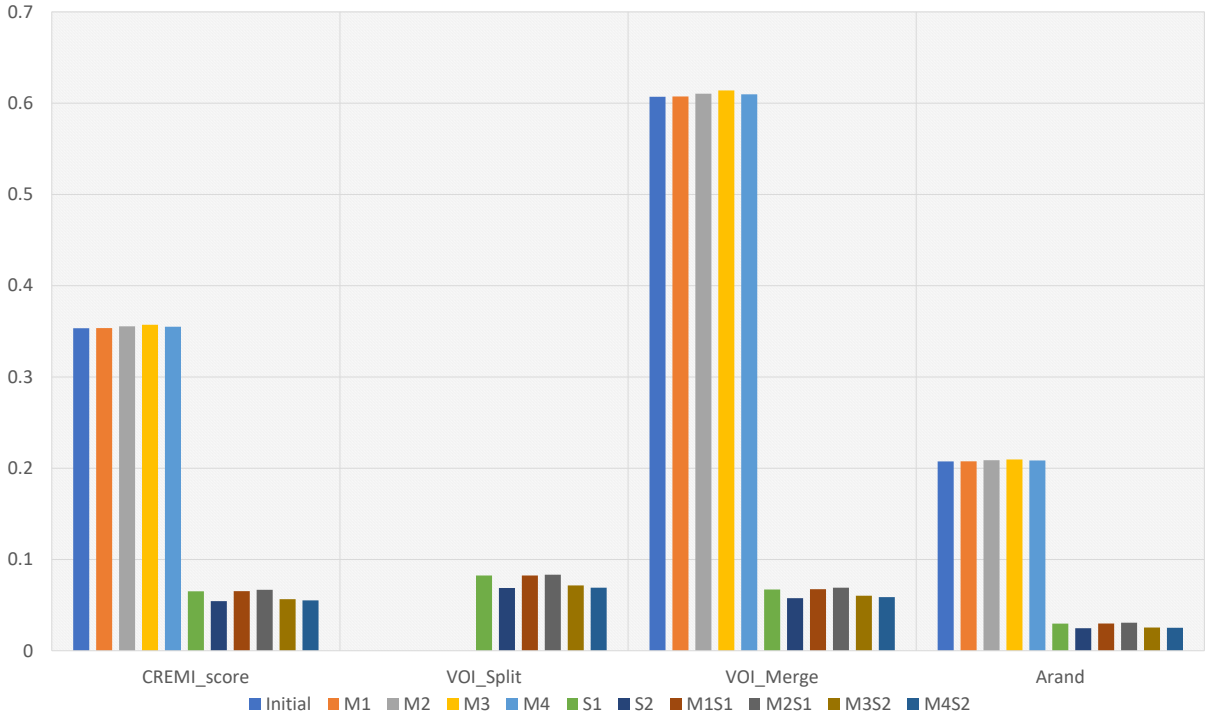


Figure 12: The result on merge error test set. The lower the score, the better result.

In Fig. 13, when the overlapped error exist, the combination of M3S2 has the best result, it reduced 75.5%$CREMI_{score}$.

In fact, the good performance on the 3 test sets can be related to the characteristic of the dataset. In this case, the CREMI A is homogeneous almost all the slices, which means the shape of the cell is not vary, so it's may easy for the agents to get the good result on the test sets. But in spite of that, the result proofs my method works.

With Segmentation Result: I use a simple U-Net [20] to get the full size 1250×1250 segmentation result on the test set; then I run the combination M3S2 to infer on the sliding window size 128×128 and step/stride size 64×64. In Fig. 14, The Patch is measured the sliding window and Image is measured the full-size image. Even the score of the Image higher 41.9% than the initial score, the score of the patch show that, the combination M3S2 did fix 16% the error on patch.

This problem happened due to the poor watershed result as shown in Fig. 15.

Evaluate Locator Agent

I trained Locator in 3 cases, with Merger-Splitter agents, random fix segments base on GT and paste all GT patch. When inference, Locator infers with Merger and Splitter (M3S2). In each

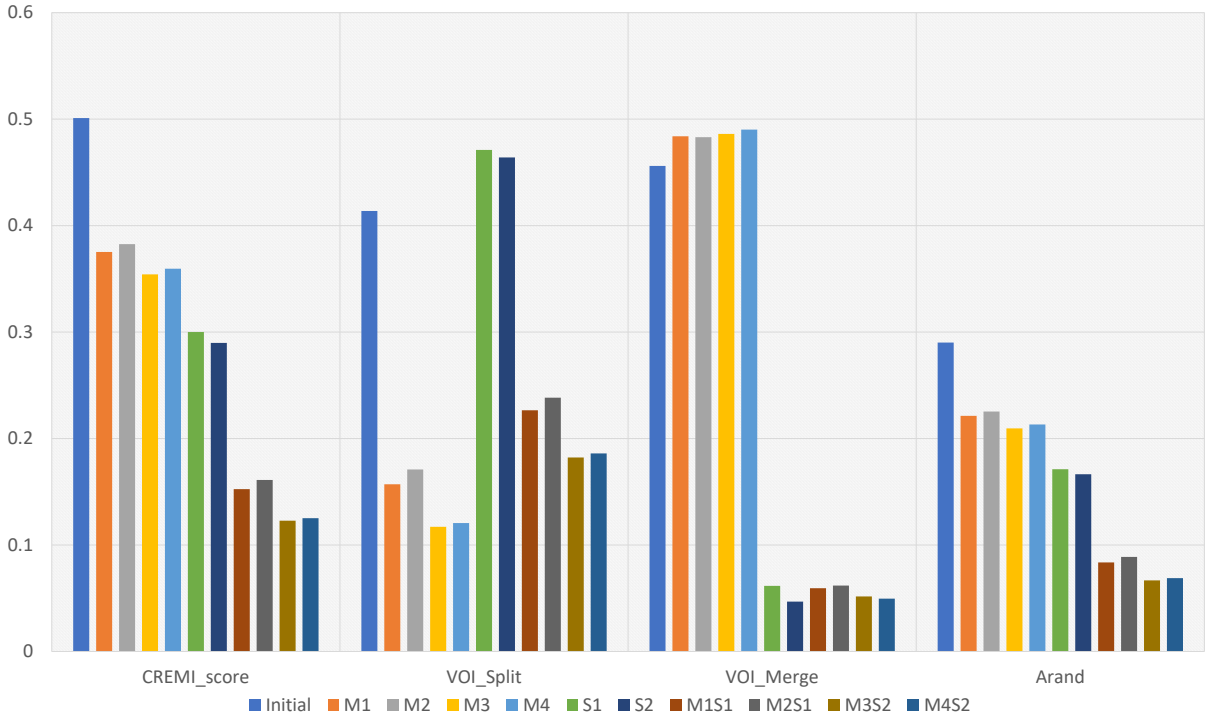


Figure 13: The result on mix error test set. The lower the score, the better result.

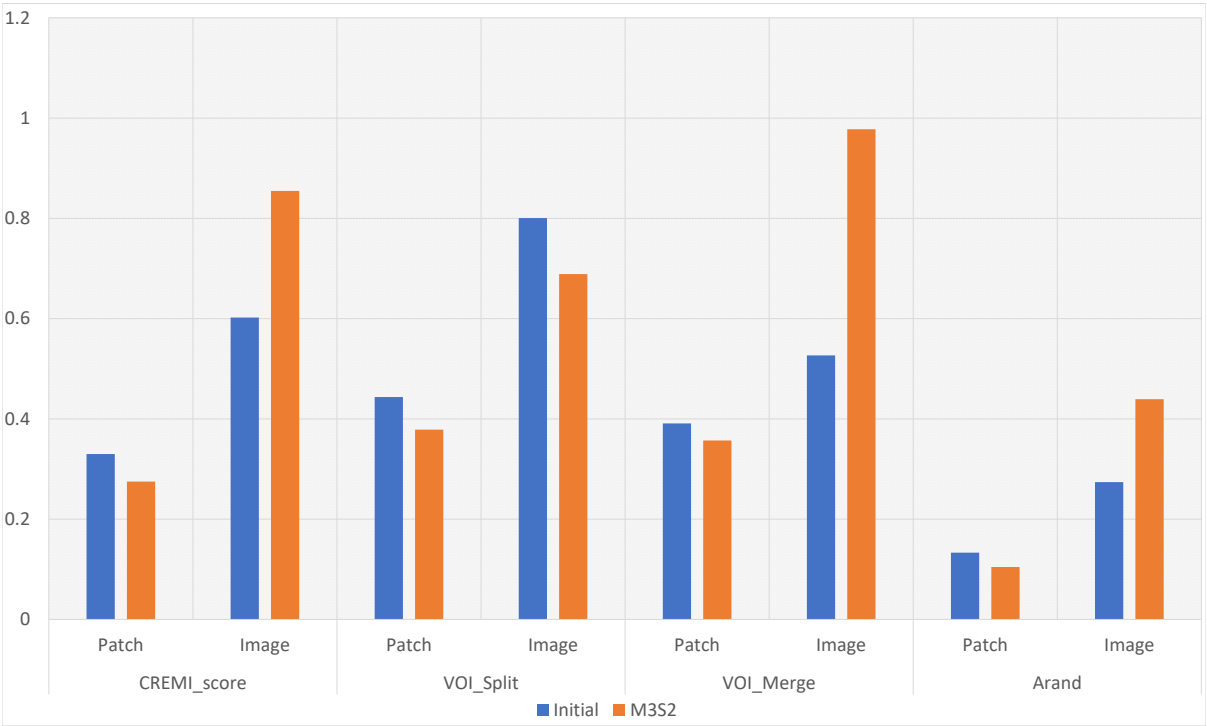


Figure 14: Inference on the full size segmentation result 1250×1250. The lower the score, the better result. The patch score is measured the sliding window output from M3S2 agent, and the Image is measured on the full size image 1250×1250. The result shows that the score of the image is increase, however, the patch score is decreased, which means M3S2 corrected the input patch correctly.

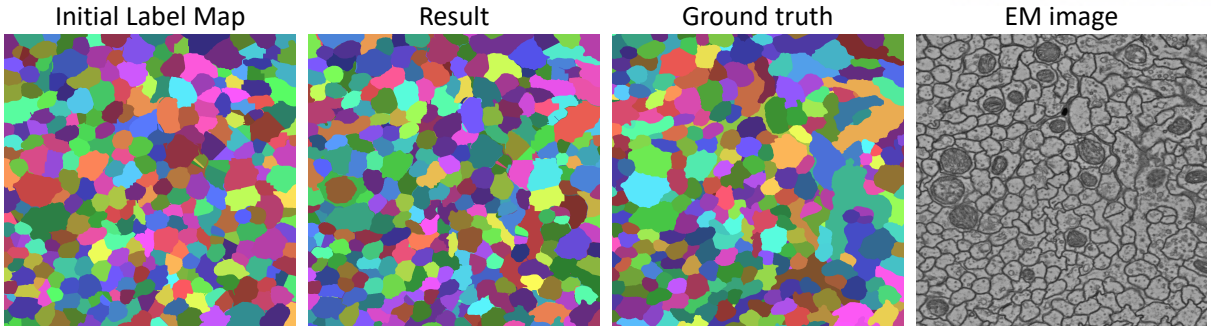


Figure 15: A result on a full-size image in test set. The size of the slit error so small, like the tail of a segment. Therefore, even the M3S2 can fix the error on the patch, the tail is still counted as an error on the image.

cases, I test the size of O, or the input size, with size 128×128 and 256×256.

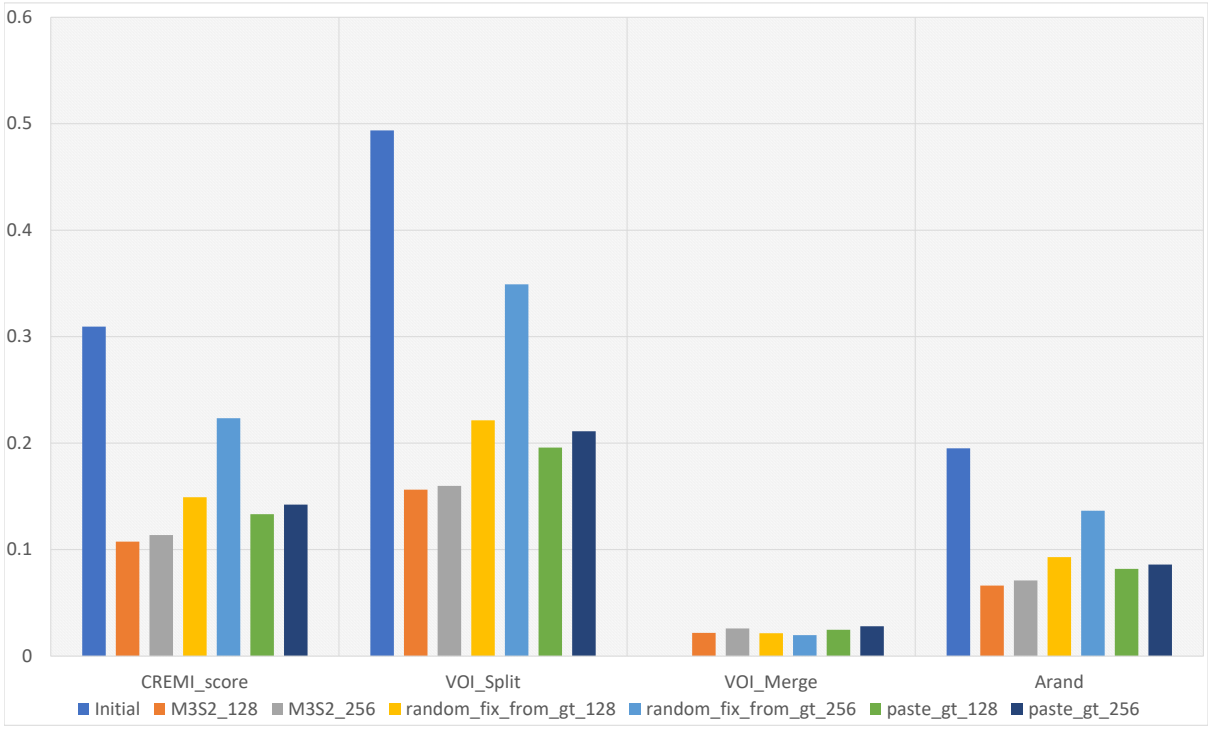


Figure 16: The result of Locator on split error test set. The lower the score, the better result.

All the results Fig. 16,Fig. 17 and Fig. 18 show that, train Locator with Merger and Splitter agents is the best option and it fixed around 70% on $CREMI\_score$ in all synthetic error test sets.

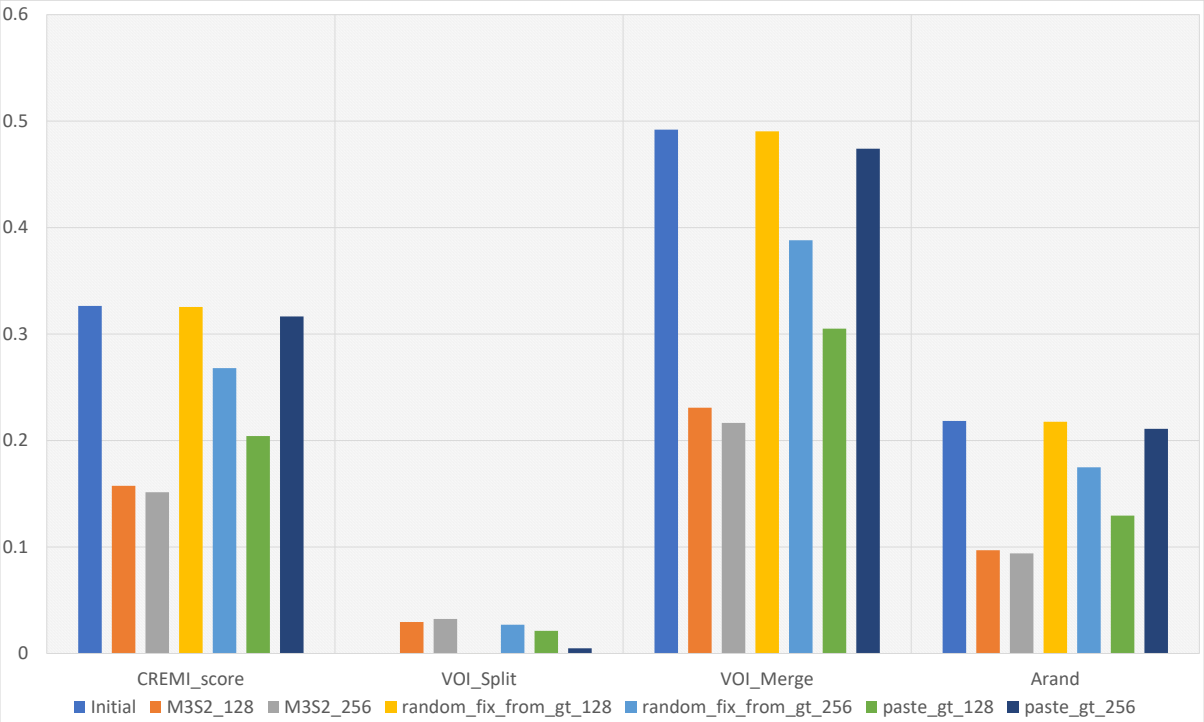


Figure 17: The result of Locator on merge error test set. The lower the score, the better result.

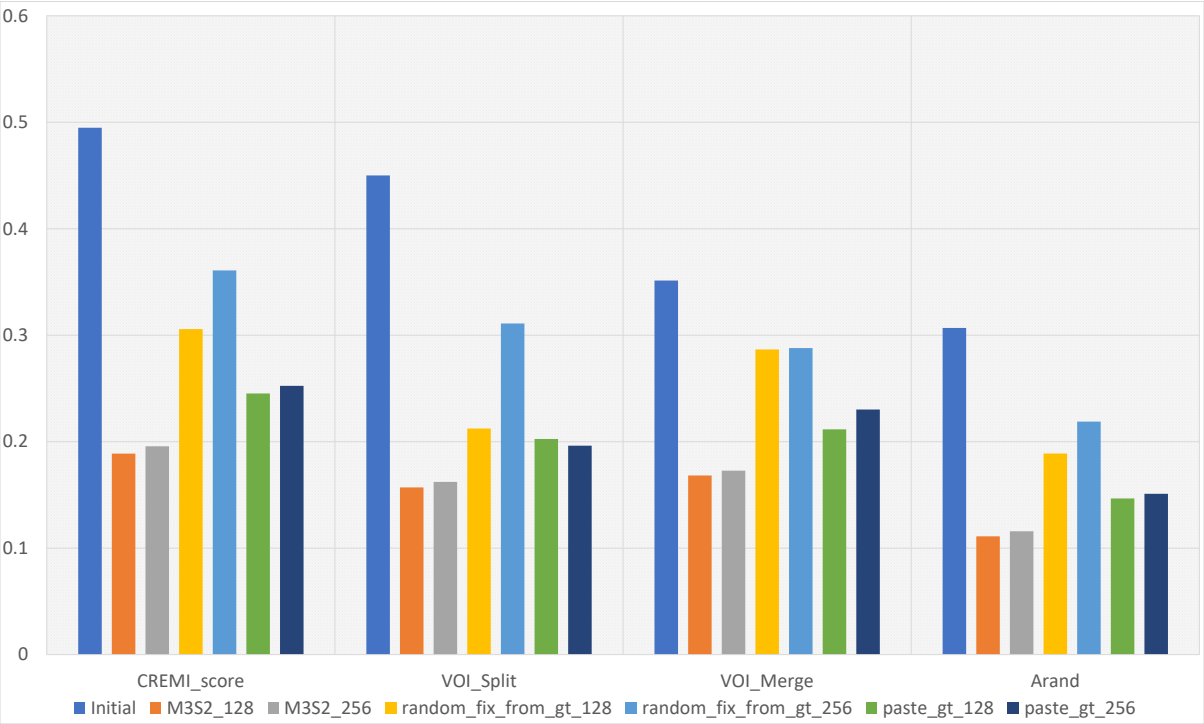


Figure 18: The result of Locator on mix error test set. The lower the score, the better result.

V Conclusion and Future Work

Nowadays, many segmentation works is proposed gradually increasing the accuracy, which made the proofreading topic becomes less attention. From the proposed system, I created a new method to resolve the bottleneck problem in proofreading task by replacing manually proofreading with automatic mimicking human proofreading process. The result in section IV shown that, this method is promising. For the future works, I will expand the agent from discrete action space to continuous action space. And also, there are many options can replace the Locator agent such as using a CNNs to classify a sliding window of HR image containing a error or not. A comparison between them are needed.

References

[1] N. W. Encyclopedia, "Nervous system — new world encyclopedia,," 2013, [Online; accessed 30-November-2020]. [Online]. Available: https://www.newworldencyclopedia.org/p/index.php?title=Nervous_system&oldid=975999

[2] K. Saladin, *Human Anatomy*. McGraw-Hill, 2013. [Online]. Available: https://books.google.co.kr/books?id=KUlBMAEACAAJ

[3] C. S. von Bartheld, J. Bahney, and S. Herculano-Houzel, "The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting," *Journal of Comparative Neurology*, vol. 524, no. 18, pp. 3865–3895, 2016.

[4] E. R. Kandel, J. H. Schwartz, T. M. Jessell, D. of Biochemistry, M. B. T. Jessell, S. Siegelbaum, and A. Hudspeth, *Principles of neural science*. McGraw-hill New York, 2000, vol. 4.

[5] Google. Google connectomics. [Online]. Available: https://research.google/teams/perception/connectomics

[6] O. Sporns, G. Tononi, and R. Kötter, "The human connectome: a structural description of the human brain," *PLoS Comput Biol*, vol. 1, no. 4, p. e42, 2005.

[7] P. Hagmann, "From diffusion mri to brain connectomics," EPFL, Tech. Rep., 2005.

[8] O. Sporns, "Connectome," *Scholarpedia*, vol. 5, no. 2, p. 5584, 2010, revision #141341.

[9] J. W. Lichtman and W. Denk, "The big and the small: challenges of imaging the brain's circuits," *Science*, vol. 334, no. 6056, pp. 618–623, 2011.

[10] S. Seung, *Connectome: How the brain's wiring makes us who we are*. HMH, 2012.

[11] M. Helmstaedter, "The mutual inspirations of machine learning and neuroscience," *Neuron*, vol. 86, no. 1, pp. 25–28, 2015.

[12] D. Haehn, J. Hoffer, B. Matejek, A. Suissa-Peleg, A. K. Al-Awami, L. Kamentsky, F. Gonda, E. Meng, W. Zhang, R. Schalek *et al.*, "Scalable interactive visualization for connectomics," in *Informatics*, vol. 4, no. 3. Multidisciplinary Digital Publishing Institute, 2017, p. 29.

[13] A. Suissa-Peleg, D. Haehn, S. Knowles-Barley, V. Kaynig, T. R. Jones, A. Wilson, R. Schalek, J. W. Lichtman, and H. Pfister, "Automatic neural reconstruction from

petavoxel of electron microscopy data," *Microscopy and Microanalysis*, vol. 22, no. S3, pp. 536–537, 2016.

[14] N. Kasthuri, K. J. Hayworth, D. R. Berger, R. L. Schalek, J. A. Conchello, S. Knowles-Barley, D. Lee, A. Vázquez-Reina, V. Kaynig, T. R. Jones *et al.*, "Saturated reconstruction of a volume of neocortex," *Cell*, vol. 162, no. 3, pp. 648–661, 2015.

[15] D. G. C. Hildebrand, M. Cicconet, R. M. Torres, W. Choi, T. M. Quan, J. Moon, A. W. Wetzel, A. S. Champion, B. J. Graham, O. Randlett *et al.*, "Whole-brain serial-section electron microscopy in larval zebrafish," *Nature*, vol. 545, no. 7654, p. 345, 2017.

[16] CREMI. (2016) Miccai challenge on circuit reconstruction from electron microscopy images. [Online]. Available: https://cremi.org

[17] V. Kaynig, A. Vazquez-Reina, S. Knowles-Barley, M. Roberts, T. R. Jones, N. Kasthuri, E. Miller, J. Lichtman, and H. Pfister, "Large-scale automatic reconstruction of neuronal processes from electron microscopy images," *Medical image analysis*, vol. 22, no. 1, pp. 77–88, 2015.

[18] R. Schalek, D. Lee, N. Kasthuri, A. Peleg, T. Jones, V. Kaynig, D. Haehn, H. Pfister, D. Cox, and J. W. Lichtman, "Imaging a 1 mm 3 volume of rat cortex using a multibeam sem," *Microscopy and Microanalysis*, vol. 22, no. S3, pp. 582–583, 2016.

[19] J. W. Lichtman, H. Pfister, and N. Shavit, "The big data challenges of connectomics," *Nature neuroscience*, vol. 17, no. 11, pp. 1448–1454, 2014.

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[21] T. M. Quan, D. G. Hildebrand, and W.-K. Jeong, "Fusionnet: A deep fully residual convolutional neural network for image segmentation in connectomics," *arXiv preprint arXiv:1612.05360*, 2016.

[22] J. Nunez-Iglesias, R. Kennedy, T. Parag, J. Shi, and D. B. Chklovskii, "Machine learning of hierarchical clustering to segment 2d and 3d images," *PloS one*, vol. 8, no. 8, p. e71715, 2013.

[23] J. Nunez-Iglesias, R. Kennedy, S. M. Plaza, A. Chakraborty, and W. T. Katz, "Graph-based active learning of agglomeration (gala): a python library to segment 2d and 3d neuroimages," *Frontiers in neuroinformatics*, vol. 8, p. 34, 2014.

[24] T. Parag, F. Tschopp, W. Grisaitis, S. C. Turaga, X. Zhang, B. Matejek, L. Kamentsky, J. W. Lichtman, and H. Pfister, "Anisotropic em segmentation by 3d affinity learning and agglomeration," *arXiv preprint arXiv:1707.08935*, 2017.

[25] K. Lee, J. Zung, P. Li, V. Jain, and H. S. Seung, "Superhuman accuracy on the snemi3d connectomics challenge," *arXiv preprint arXiv:1706.00120*, 2017.

[26] J. Funke, F. Tschopp, W. Grisaitis, A. Sheridan, C. Singh, S. Saalfeld, and S. C. Turaga, "Large scale image segmentation with structured loss based deep learning for connectome reconstruction," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 7, pp. 1669–1680, 2018.

[27] M. Januszewski, J. Kornfeld, P. H. Li, A. Pope, T. Blakely, L. Lindsey, J. Maitin-Shepard, M. Tyka, W. Denk, and V. Jain, "High-precision automated reconstruction of neurons with flood-filling networks," *Nature methods*, vol. 15, no. 8, pp. 605–610, 2018.

[28] P. Li, L. Lindsey, M. Januszewski, Z. Zheng, A. S. Bates, I. Taisz, M. Tyka, M. Nichols, F. Li, E. Perlman, J. Maitin-Shepard, T. Blakely, L. J. Leavitt, G. S. Jefferis, D. Bock, and V. Jain, "Automated reconstruction of a serial-section em drosophila brain with flood-filling networks and local realignment," *bioRxiv*, 2019. [Online]. Available: https://www.biorxiv.org/content/10.1101/605634v2

[29] R. Sicat, M. Hadwiger, and N. J. Mitra, "Graph abstraction for simplified proofreading of slice-based volume segmentation." in *Eurographics (Short Papers)*, 2013, pp. 77–80.

[30] S. Knowles-Barley, M. Roberts, N. Kasthuri, D. Lee, H. Pfister, and J. W. Lichtman, "Mojo 2.0: Connectome annotation tool," *Frontiers in Neuroinformatics*, vol. 60, p. 1, 2013.

[31] D. Haehn, S. Knowles-Barley, M. Roberts, J. Beyer, N. Kasthuri, J. W. Lichtman, and H. Pfister, "Design and evaluation of interactive proofreading tools for connectomics," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2466–2475, 2014.

[32] H. Peng, F. Long, T. Zhao, and E. Myers, "Proof-editing is the bottleneck of 3d neuron reconstruction: the problem and solutions," *Neuroinformatics*, vol. 9, no. 2, pp. 103–105, 2011.

[33] S. M. Plaza, "Focused proofreading to reconstruct neural connectomes from em images at scale," in *Deep Learning and Data Labeling for Medical Applications*. Springer, 2016, pp. 249–258.

[34] D. Haehn, V. Kaynig, J. Tompkin, J. W. Lichtman, and H. Pfister, "Guided proofreading of automatic segmentations for connectomics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9319–9328.

[35] J. Zung, I. Tartavull, K. Lee, and H. S. Seung, "An error detection and correction framework for connectomics," in *Advances in Neural Information Processing Systems*, 2017, pp. 6818–6829.

[36] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction Second Edition*. MIT Press, Cambridge, MA, 2018, 2018.

[37] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[38] J. S. Kim, M. J. Greene, A. Zlateski, K. Lee, M. Richardson, S. C. Turaga, M. Purcaro, M. Balkam, A. Robinson, B. F. Behabadi *et al.*, "Space–time wiring specificity supports direction selectivity in the retina," *Nature*, vol. 509, no. 7500, pp. 331–336, 2014.

[39] C.-S. Lee, M.-H. Wang, S.-J. Yen, T.-H. Wei, I.-C. Wu, P.-C. Chou, C.-H. Chou, M.-W. Wang, and T.-H. Yan, "Human vs. computer go: Review and prospect [discussion forum]," *IEEE Computational Intelligence Magazine*, vol. 11, no. 3, pp. 67–72, 2016.

[40] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[41] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[42] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019.

[43] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," *arXiv preprint arXiv:2003.13350*, 2020.

[44] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. **v**Zídek, A. W. Nelson, A. Bridgland *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[45] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 163–168.

[46] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[47] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[48] V. Jain, B. Bollmann, M. Richardson, D. R. Berger, M. N. Helmstaedter, K. L. Briggman, W. Denk, J. B. Bowden, J. M. Mendenhall, W. C. Abraham *et al.*, "Boundary learning by optimization with topological constraints," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 2488–2495.

[49] T. Liu, C. Jones, M. Seyedhosseini, and T. Tasdizen, "A modular hierarchical approach to 3d electron microscopy image segmentation," *Journal of neuroscience methods*, vol. 226, pp. 88–102, 2014.

[50] V. Jain, J. F. Murray, F. Roth, S. Turaga, V. Zhigulin, K. L. Briggman, M. N. Helmstaedter, W. Denk, and H. S. Seung, "Supervised learning of image restoration with convolutional networks," in *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.

[51] S. C. Turaga, J. F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, and H. S. Seung, "Convolutional networks can learn to generate affinity graphs for image segmentation," *Neural computation*, vol. 22, no. 2, pp. 511–538, 2010.

[52] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer, 2018, pp. 3–11.

[53] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz, B. Glocker, and D. Rueckert, "Attention u-net: Learning where to look for the pancreas," 2018.

[54] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. R. Zaiane, and M. Jagersand, "U2-net: Going deeper with nested u-structure for salient object detection," *Pattern Recognition*, vol. 106, p. 107404, 2020.

[55] SNEMI3D. (2013) Isbi 2013 challenge: 3d segmentation of neurites in em images. [Online]. Available: http://brainiac2.mit.edu/SNEMI3D/

[56] D. B. Chklovskii, S. Vitaladevuni, and L. K. Scheffer, "Semi-automated reconstruction of neural circuits using electron microscopy," *Current opinion in neurobiology*, vol. 20, no. 5, pp. 667–675, 2010.

[57] J. F. F. Project. Raveler. [Online]. Available: https://janelia-flyem.github.io/

[58] M. G. Uzunbas, C. Chen, and D. Metaxas, "An efficient conditional random field approach for automatic and interactive neuron segmentation," *Medical image analysis*, vol. 27, pp. 31–44, 2016.

[59] G. Song, H. Myeong, and K. Mu Lee, "Seednet: Automatic seed generation with deep reinforcement learning for robust interactive segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1760–1768.

[60] N. Dong, M. Kampffmeyer, X. Liang, Z. Wang, W. Dai, and E. Xing, "Reinforced auto-zoom net: Towards accurate and fast breast cancer segmentation in whole-slide images," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer, 2018, pp. 317–325.

[61] M. Gao, R. Yu, A. Li, V. I. Morariu, and L. S. Davis, "Dynamic zoom-in network for fast object detection in large images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6926–6935.

[62] B. Uzkent and S. Ermon, "Learning When and Where to Zoom With Deep Reinforcement Learning," 2020, pp. 12 345–12 354. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2020/html/Uzkent_Learning_When_and_Where_to_Zoom_With_Deep_Reinforcement_Learning_CVPR_2020_paper.html

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, pp. 5998–6008, 2017.

[64] I. Arganda-Carreras, S. C. Turaga, D. R. Berger, D. Cireşan, A. Giusti, L. M. Gambardella, J. Schmidhuber, D. Laptev, S. Dwivedi, J. M. Buhmann *et al.*, "Crowdsourcing the creation of image segmentation algorithms for connectomics," *Frontiers in neuroanatomy*, vol. 9, p. 142, 2015.

[65] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[66] D. Griffis, "Gpu/cpu architecture of a3c," 2019, https://github.com/dgriff777/rl_a3c_pytorch.

Acknowledgements

I'm extremely grateful to my advisor Prof. Won-Ki Jeong for guiding me from the beginning of my Master's program, not only technical guidance but also research skills. He gave me an wonderful opportunity to begin my journey in research.

In addition, I would like to thank the rest of my thesis defense committee members: Prof. Se Young Chun and Prof. Jae-Young Sim, for their meaningful comments and also all professors taught me in UNIST.

Many thanks to my labmates for supporting me and their friendship. Especially Tuan, It was great working with you.

In the end, I would like to thank my family for supporting me. I wouldn't be able to do anything without them.