3-Dimensional CT Organ Localization with Deep Reinforcement Learning

Final Project Report

Kexin Yang
Department of Biostatistics
Harvard University
Cambridge, MA 02138
kexinyang@hsph.harvard.edu

Haoming Chen
Department of Biostatistics
Harvard University
Cambridge, MA 02138
haomingchen@hsph.harvard.edu

Hainan Xiong
Department of Biostatistics
Harvard University
Cambridge, MA 02138
hainanxiong@hsph.harvard.edu

Abstract

Conventional manual localization of different organs in 3D imaging scans is time-consuming, therefore reducing the time of this task can also reduce the time for the disease diagnosis process. In this paper, we designed a reinforcement learning algorithm to automatically localize specified organ(s) in CT images to help radiologists evaluate the CT scans more easily and reduce the time for disease diagnosis. Our model was based on the reinforcement learning-based organ localization model by Navarro et al. [4], referred to as the "baseline model". They trained their RL model to identify a "bounding box", which is the smallest 3-dimensional box containing the specified organ. In this paper, we proposed three novel approaches to improve the baseline model: *(a)* an optimized reward function; *(b)* a new termination condition; *(c)* a novel action update method. These three approaches together were designed to both reduce computation time and increase accuracy. Our model was trained and tested on a publicly available multi-organ segmentation dataset for liver localization. We evaluated the model performance using the 3-dimensional Intersection over Union (IoU) score between the model predictions bounding box and the true label. For future direction, if our model can generate a better performance on the liver localization task, we plan to train the model on more organ localization tasks using larger datasets to test for its generalizability. If the model can achieve high organ localization across different organs on large datasets, it may be possible to deploy this model in hospitals as a part of the CT scan pre-processing pipeline to help the radiologists localize the organ(s) of interest in advance and improve their working efficiency.

1 Introduction

3D organ localization is the process of identifying the location of organs within a 3-dimensional image such as a CT scan. This is an important task in medical imaging, as it allows radiologists and doctors to better understand the position and condition of organs and make disease diagnoses, and plan surgeries or other medical procedures. It is an essential preprocessing step for many medical image analysis tasks. For example, it could improve lesion detection as it can reduce the search space. Traditionally, organ localization has been done manually, which can be time-consuming and prone to error. With the emergence of deep learning techniques in recent years, scientists have proposed numerous novel methods for 3D organ localization and other image tasks. Image segmentation algorithms based on deep neural networks have improved the accuracy of localizing organs of interest and at the same time reduce the time of searching and labeling manually. However, there are some challenges associated with deep learning-based methods in medical applications. First, most state-of-the-art (SOTA) deep Convolutional Neural Network (CNN) models designed for medical imaging, such as U-Net, are computationally expensive since these models contain a large number of parameters that need to be trained. Second, most of the deep CNN models require a large corpus of annotated training data in order to achieve a good performance, making them prohibitive for medical tasks since it is hard to obtain a sufficient amount of annotated data, especially for 3D CT scans.

Recently, some deep Reinforcement Learning (RL) models have been proposed as a new solution for 3D organ localization. These models use a combination of deep learning and reinforcement learning to automatically learn the

location of organs within a 3D space. This is done by using a combination of 3D imaging data and reward signals to train the model. The model can learn to localize the organs in a 3D CT scan environment with a high degree of accuracy. Deep RL models can resolve the two problems associated with the deep CNN models mentioned before. First, it is able to achieve good performance with a much smaller model size due to the use of the reward function. Second, it only requires a very small amount of training dataset since RL models learn the task by repetitively interacting with the environment and storing these "game trajectories" in the replay buffer.

In this work, inspired by the work of Navarro et al. [4], we implemented a reinforcement learning algorithm to automatically localize specified organs in CT images. We also proposed several improvements to further optimize the model architecture and improve the model performance.

2 Related Works

2.1 Deep Reinforcement Learning

Reinforcement Learning: Reinforcement learning [5] is a type of machine learning inspired by the human decision-making process. It involves an artificial agent that interacts with an environment E by taking actions a based on observations s of the current state. The environment then responds to the agent's actions with a reward signal R, which the agent uses to evaluate the effectiveness of its actions. The goal of the agent is to learn a behavior function that maximizes the total reward it receives until reaching the terminate state.

Q learning: Reinforcement learning is essentially a mapping function between states and actions, which can maximize future rewards. Q learning [7] is one specific method in reinforcement learning which selects the optimal action policy by measuring the quality of taking a specific action a in a given state s, represented by $Q(s, a)$. This function, also known as the Q-function, is the expected value of the accumulated future rewards, discounted by a factor of $\gamma \in [0,1]$ to account for the uncertainty in the agent's environment. This value function can be calculated recursively using the Bellman Equation, allowing for an iterative solution to the problem.

Deep Reinforcement Learning: Deep reinforcement learning [3] is a type of reinforcement learning that uses deep neural networks as the function approximator to represent the value or policy function. This allows deep reinforcement learning algorithms to learn directly from high-dimensional sensory inputs, such as images or video, without the need for hand-engineered features. Deep reinforcement learning has been applied to a variety of challenging problems, such as playing video games, robot control, and autonomous driving. One of the key challenges in deep reinforcement learning is balancing exploration and exploitation, as the agent must learn to explore its environment in order to discover new rewards, but also must learn to exploit its knowledge in order to maximize its reward. This trade-off has been addressed in a variety of ways, including the use of exploration strategies, such as epsilon-greedy exploration, and the use of intrinsic motivation, where the agent is motivated to explore based on its own curiosity or learning progress.

Our model follows the architecture in the baseline paper [4] and uses the Deep Double Q-Network (DDQN) [2, 6], which combines the Q learning and Deep Reinforcement Learning methods, and uses a deep neural network to represent the feature map and approximate the q-values for the actions.

2.2 Object Localization

Object localization is a technique in computer vision that involves identifying the location and extent of objects in an image or video. This is typically done by training a machine learning model to predict the bounding boxes around objects in an image, along with a corresponding label for each object. The bounding box is defined by its coordinates in the image, and the label is a class label that specifies the type of object. Object localization can be used for a variety of applications, such as object detection in surveillance systems, or as part of a larger object recognition system that aims to identify and classify objects in images.

Object Localization with Deep Reinforcement Learning: Navarro et al. [4] proposed a deep reinforcement learning-based approach for object localization tasks for the first time. Previous attempts for organ localization include multi-atlas registration, machine learning-based approaches, and 2D / 3D Convolutional Neural Network-based approaches. Although some could achieve high accuracy, all of the previous approaches are either computationally heavy or require a lot of labeled training data. To mitigate those issues, the proposed approach introduces 11 actions tailored for organ localization tasks and trains with only a limited amount of labeled data. It outperforms other baseline methods on the VISCERAL dataset, making the future of the RL-based approach for organ localization works promising.

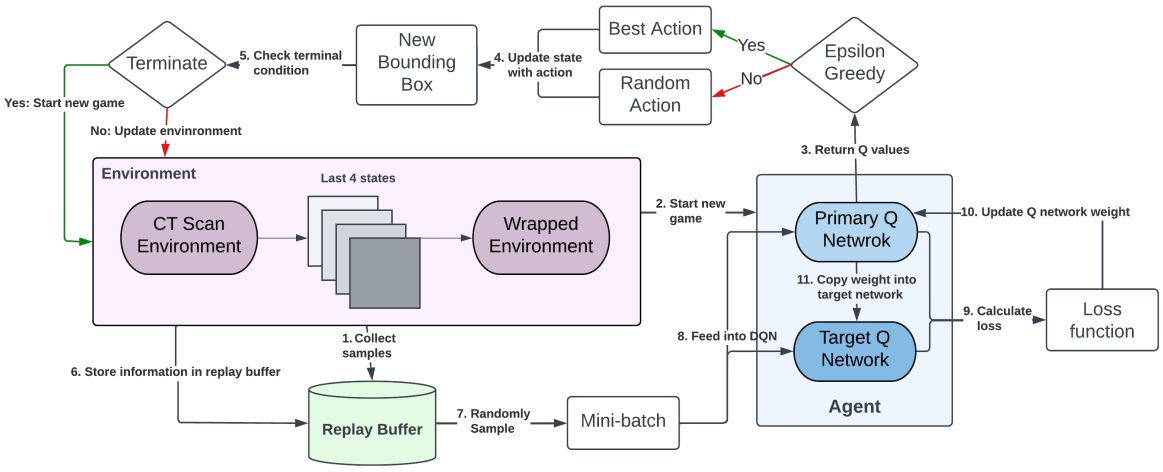


Figure 1: Model framework for our proposed deep reinforcement learning model

3 Methods

Our model mainly consists of three parts: an environment that contains the CT scan contents, an agent which will make the decision on how to move the bounding box, and a replay buffer that contains the past game trajectory information such as the action, bounding box coordinates and rewards for each step.

To train the model, we first collected samples to fill the replay buffer. Then, during each training iteration, a CT scan was sampled from the training dataset and used as the environment for this game episode. The initial bounding box was generated at the center of the scan. The scan contents inside the bounding box were used to represent the current states. The current plus the previous 3 states were fed into the agent's primary Q network to calculate the q values for these states. We implemented the agent to follow an ϵ-greedy policy when choosing the action, where the value of ϵ sets the probabilistic threshold of choosing a random action. A random number was generated each time to determine the action strategy. If the random number is smaller than ϵ, the action will be chosen randomly, otherwise, the action with the highest q-value was chosen. The purpose of using this ϵ-greedy policy was to allow some degree of exploration so that the agents can try random actions and see the results. The new bounding box coordinate was calculated after performing the action, and this loop was repeated until the termination condition was reached. This new game trajectory was also stored in the replay buffer, and after playing a certain number of games, we sampled a mini-batch of trajectory from the replay buffer to calculate the loss and update the weight of the primary Q network.

3.1 Environment

The entire 3-dimensional CT scan is used to represent the environment of the current game episode. It records the current and true bounding box coordinates at each time step. It also contains a *step()* function that takes the action chosen by the agent as an argument and calculates the new bounding box coordinate after performing the action. The voxel values inside the new bounding box represent the current states of the game, and it is warped into a fixed screen-size matrix using affine transformation to ensure that the size of the state inputs to the agent's Q Network is fixed across different game episodes. The updated IoU score and rewards are calculated using the new bounding box, and the terminal condition is also checked each time to determine whether the current game episode should continue or not.

Actions: We provide 11 possible actions which allow the bounding box to be moved and transformed to cover all possible regions in the CT scan environment. This is the same as the implementation in the baseline model. There are 6 translation actions, which move the bounding box in one of the six possible directions (left, right, front, back, up, down). The 2 scaling actions change the size of the bounding box and allow it to be zoomed in or out. The 3 deformation actions transform the aspect ratio of the bounding box in one of the three possible axes and make the bounding box to be thinner, flatter, or taller. Figure 2 (b) illustrate how the bounding box is transformed by each action.

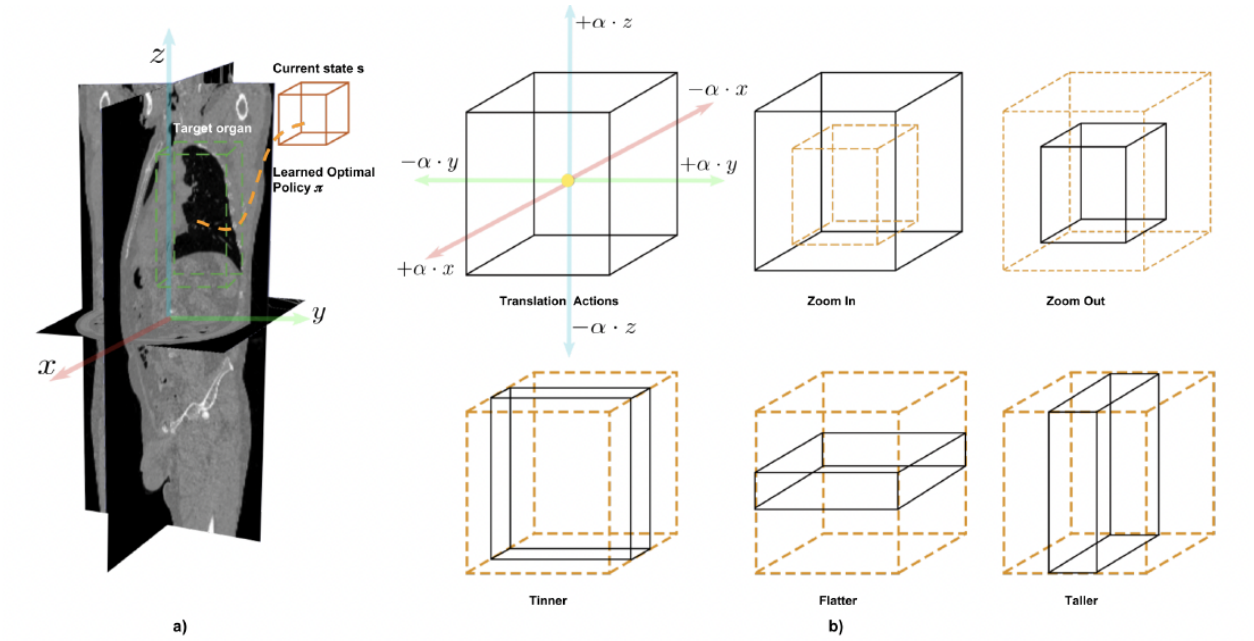


Figure 2: a) The entire 3D CT scan is used as the environment for the current game episode. The orange box indicates the current bounding box and the green box shows the true bounding box label. b) Illustration of 11 possible actions. The 6 translation action moves the bounding box into one of the six possible directions; the 2 scaling actions zoom the box in or out, and the 3 deformation actions transform the box to be thinner, flatter or taller.

Action Update Method: We proposed an adaptive step size scheme to reduce the number of steps taken to reach the target location. The action step size is proportional to the current bounding box length, as implemented in the baseline. In addition, we defined minimum and maximum step sizes as hyperparameters to prevent the liver bounding box from getting too large or too small. For out-of-boundary bounding boxes, we cropped the bounding box size to only include valid regions and assign a penalty to the reward function. The penalty score is also a hyperparameter.

Terminal Condition: The terminal condition is different between training and testing since the ground truth label is not available in testing. The baseline model terminates the game if the predicted bounding box reaches a certain IoU score threshold. During training, the agent stops searching when *(a)* the intersection-over-union (IoU) between the current state and the target coordinates is greater than or equal to a predefined threshold value; *(b)* maximum step number is reached. In our model, in order to allow the model to gradually adapt to more challenging training data, we implemented the terminate IoU score threshold and max step size to be dynamic and increase with more training epochs. The range of terminate IoU score threshold and max step size are also hyperparameters. During testing, the localization process is terminated when the max step number is reached.

Rewards: The reward function tells the agent how much the current action a^t taken in state s^t improved the IoU compared to the previous IoU. The baseline used a binary reward function to indicate whether a specific action improve or decrease the IoU score. However, this reward function can only differentiate between bad and good actions but is insensitive to the magnitude of improvement. Therefore, we propose a new reward function that consists of three components and allows the agent to be more responsive to the quality of actions, and it is shown below.

$$
\begin{aligned}
& R\left(s^t, a^t, s^{t-1}, a^{t-1}\right) \\
& = \alpha_{\text{IoU}} \cdot \left(\text{IoU}(\text{box}^t_{\text{pred}}, \text{box}^t_{\text{target}}) - \text{IoU}(\text{box}^{t-1}_{\text{pred}}, \text{box}^{t-1}_{\text{target}})\right) \\
& + \alpha_{\text{terminate}} \cdot \mathbb{1}(\text{exceed terminate IoU}) + \alpha_{\text{out}} \cdot \mathbb{1}(\text{out of boundry})
\end{aligned}
$$

The reward function consists of three components:

- The difference between current and previous IoU times a scaling factor α_{iou} to reflect the degree of improvement.
- Add additional rewards times a scaling factor $\alpha_{terminate}$ if the terminal IoU is reached within max step size
- Give penalty times a scaling factor α_{out} if the new bounding box exceeds the boundary

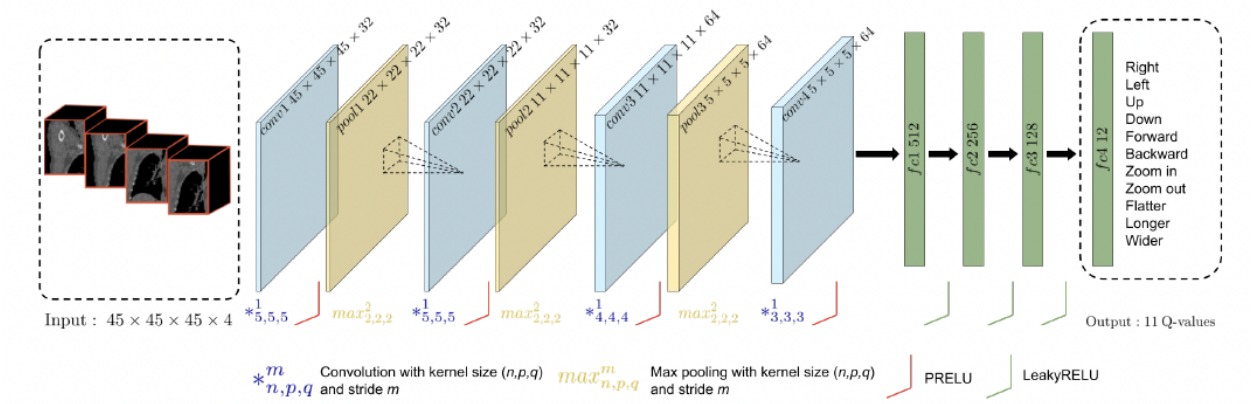


Figure 3: Deep-Q Network architecture. The CT scan voxel values inside the current bounding box represents the current states, and the current plus the last 3 states are fed into the Agent's primary Q network. The network then outputs the q-values for the 11 possible actions.

3.2 Agent

The agent moves around by taking actions from the set of 11 possible actions, which results in a change of the current location of interest. The combination of these actions will enable the agent to reach any possible location with infinite steps. At each time step, the current plus three previous states matrix is fed into the agent's primary Q network, which then outputs the q-values for each of the 11 actions. During the training, the agent follows an epsilon greedy policy to choose the action to allow both exploration and exploitation. During testing, the action with the highest q-value is always selected.

Neural Network Architecture: This network architecture was the same as demonstrated in the work of Alansary et al (2019) [1] for landmark detection. The input to the network consists of the stacked voxel values from the last four frames taken by the agent. The output of the network is the q-value distribution of possible actions, as illustrated in Figure 3.

ϵ-Greedy Policy: The agent uses an ϵ-greedy policy during training, which means it gradually moves from exploring possible actions to exploiting actions that have the highest expected reward. During training, a random number is generated before the agent chooses the action. If this number is smaller than ϵ, a random action is chosen, and if the number is greater than ϵ, the action with the highest q-value is chosen. We implemented an annealing ϵ across the training epoch to allow the player to have more exploration of all possible game states during the initial training epochs, and gradually shifted to more exploitation as the training proceeds. The equation below shows how the value of ϵ is calculated, where the decay factor equals $\min(\text{epoch} + 1/3, 5)$.

$$\epsilon = \epsilon_{min} + (\epsilon_{start} - \epsilon_{min}) \cdot \exp(-1 \cdot \frac{\text{episode}}{\text{num episode}} \cdot \text{decay factor})$$

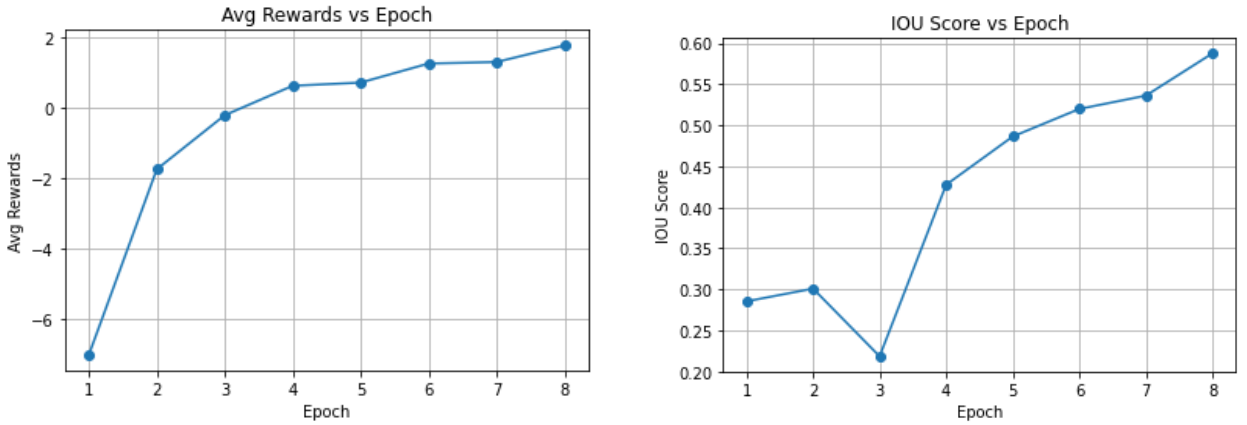
3.3 Replay Buffer

To address the issue of the strong correlation within an episode when an agent travels around the environment space, we use an experience replay buffer D to store the transitions of current states s^t, actions a^t, rewards r^t, next states s^{t+1}, and whether the game is terminated. The replay buffer class is implemented as a Deque that has a maximum store size. At the start of training, we first collect samples to fill the replay buffer. Then during training, the oldest sample will be popped from the Deque when the new sample enters. When a certain training episode is reached, we randomly sample mini-batches from the replay buffer to calculate the loss and update the weight of the primary Q-network.

4 Experiments and Results

4.1 Hyperparameter

Figure 8 in the Appendix section 6 listed all the hyperparameter values we used to train the model. In our study, we trained the model for eight epochs. For each epoch, we played 1000 game episodes and distributed them evenly across



(a) Action rewards

(b) Bounding box IoU

Figure 4: Average action rewards and predicted bounding box IoU score after each training epoch.

all training images. We chose a screen size of (90, 90, 90) to represent the current states inside the bounding box. We used the same scaling factor (10) for early termination and out-of-boundary penalties. We chose a scaling factor of 100 for intermediate IoU rewards to ensure that all rewards and penalties were on the same scale. We trained the primary network every 10 episodes and synchronized the weights between the primary network and the target network every 200 episodes to improve stability. The primary network was trained using a single GPU, with a batch size of 8 and a replay buffer of size 1000. We used a cyclic annealing scheduler with a base learning rate of 1e-6 and a maximum learning rate of 1e-5. We employed an epsilon-greedy policy for each epoch, with an initial epsilon of 0.95 that decayed at a rate proportional to the epoch number (but never lower than 0.05).

During training, we set the initial threshold for stopping the localization process to 0.5. When the Intersection over Union (IoU) was greater than the stop threshold or reached the maximum number of steps (15), the organ was considered localized and the agent was restarted on a new CT scan. Both the stop threshold and the maximum number of steps was increased along with the epoch to enable the agent to handle more complex scenarios. These values were bounded by 0.8 and 20, respectively, as additional actions could introduce new errors.

During testing, the agent stopped the search after a fixed number of steps (15 in our experiment). We recognize that this may not be sufficient to localize every object, and we plan to continue tuning this hyperparameter in the future.

4.2 Liver Localization Result

Figure 4 shows the average action rewards and bounding box IoU during training after each training epoch. We can see a clear increasing trend in both plots, and since the improvement hasn't converged, we expect the model performance to continue to increase with more training.

In Figure 4 (a), we can see that the average rewards for each action rise from around -6.8 to 1.9 after 8 epochs of training. This means that, in general, the model learns to take action that can increase the bounding box IoU, and it also learns to avoid taking action that would move the bounding box out of the boundary, since in our hyperparameter setting, the penalty of out-of-boundary action is set to be -10. In Figure 4 (b), the average liver localization IoU score increases from around 0.28 to almost 0.60. This suggests that, on average, the model is able to locate the majority part of the liver in the CT scan.

Figure 5 shows the distribution of liver localization IoU scores on both the training and testing dataset. We can see that the distribution is similar, which supports that our model can generalize well to unseen datasets. It should also be noted that the model is able to achieve over 0.7 IoU score on several training samples, indicating its potential to localize the liver in CT scans with very high accuracy. Therefore, we think it is reasonable to assume that when training with more epochs, our model can potentially obtain equivalent liver localization IoU of 0.8 as the baseline model despite having less training size (24 vs. 70).

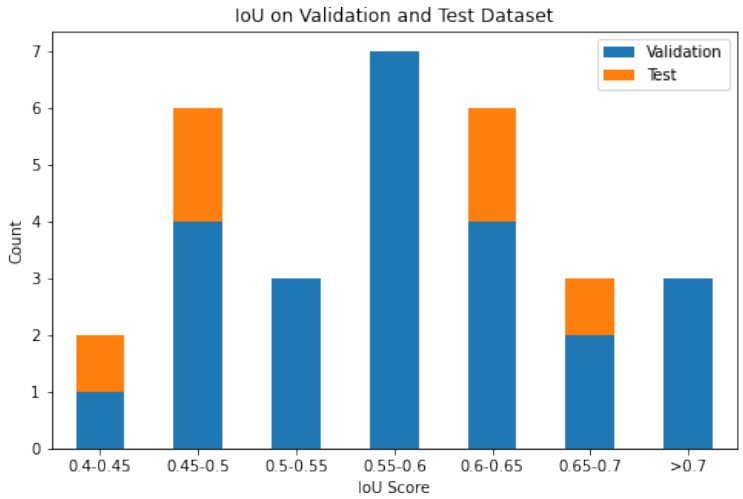


Figure 5: Distribution of predicted bounding box IoU score on training and testing dataset

4.3 Liver Localization Examples

We also visualized the final liver bounding box predicted by the model and compare it with the true bounding box label to analyze the reasons for the IoU score differences between different samples. Figure 6 shows 2 examples of final liver bounding boxes with (a) the highest and (b) the lowest IoU score. We can see that these 2 examples show a large difference in liver localization accuracy. We are able to come up with a conjecture that may explain the differences in liver localization accuracy. In Figure 6 (a), we noticed the true liver bounding box label has a relatively squared shape and the volume is also much larger, whereas in Figure 6 (b), the true liver bounding box label is rectangular and much smaller in volume. While the larger volume of the liver likely makes it easier to find, we think the shape of the true bounding box label also plays an important factor in terms of levels of localization difficulty. This is because, in our model implementation, we generate the initial bounding box to cover 50% of the contents in each axis in the center. We also limit the step size of each action to be proportional to the current bounding box length, specifically, the proportion is set to be 10%. So if the true liver bounding box label has a much smaller aspect ratio in some of the axis, as is the case in Figure 6 (b), then it would require several consecutive corresponding deformation actions (flatter, thinner, taller) to be taken in order to match the target aspect ratio. In comparison, in the case of Figure 6 (a), mostly translation and scaling action is sufficient to move the initial bounding box to the target location. Therefore, the label bounding box ratio can play an important factor in determining the level of liver localization difficulty since a lot more deformation action may be required in addition to the translation and scaling action.

5 Discussion

Our study replicated the Deep Reinforcement learning model proposed in Navarro et al. [4], and we also proposed several innovations baseline on this baseline model to optimize the model architecture. The results from our model show that the Deep Reinforcement Learning model can achieve good performance on liver localization, and it is also able to generalize to unseen test datasets. This is consistent with the results from the baseline model, and it also demonstrates the Deep Reinforcement Learning model's ability to learn from an even smaller training dataset.

5.1 Limitation

It should be noted that our study has a few limitations.

Limited Training and Hyperparameter Tuning: Due to limitations on time and computation resources, our model was only trained for 8 epochs and likely did not reach its best performance. It would be interesting to see whether our model can eventually achieve equivalent or better performance compared to the baseline with more training.

Need to Test The Model's Generalizability: It is important to verify the generalizability of our model on other 3D CT scan datasets and on other organs as well. The results from our study showed that our model can generalize well on unseen testing samples from the same 3D CT scan dataset. However, since different 3D scan datasets may use different parameters such as volumes and resolution, the model should also be tested on other datasets in order to

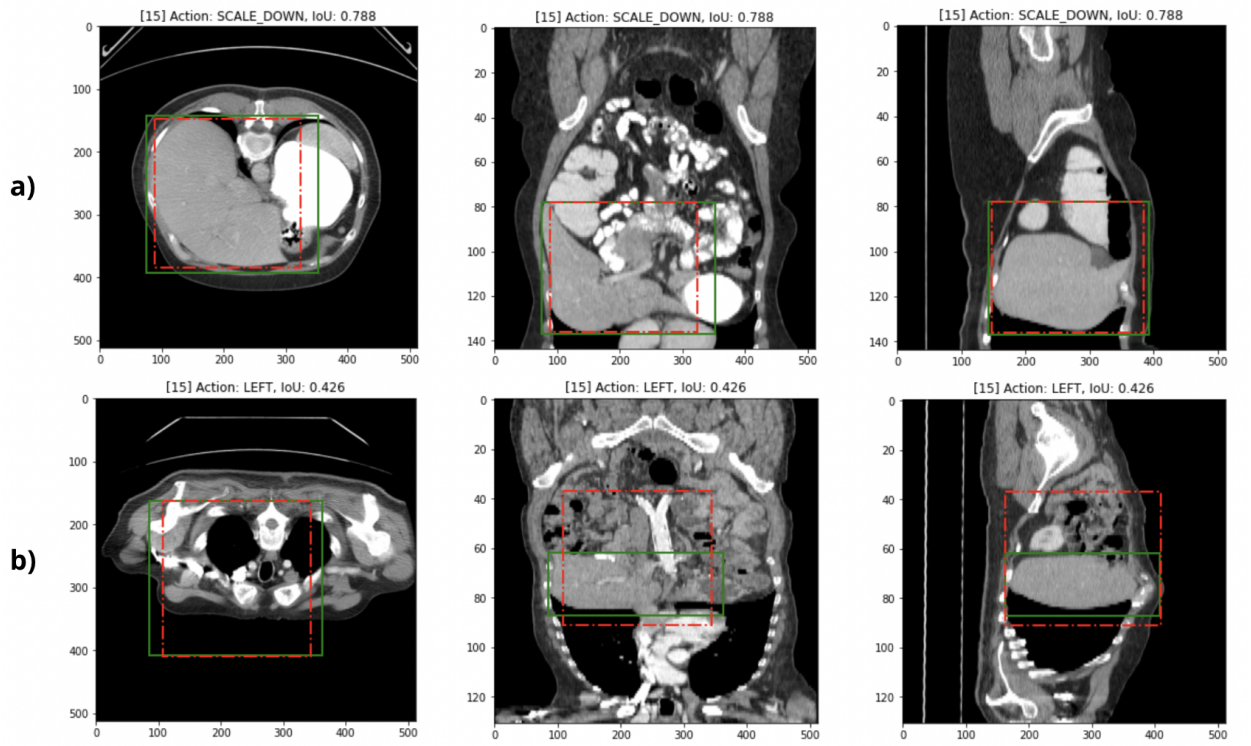


Figure 6: Examples of final bounding box for liver localization generated by the model. The green box indicates the true label location for liver, and the red box shows the model prediction. The max step size is set to 15. (a) example with the highest IoU score (b) example with the lowest IoU score

thoroughly understand its ability to generalize. Moreover, we only trained our model for the liver localization task. The baseline paper trained and measured the localization IoU score in 7 different organs, where the baseline model shows relatively poor performance in some of the organs. Therefore, it is crucial to test whether the model can perform well on localization tasks for other organs, or if adjustments and tuning are required.

5.2 Future Direction

As a future step, we would first like to continue training our model and train the model on localization tasks for different organs to explore its potential and generalizability.

Bounding Box Initialization: When analyzing the model prediction trajectory and results, we discovered some issues in the current model architecture, and we proposed some methods that can potentially be implemented to account for these issues. For the problem regarding different aspect ratios between the initial and true label bounding box, as discussed in Section 4.3, we proposed two potential methods that we think are worth testing in later research: (a) increase the maximum step size to allow more refinement on the liver bounding box prediction (b) add a certain percentage of randomness in the bounding box initialization. Specifically, we can allow the initial bounding box to generate at different locations or to cover varying content percentages so that it has different initial aspect ratios. This can push the model to explore bounding boxes with varying aspect ratios and allow it to refine the bounding box more carefully with more action steps.

Current State Representation: Another possible direction for improvement is how the current state of the environment is represented. Our current implementation followed the baseline model where the current state is represented as a matrix with a fixed screen size, which was set to be the same in all 3 dimensions. Specifically, the baseline model used a screen size of (45, 45, 45) and we used (90, 90, 90). The voxel values inside the bounding box will be transformed into the fixed screen size using affine transformation. While this is a simple method that allows the state representation to have the same size during each time step before feeding into the primary Q network, it will distort the aspect ratio of the original image in the bounding box. For instance, if the current bounding box has a size of (180, 90, 90), then after affine transformation, the current state will be compressed in the first dimension (x-axis). This may make it harder for

the primary Q network to learn the feature representation since the convolution layers are only invariant to scale, but not invariant to aspect ratio. To account for this issue, we can instead include voxel values from the adjacent region or simply pad the current bounding box to ensure the current state always has the same aspect ratio. This way the affine transformation will not distort the contents inside the current bounding box, which can help the primary Q network with the feature representation.

6 Conclusion

In this study, we proposed an optimized deep Reinforcement Learning model for 3D organ localization based on the work of Navarro et al. [4], and our results also supported their finding that the deep RL model is useful for organ localization tasks in 3D CT scans. While our study has some limitations, we believe future research can be done to explore them. We also proposed a few ideas to further improve the current model architecture. We believe the deep RL model has the potential to achieve high accuracy in organ localization tasks and generalize across different datasets and different organs. Therefore, it is possible to deploy this model in the hospital to automize the organ localization in the CT scan as a pre-processing step and reduce the time radiologists spend on analyzing CT scan results.

References

[1] Amir Alansary, Ozan Oktay, Yuanwei Li, Loic Le Folgoc, Benjamin Hou, Ghislain Vaillant, Konstantinos Kamnitsas, Athanasios Vlontzos, Ben Glocker, Bernhard Kainz, et al. Evaluating reinforcement learning agents for anatomical landmark detection. *Medical image analysis*, 53:156–164, 2019.

[2] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[3] 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.

[4] Fernando Navarro, Anjany Sekuboyina, Diana Waldmannstetter, Jan C Peeken, Stephanie E Combs, and Bjoern H Menze. Deep reinforcement learning for organ localization in ct. In *Medical Imaging with Deep Learning*, pages 544–554. PMLR, 2020.

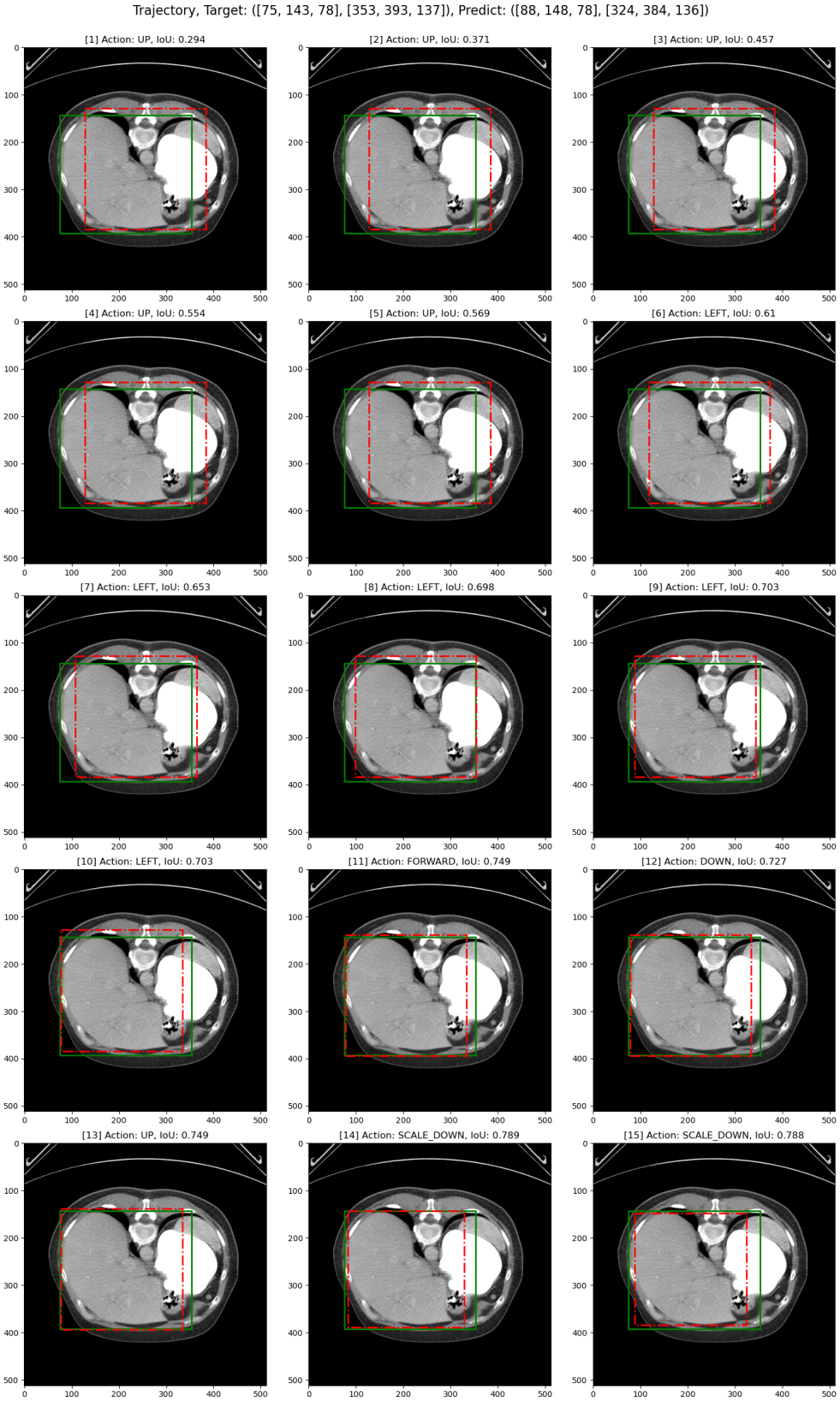
[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[6] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

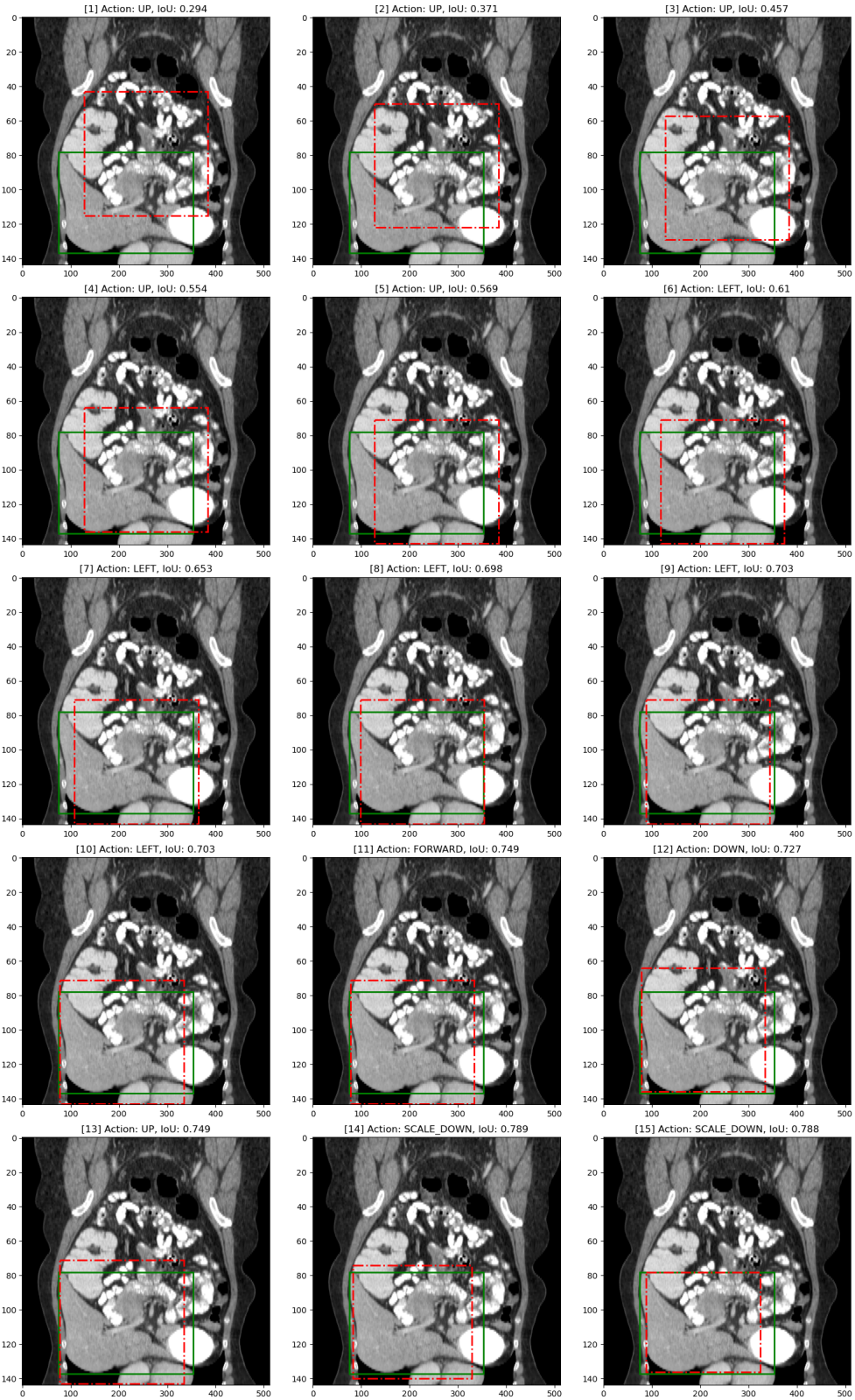
[7] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Figure 7: Example of liver localization trajectory generated by the model. Max step size is set to be 15.

(a) Axial View



(b) Coronal View



(c) Sagittal View

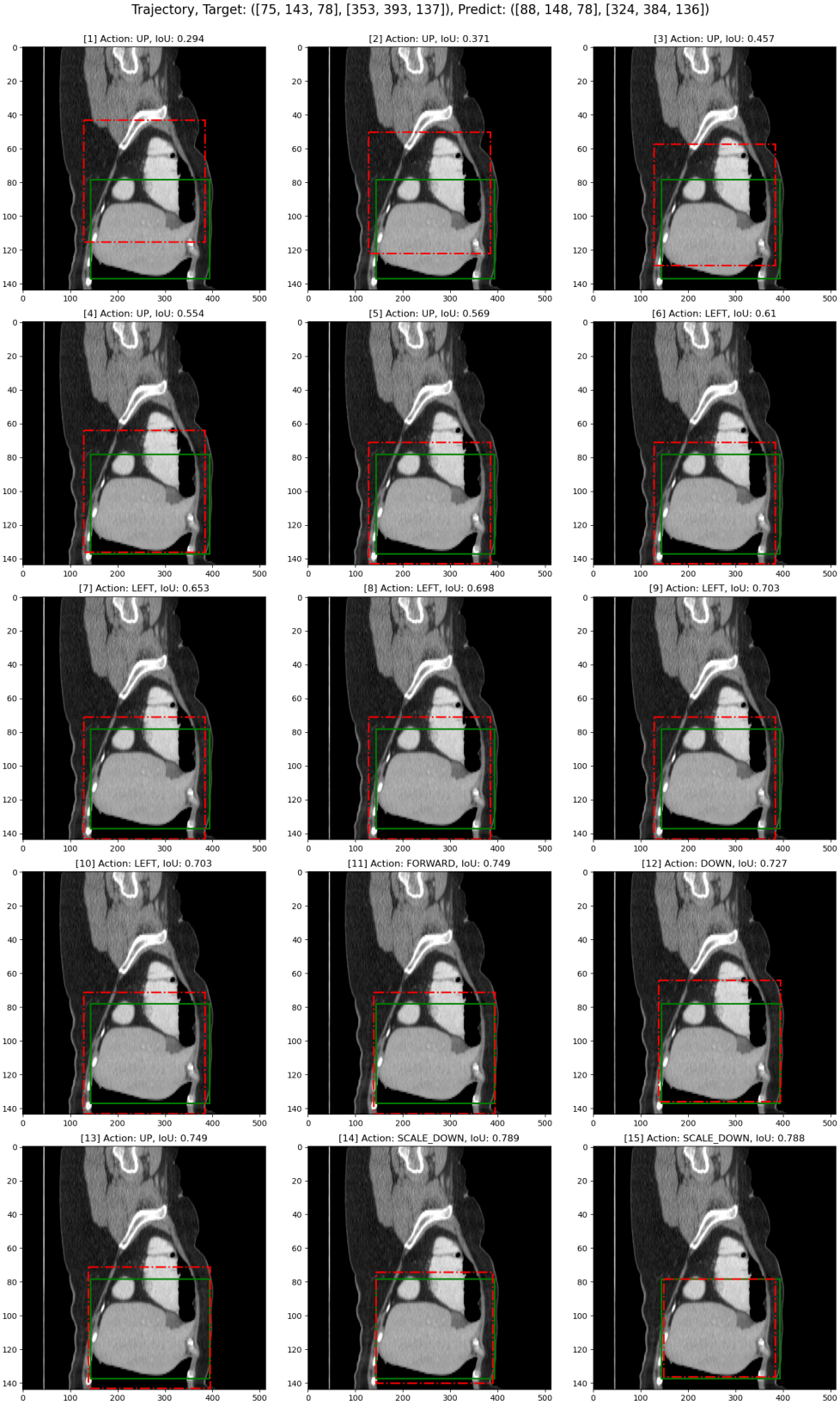


Figure 8: Config file containing all the hyperparameter values we used to train the model

```
# Dataset
config.data_dir = "/home/jupyter/capstone/RawData"
config.model_folder =  "/home/jupyter/capstone/model/" + config.time_stamp
config.model_name = "model"
config.log_folder = "/home/jupyter/capstone/log"
config.log_file = 'log.log'

# Dataset
config.batch_size = 8
config.dataloader_size = 24

# Training parameter
config.device = "cuda" if torch.cuda.is_available() else "cpu"
config.start_epoch = 1
config.num_epoch = 8 - config.start_epoch + 1
config.num_episode = 1000
config.train_frequency = 10
config.save_frequency = 100
config.update_frequency = 200
config.validate_frequency = 500
config.epsilon_min = 0.05
config.epsilon_start = 0.95
config.epsilon_decay = 1

# Reply buffer
config.max_memory_size = 1000

# Environment
config.screen_dims = (90, 90, 90)
config.max_steps = 15
config.init_terminate_iou = 0.50
config.terminate_iou_step = 0.05
config.max_terminate_iou = 0.85
config.validate_terminal_iou = 0.75
config.action_step_ratio = 0.1
config.max_action_step = 10
config.min_action_len = 10
config.history_length = 4
config.num_actions = 11

# DQN Training
config.in_features = int(512 * math.pow(config.screen_dims[0] / 45, 2)
                         * math.pow(config.screen_dims[0] / 45, 2) * math.pow(config.screen_dims[0] / 45, 2))

# Weight update
config.optimizer = "AdamW"
config.lr_scheduler = "CyclicLR"
config.lr_update = 200
config.lr_decay_episode = 0
config.step_size_up =  int(config.num_episode / config.lr_update) # step up time for cyclic lr scheduler
config.lr = 1e-4
config.min_lr = 5e-6
config.gamma = 0.9 # step size for exponential lr scheduler
config.discount_factor = 0.9

# Reward Function
config.reward_method = 'iou_diff'
config.iou_reward_scale = 100
config.terminal_reward = 10
config.go_out_reward = -10
```