

SvgAI – Training Artificial Intelligent Agent to use SVG Editor

Anh H. Dang*, Wataru Kameyama**

*GITS, Waseda University, Tokyo, Japan

** Faculty of Science and Engineering, Waseda University, Tokyo, Japan

danghoanganh@akane.waseda.jp, wataru@waseda.jp

Abstract— Deep reinforcement learning has been successfully used to train artificial intelligent (AI) agents to outperform humans in many tasks as well as to enhance the capability in robotic automation. In this paper, we propose a framework to train an AI agent to use scalable vector graphic (SVG) editor to draw SVG images. Hence, the objective of this AI agent is to draw SVG images that are similar as much as possible to their target raster images. We find that it is crucial to distinguish the action space into two sets and apply a different exploration policy on each set during the training process. Evaluations show that our proposed dual-exploration policy greatly stabilizes the training process and increases the accuracy of the AI agent. SVG images produced by the proposed AI agent also have superior quality compared to popular raster-to-SVG conversion software.

Keywords— SvgAI, Reinforcement Learning, SVG, Exploration Policy, Q-Learning

I. INTRODUCTION

Besides defeating the world best human player in Go [1], AI agents (hereafter referred as agent) trained by deep reinforcement learning (RL) [2] has achieved human-level control in a wide variety of tasks like 3D first-person shooter (FPS) game playing [3], and enhances the capability of robotic automation [4]. For example, Mnih et. al. have introduced Deep Q-Network (DQN) [5] that plays Atari 2600 games well above the skill of human players or any other linear models. Subsequently, the works on prioritized experience replay [6], double Q-network [7], duel Q-network [8] and asynchronous actor-critic method [9] further enhance the efficiency of the training process.

On the subject of raster-to-vector (R2V) conversion, Karpathy et. al. present a breakthrough work [10] on training a deep neural network (DNN) [11] for automatic image description. Beltramelli proposes Pix2Code [12] DNN that generates Extensible Markup Language (XML) based code from raster screenshot of graphical user interface (GUI). Despite being a mature branch of research, image processing based R2V conversion is not yet reliable [13]. Therefore, we propose a framework to train an agent to use SVG editor (hereafter referred as editor). The objective of this agent is to draw an SVG image that is similar as much as possible to a given a raster target image. It can be considered as a new paradigm for R2V solution.

In this paper, we concentrate on exploring the feasibility of this new paradigm by training the agent to work on plain target image. A custom editor is created for carrying out the research. This editor has modeled after OpenAI Gym [14] environment due to its robustness in interface design and potential compatibility with available agents. To successfully train the agent, we separate the action space into two sets and apply a separated exploration policy on each set during the training process. The detail of the editor, its action space, and dual-exploration policy are explained in Section III.

We have trained the agent under Q-Learning paradigms with weighted memory replay, and evaluate the agent performance by comparing the similarity between SVG images and target images. We also compare the effectiveness of the dual-exploration policy with the conventional methods. Finally, we compare the quality between the SVG image produced by the proposed agent with the SVG image produced by popular R2V software. Section IV describes the agent model. Section V and VI describe the framework, training and evaluation process in detail. Section VII concludes the paper with possible improvement and future works.

II. RELATED RESEARCH

The agent needs to be trained ultimately converts a raster image to a vector equivalent representation by using the editor. Therefore, this research is related to not only AI research but also series of visual-language cross model works. There are two categories of related visual-language cross model works: image processing based R2V and deep learning based R2V.

A. Image Processing Based R2V

Despite being a mature branch of research, image processing based R2V conversion is not yet reliable [13]. Major problems are difficulties of color quantization, aliasing effects, shift and superposition effects, texture and text miss-identifying [15].

There are numerous works to solve the above-mentioned problems. For example, Kansal et. al. propose a framework [16] to reproduce linear filled gradient. Vector representation of halftone dots in binary images is presented by Kawamura et. al. [17]. However, they leave the works of identifying the fundamental problems to human operator. Thus, for example, well-known conversion tools such as Potrace [18] still requires human intervention to achieve desirable results [19].

B. Deep Learning Based R2V

Image annotation has been an active research area for a long time. This topic of research becomes overly crowded recently due to the advancement in deep learning based natural language processing and computer vision. Karpathy et. al. achieve a breakthrough [10] with an end-to-end DNN model. Learning only from images and corresponding annotations, this model can then annotate image at multiple hierarchical levels. It also detects and locates objects within the images. This model is realized by the embedding visual model using RCNN (Regional Convolutional Neural Network) [20] and the language model using BRNN (Bi-directional Recurrent Neural Network) [21] into the same multimodal space.

Beltramelli proposes Pix2Code [12], an end-to-end DNN that generates XML based GUI code from mock-up images. Even though similar to [10] in general design, the visual model used in Pix2Code is a plain CNN block while language model is handled by a Long-Short Term Memory (LSTM) network [22] block. Another LSTM block is used to decode outputs of visual and language models into code tokens. This work can be understood as a rigid version of automatic image annotation extractor. Although this model can produce GUI code from a raster image, it is not truly flexible because the visual presentations of all the GUI elements in this work are templated. Thus, the visual representation variance of input images is relatively low, and it only works with GUI images based on the fixed templates.

C. Reinforcement Learning

1) Q-Learning: The agent in deep reinforcement learning holds a policy set that ultimately decides which action to be performed in the next step. At each state of time step t , the agent observes the state s_t of the environment, and decides action a_t based on its current policy π . Reward r_t is then given to the agent as a feedback from the environment. The objective of deep reinforcement learning is to train the agent so that its policy will result in maximized reward in the future. The expected reward is calculated as follows.

$$R_t = \sum_{i=t}^T \gamma^{t'-t} r_{t'} \quad (1)$$

Where R_t is the sum of expected discounted reward at time step t , T is the time when the operation is terminated, and γ is the discount factor which lies within the range of $[0 \dots 1]$. The higher the value is, the more important the future reward. The Q -value of an action according to policy π is calculated as follows:

$$Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a] \quad (2)$$

Hence, $Q^\pi(s, a)$ is essentially an expected future reward if the agent performs action a in the given state s . In practice, the action that gives the highest Q value is chosen to be executed by the agent. The optimal value $Q^*(s, a)$ is defined as follows:

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a] \quad (3)$$

Hence, using (1) and (2), it can be written as follows:

$$Q^*(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} [Q^*(s', a') | s, a] \right] \quad (4)$$

Where s' and a' are the state and the action of subsequent time step, respectively, and r is the immediate reward for action a given in state s . In deep Q -learning, this value is approximated by the neural network parameterized by θ :

$$Q_\theta(s, a) \approx Q^*(s, a) \quad (5)$$

Thus, if the neural network approximates the Q value well, the Q value of an action a can be estimated as follows:

$$Q_\theta(s, a) \approx r_t + \gamma \max_{a'} Q_\theta(s', a') \quad (6)$$

Given that $y_t = r_t + \gamma \max_{a'} Q_\theta(s', a')$, the loss function is defined as follows:

$$L_t(\theta_t) = \mathbb{E}_{s,a,r,s'} \left[(y_t - Q_{\theta_t}(s, a))^2 \right] \quad (7)$$

In practice, instead of squared difference loss, Huber loss is usually used to stabilize the training process:

$$L(\alpha) = \begin{cases} \frac{1}{2} \alpha^2 & \text{if } |\alpha| \leq \sigma \\ \sigma \left(|\alpha| - \frac{1}{2} \sigma \right), & \text{otherwise} \end{cases} \quad (8)$$

Where α is the difference between y_t and $Q_{\theta_t}(s, a)$.

2) Experience Memory Replay: One of the most significant difficulties in training an agent is the strong correlation between the network policy and the action outcome of subsequent time steps. This difficulty makes online training impossible. To break the strong correlation, experience memory replay [6] is used.

In the experience memory replay, at every time step, the experience (s_t, a_t, r_t, s_{t+1}) of agent is stored in replay memory. This replay memory is a large capacity queue, and is usually set to store one million experiences. When it's full, the oldest experience in the queue is removed to make place for a new experience. The network is trained using mini batches randomly drawn from this memory.

3) Exploration in RL: Training an agent using reinforcement learning requires a right balance between exploitation and exploration. Exploitation is relying on the learned policy to improve the prediction accuracy while exploration allows the agent to seek for potential better solutions (i.e. avoiding sub-optimal trap). A popular exploration policy being used in RL is ϵ -greedy [2]. Under this policy, the output of the agent has a ϵ chance of being random.

There are other exploration policies based on randomization such as Thompson sampling [23], Bayesian sampling [24]. However, a variance of ϵ -greedy policy named reducing ϵ -greedy is most commonly used. With this policy, the agent starts with high exploration rate, and gradually reduces it over time. Thus, it allows the agent to avoid biased exploration in

the beginning, and to focus more on exploitation in the later phase of the training process.

III. SVG EDITOR

Like other reinforcement learning settings, the proposed framework consists of two parts: the agent and the environment. As shown in Figure 1. in this research, the editor is playing the role of the environment.

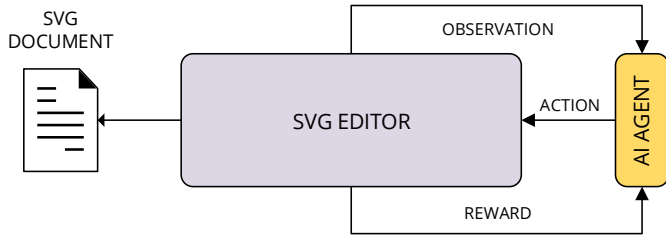


Figure 1. The proposed framework of SvgAI. The editor is playing the role of the environment as seen in many AI research.

A. SVG Construction

SVG is an XML based vector image format. An SVG is composed by different type of elements. Previous works on R2V usually produce SVG image using path element exclusively because this element is flexible and can be used to form any type of shape. Furthermore, more fundamental shape elements such as a rectangle, circle or arc not only requires the higher level of visual understanding but also comes with more none trivial challenges.

On the other hand, using large number of elements and path significantly increases the SVG image size and requires more computational resource to be displayed. In this research, we utilize two fundamental shapes (with their transformations) that are rectangle and circle for SVG image construction.

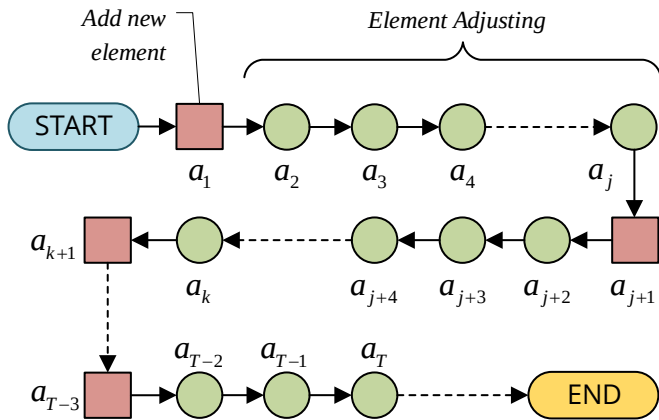


Figure 2. The process of SVG composition: the agent keeps adding and editing new elements until the desired result is achieved.

Our editor allows the agent to either add a new element into the SVG image or edit the added element. In the beginning of each episode, the editor produces a blank image. For every time step, the agent observes the state of the editor (step 1). The agent processes the observed state, and sends a new action to the editor (step 2). The editor performs the action as requested

(step 3), and the process goes back to step 1. Hence, during the process, the agent is incrementally adding and editing new elements to and in the image. Finally, the editor saves the image into an SVG file. The process is illustrated in Figure 2.

The newly added element have a default presentation when it's firstly added. The default presentations of the circle and square elements are shown in sub-figure Circle-0 and Square-0 in Figure 4. As the agent keeps editing, the presentation of the element is to be updated. For example, subfigure Circle-400 in Figure 4. is the presentation of a circle element after 400 steps of editing.

B. Action Space

With the above-explained process, once a new element is added to the SVG image, the old element is no longer editable. Thus, the consequence of adding new elements is more significant compared to editing action. Therefore, it is crucial to distinguish the two sets of actions, set A and set B, and apply separated exploration policies during the training process as explained in Section IV-B. Otherwise, it is impossible for the model explained in Section IV to converge. TABLE 1 lists all the actions of set A and B supported by the editor.

TABLE 1. LIST OF ACTION SUPPORTED BY THE EDITOR

Set	Action Id.	Description
A	0, 1	Add circle and square element
B	2, 3, 4, 5	Move element left, right, up, down
B	6, 7, 8, 9	Compress, expand element horizontally, vertically
B	10, 11	Rotate the element clockwise/counter clockwise
B	12, 13	Reduce, increase element's line thickness
B	14, ..., 21	Adjust ^a element's line color (RGB α)
B	22, ..., 29	Adjust ^a of element's fill color (RGB α)

^a. Increase/decrease value of each channel

C. State Observation

The state of the environment is sent to the agent to decide the next action. The state of the editor at any time step t consists of 3 components:

1. Raster version of the image being edited I_t .
2. Raster target image Y , i.e. the image which the agent attempts to compose by using the editor.
3. An auxiliary vector that describes the state of the SVG element.

All raster images are in the RGB α format with the resolution of 128 by 128 pixels. Because the task of the agent is to make an image transformed into SVG, it is essential to retain color information of the target. Thus, each state has over 65k parameters including the auxiliary vector.

IV. MODEL

Figure 3 shows the architecture of the proposed agent. The stack of raster images is processed by convolutional neural network (CNN) blocks. These blocks produce around 27k parameters. The combination of this vector and the auxiliary vector is then processed by the fully connected block to produce a score for each action in the environment's action set.

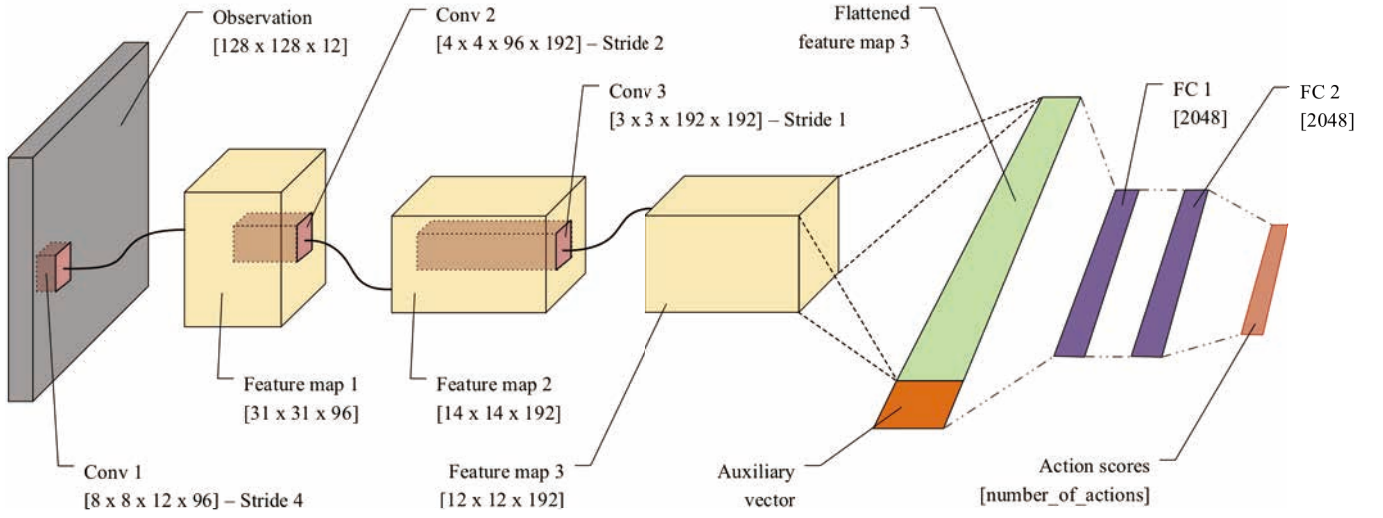


Figure 3. The architecture of the agent. The first three state components mentioned in Section III-C are processed by the convolution layers. Auxiliary vector is concatenated with the output of the convolution layers before feeding to fully connected blocks at the end of the network.

A. Error and Reward

With every action received from the agent, the error is calculated as follows:

$$\varepsilon_t = \frac{\sum |I_t - Y|}{\max(\sum I_t, \sum Y)} \quad (9)$$

Where ε_t is the error at time step t , which describes the difference between a rasterized version I_t of the SVG being composed and the target image Y . $\sum I_t$ and $\sum Y$ are summations of all pixels over I_t and Y , respectively. Thus, the similarity between the two images s_t is defined as follows:

$$s_t = 1 - \varepsilon_t \quad (10)$$

Base on the similarity s_t , the reward is given to the agent as follows:

$$r_t = \begin{cases} 1 + \frac{e^{s_t}}{e} & ; \text{if } s_t > s_{t-1} \\ 0 & ; \text{otherwise} \end{cases} \quad (11)$$

Where e^{s_t} is the exponential of s_t , and it is divided by e constant. Therefore, the environment returns 0 reward when the action results in no improvement, and returns a small reward ranging from 1 to 2 depending on the similarity between the result and the target image.

Given n_t which is the number of SVG elements at time step t , the penalty p_t is given as follows:

$$p_t = \begin{cases} -1 & ; \text{if } n_t > n_{t-1} \\ 0 & ; \text{otherwise} \end{cases} \quad (12)$$

B. Exploration Policy

In our early attempts, the conventional reducing ϵ -greedy exploration policy was used. On average, with the action set explained in Section III-B, there was a new element added into the image for every 14 adjustments in exploration steps. Regardless the value set to exploration rate ϵ , actions in set A were explored more than necessary. Hence, it was considered

as bad exploration most likely making the network to be trapped in sub-optimal solution [25].

The simplest solution is to apply lower weight for the action set A during the randomizing process. Given W which is the set of weight corresponding to the action space. The probability for an action being drawn by the ϵ -greedy policy is:

$$P(W) = \frac{\epsilon W}{\sum W} = \frac{\epsilon W_A \oplus \epsilon W_B}{\sum W} \quad (13)$$

Where W_A and W_B are weights for the action set A and B, $\oplus$ is concatenation operation. However, we observe that the agent is eternally biased towards a certain element (either circle or square). A possible explanation for this problem is that the ϵ is already saturated when the agent doesn't learn yet the long-term reward of adding the right element.

Prolonging the exploration phase does not help either because the agent has to be trained in low ϵ value to learn the long-term reward for using each element. Theoretically, the problem will be eventually solved after many iterations in this setting. However, it is inefficient due to the low probability of random element spawning.

Our approach to solve this problem is to apply dual-exploration policy. In dual ϵ -greedy setting, there are two separated reducing ϵ -greedy policies applied for each action set:

$$P(W) = \frac{\epsilon_A W_A \oplus \epsilon_B W_B}{\sum W} \quad (14)$$

Where ϵ_A and ϵ_B are values of two ϵ -greedy policies for the action set A and B, respectively. In this way, the action set A and the action set B can be independently explored.

V. EXPERIMENT

A. Dataset

Previous works on image processing based R2V have widely diverse objectives. As the result, they mostly use relatively plain and small datasets. The diversity in research objectives also leads to the lack of unified evaluation dataset [15]. While

the complexity of these data sets is suitable for this research, their modest size and heterogeneous properties are not adequate to be used to train DNN.

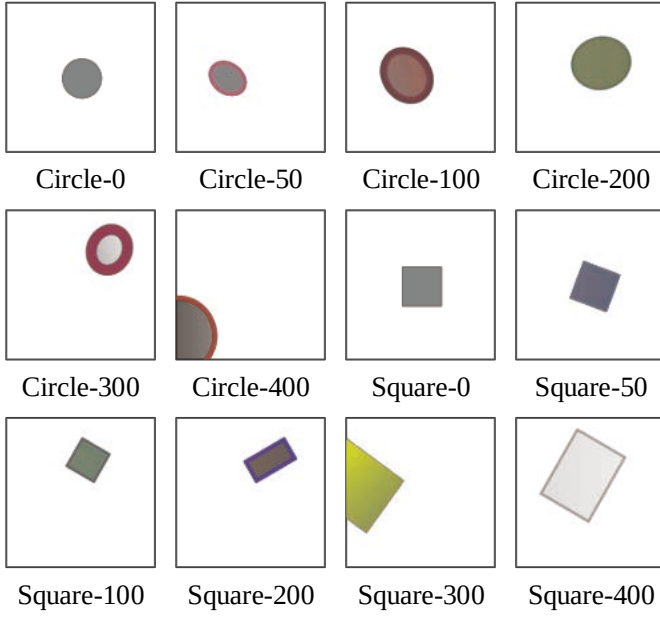


Figure 4. Examples of images generated for training and evaluation. The caption under each image shows the name of the shape and the number of steps required to transform from the default presentation (i.e. Square-0 and Circle-0) using the editor.

With the above reasons, the both of the training and the evaluation of the proposed method have been done on a synthesized data set. Working on synthesized data not only helps in avoiding the above-mentioned problems, but also provides a controlled level of difficulty and the uniformity of the dataset.

Figure 4. shows the examples of images generated for the training and the evaluation processes. The first (Circle-0) and the sixth image (Square-0) are the default presentations of the two shapes as explained in Section III-A. The difficulty of the target image is controlled by the minimum number of steps required for the agent to transform from the default presentations.

In the beginning of each episode, a random target image is generated which contains a single shape with random difficulty. The episode is terminated if one of the following conditions is met:

- After a fixed time step T . Since there is no further step after the termination, the Q value as mentioned in (6) at the final step T is: $Q_\theta(s_T, a_T) \approx r_T$.
- When ε_t in (9) is less than a certain threshold σ .

B. Frame Skip

Due to the computational demand of the agent, it is inefficient to let the agent perform an action at every time step. The popular solution is the frame skip [14] where the agent only interacts with the environment in every k steps. Therefore, once an action is decided by agent, it repeats k -time steps. A

popular value of k in many tasks is 4 as it is usually a good tradeoff between the performance and the training speed.

In this problem set, dismissing frame skip as the agent approaching convergence helps to improve the performance of the agent, because one action repeated several times makes the agent miss its target by overshooting the target.

C. Parameter Update

The agent does not update its parameter at every time step, but once for every L experiences added in the replay memory. Thus, given the batch size of N , one experience is learned by the agent $\frac{N}{L}$ times in average.

VI. EVALUATION

A. Exploration Policy

We have implemented the deep Q-Network scheme as proposed in [5] for the training process, and evaluate the performance of the agent trained under different exploration policies:

- Reducing ϵ -greedy policy.
- Reducing ϵ -greedy policy with weighted action set: this policy follows equation (13).
- Dual-reducing ϵ -greedy policy: this policy follows equation (14).

TABLE 2. HYPER-PARAMETER SETTING FOR EXPERIMENTS

	Greedy	Weighted	Dual
Minibatch size	32	32	32
Train Interval	16	16	16
Discount factor γ	0.999	0.999	0.999
Memory replay size	1e6	1e6	1e6
ϵ -greedy (max - min)	1 - 0.1	1 - 0.1	-
ϵ -greedy (start - end)	0 - 1e6	0 - 1e6	-
ϵ_A -greedy (max - min)	-	-	1e-3 - 0
ϵ_A -greedy (start - end)	-	-	5e6 - 10e6
ϵ_B -greedy (max - min)	-	-	0.999 - 0.1
ϵ_B -greedy (start - end)	-	-	0 - 1e6
W_A	1	1e-3	1
W_B	1	999e-3	1

We train the agent over 175 thousand updates and observe a favorable result. TABLE 2 show the hyperparameter used for experiments. Figure 5. shows the average evaluation score of the agent during the training process under the three exploration policies. The evaluation score is calculated using equation (10). The mean score is calculated by averaging the score of 15 adjacent evaluation steps. Our proposed dual-exploration policy not only shows significant performance gain but also highly stable compared to others policies during the training process.

B. Accuracy

Even though archiving superior performance using dual-exploration policy for the training process, our trained agent appears to be more “skillful” in drawing circle. To clearly observe the difference in performance of the agent in detecting

and editing different elements, we divide evaluation dataset into two sets: the circle set and the square set. The circle set consists of target images that contain circle element and the square set consist of target images that contain square element.

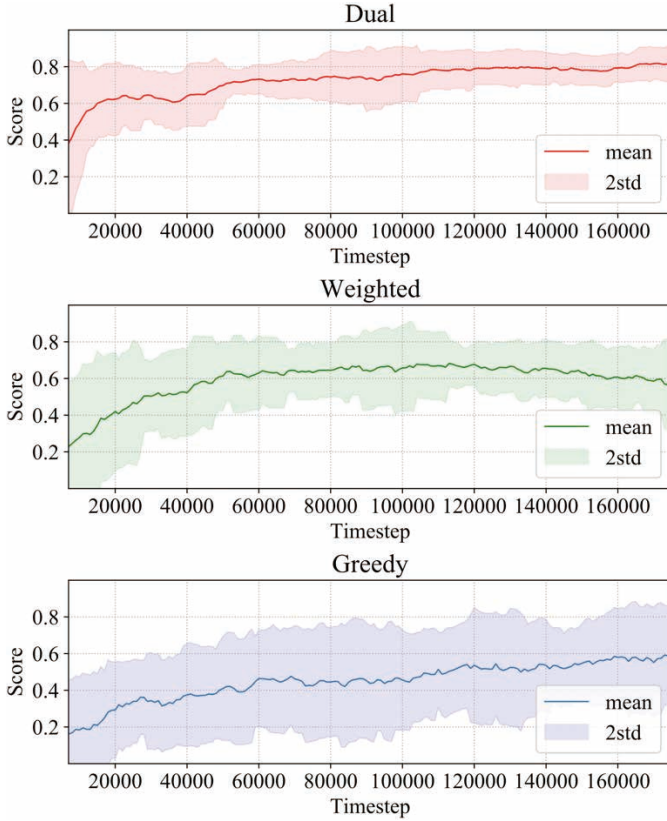


Figure 5. Average evaluation score of the agent throughout the training process under different exploration policies. The 2std, the range between $\pm 2 \times \text{standard deviation}$ of the evaluation score, is shown in each one. The small 2std area shows more stable training process.

TABLE 3 shows the evaluation of the agent under several aspects using the two datasets:

- Shape detection accuracy: the accuracy of the agent's shape detection is shown by the correctness of new element added into the image. Correctly detecting a shape is also important since further editing an incorrect element results in a large number of bad experience that negatively affects the agent's policy network.
- Performance of element editing: while incorrect shape detection poses significant weakness, the agent performs circle shape adjustment well even in the case of incorrect shape detection.

TABLE 3. AGENT PERFORMANCE EVALUATION ON CIRCLE SET AND SQUARE SET

	Circle Set	Square Set
Accuracy	94%	76%
Evaluation Score (Circle)	0.93	0.78
Evaluation Score (Square)	0.72	0.89

C. SVG Quality

We visually compare the SVG image produced by our agent with two popular opensource and commercial R2V solutions: Potrace and AutoTrace [26]. Without manual configuration prior to the conversion/drawing process, our agent produces significantly better results on the generated dataset.

Figure 6. shows the comparison between the outputs. As shown in the figure, Potrace not only has a problem on color quantization but also results in eddy circle. On the other hand, AutoTrace produces a much better result, however, the linear fill gradient has been converted into color blobs.

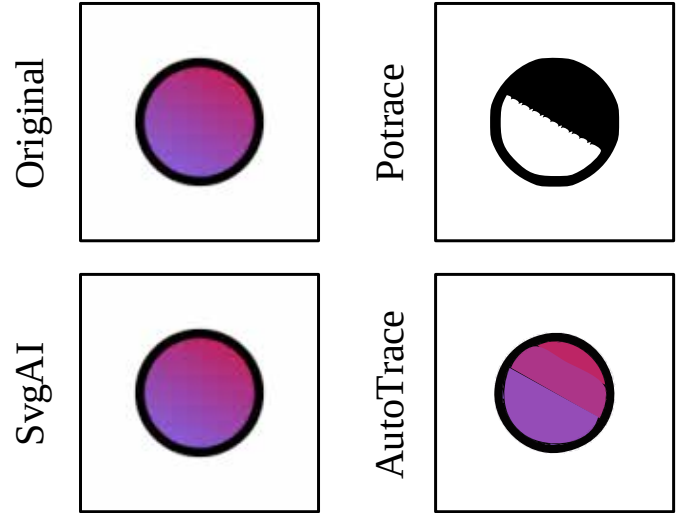


Figure 6. Comparison between SVG images produced by our agent (SvgAI), Potrace and AutoTrace

Not only visually better, the produced SVG image by our agent is also smaller both in file size and node count. As shown in TABLE 4, SVG images produced by our agent are 40% smaller in size and 63% smaller in node count than the second best solution in average over 100 images.

TABLE 4. SVG SIZE COMPARISON

	Original	SvgAI	Potrace	AutoTrace
File Size	5.8KB	852B	1.4KB	15.5KB
Node Count	-	1	2.7	174
Color Count	$\geq 16\text{mil}$	$\geq 16\text{mil}$	2	10

VII. CONCLUSION

In this paper, we introduce a new paradigm to solve raster-to-vector conversion problem. We propose an agent model and a framework to train the agent to use SVG editor. In order to successfully train the agent, we divide the action space into two sets, and apply separated exploration policies on each action set. Evaluation results show the efficiency of the proposed dual-exploration policy and the superiority in SVG image quality produced by our agent compared to the popular software solutions. The problem of incorrect shape detection as explained in Section VI-B shows a weakness of Q-Learning in applying to this problem. For future works, we will investigate

and improve the training process for better reward backpropagation for better shape detection accuracy. Policy gradient is difficult to implement and train, however, it is a promising solution to solve the above-mentioned problem.

REFERENCES

- [1] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.
- [2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1, no. 1. MIT press Cambridge, 1998.
- [3] G. Lampe and D. S. Chaplot, "Playing FPS Games with Deep Reinforcement Learning," in *AAAI*, 2017, pp. 2140–2146.
- [4] Lin and Long-Ji, "Reinforcement learning for robots using neural networks." Carnegie Mellon University, 1992.
- [5] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–33, Feb. 2015.
- [6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," Nov. 2015.
- [7] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press, pp. 2094–2100, 2016.
- [8] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. JMLR.org, pp. 1995–2003, 2016.
- [9] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, vol. 48, pp. 1928–1937.
- [10] A. Karpathy and L. Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 664–676, Apr. 2017.
- [11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.
- [12] T. Beltramelli, "pix2code: Generating Code from a Graphical User Interface Screenshot," May 2017.
- [13] H. S. M. Al-Khaffaf, A. Z. Talib, and R. A. Salam, "Empirical performance evaluation of raster-to-vector conversion methods: A study on multi-level interactions between different factors," *IEICE Trans. Inf. Syst.*, vol. 94, no. 6, pp. 1278–1288, 2011.
- [14] G. Brockman *et al.*, "OpenAI Gym," Jun. 2016.
- [15] V. Lacroix, "Raster-to-Vector Conversion: Problems and Tools Towards a Solution A Map Segmentation Application," in *2009 Seventh International Conference on Advances in Pattern Recognition*, 2009, pp. 318–321.
- [16] R. Kansal and S. Kumar, "A framework for detection of linear gradient filled regions and their reconstruction for vector graphics," 2013.
- [17] K. Kawamura, H. Watanabe, and H. Tominaga, "Vector representation of binary images containing halftone dots," in *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)*, pp. 335–338.
- [18] P. Selinger and P. Selinger, "Potrace: a polygon-based tracing algorithm," IN [HTTP://POTRACE.SOURCEFORGE.NET](http://POTRACE.SOURCEFORGE.NET), 2003.
- [19] X. Hilaire and K. Tombre, "Robust and accurate vectorization of line drawings," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 6, pp. 890–904, Jun. 2006.
- [20] R. Girshick, "Fast R-CNN," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.
- [21] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997.
- [22] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
- [23] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.
- [24] J. Asmuth, L. Li, M. L. Littman, A. Nouri, and D. Wingate, "A Bayesian sampling approach to exploration in reinforcement learning,"

in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2009, pp. 19–26.

- [25] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.
- [26] M. Weber, "Autotrace-converts bitmap to vector graphics." 2004.



and computer vision.

Anh H. Dang (S'09) received his bachelor degree in business administration, information & communication technology from Ritsumeikan Asia Pacific University (Beppu, Oita, Japan) in 2010. He then received the master degree in computer science from Waseda University (Shinjuku, Tokyo, Japan) in 2012. Since 2012, he is a Ph.D. candidate at Waseda University. He is a member of IEEE, ACM, and IEICE. His research interests are machine learning, artificial intelligence,



Prof. Wataru Kameyama (M'86) received the bachelor's, master's, and D.Eng. degrees from the School of Science and Engineering, Waseda University, in 1985, 1987, and 1990, respectively. He joined ASCII Corporation in 1992, and was transferred to France Telecom CCETT from 1994 to 1996 for his secondment. After joining Waseda University as an Associate Professor in 1999, he has been a Professor with the Department of Communications and Computer Engineering, School of Fundamental Science and Engineering, Waseda University, since 2014. He has been involved in MPEG, MHEG, DAVIC, and the TV-Anytime Forum activities. He was a Chairman of ISO/IEC JTC1/SC29/WG12, and a Secretariat and Vice Chairman of the TV-Anytime Forum. He is a member of IEICE, IPSJ, ITE, IEEEJ, and ACM. He received the Best Paper Award of Niwa-Takayanagi in 2006, the Best Author Award of Niwa-Takayanagi in 2009 from the Institute of Image Information and Television Engineers, and the International Cooperation Award from the ITU Association of Japan in 2012.