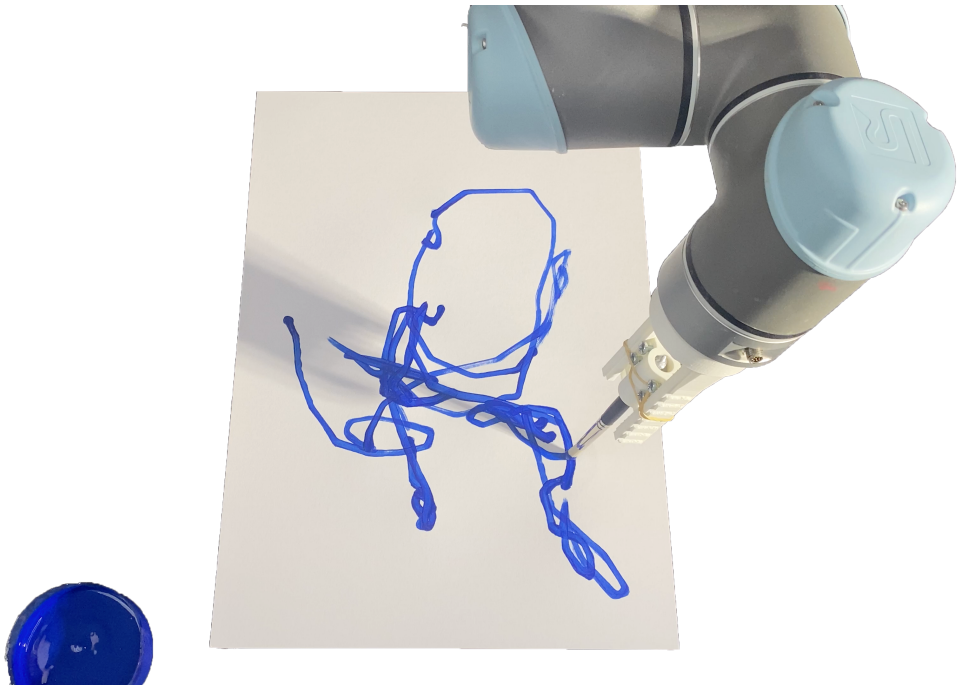
Algorithmic Muse: Robotic-Arm Learns to Draw Humans

Alessandro Mac-Nelly
Universitat der Kunste
MA Design & Computation
a.mac-nelly@udk-berlin.de

Abstract

This study explores training an AI Painting-Model, Algorithmic Muse, with Reinforcement Learning (RL) to draw abstract paintings of humans. Text to Image generators can already produce art images, but results are only pixel images representing the final result. Instead, our focus lies on how the drawing is created as a process of actions, including the brush strokes and calculating these strokes step by step. Picasso's Bull drawings 1945 are an example of this technique. Building a reward function that leads to good training results is a central topic this research focused on, and it includes evaluating artistic outcomes, evaluating a stroke, and giving the agent incentives to explore the canvas. This necessary training environment was built based on Gymnasiums Env Class and is compatible with standard RL frameworks. We used the Proximal Policy Optimization (PPO) implementation from Stable-Baselines3 and trained it in the drawing environment with positive results. In practice, a mix of AI predictions and a tree search to compare potential strokes was used to create a drawing. Our study, which included 200 subjects, evaluated and ranked 80 AI drawings, showing that some AI generated artwork is as aesthetically pleasing as human drawings. The vision of this project is to make Algorithmic Muse, while being connected to a robotic arm, collaborate with human artists in drawings where they take turns in the process.



I. Problem

Generative Adversarial Networks (GANs) [3] have significantly impacted the art scene, evidenced by Jason M. Allen's AI-generated artwork win at the 2022 Colorado State Fair [1]. These technologies, like Midjourney [2] and Dall-E2 [3], create art from text prompts. Yet, they face criticism for lacking the nuanced, iterative creation process inherent in human artistry and for potential intellectual property issues, as they replicate the features of their training datasets [14].

To overcome this, I propose Algorithmic Muse, a novel AI Painting Model designed to mimic the nuanced human approach to drawing, where the focus lies on the creation process. Using an Agent trained with Reinforcement Learning in a drawing environment native to humans, either a digital environment or a robotic arm on a canvas, enables the Agent to assimilate the human drawing process.

The Key problem this research tries to solve is how to simulate such an environment that closely resembles the human drawing process, not only in technique but also in the way creative decisions are made. The Agent has specific possibilities of moving and drawing on the canvas so-called actions. Having a too-large action space can lead to difficulties for the Agent to learn a policy [6], but a too-small one limits the drawing complexity. Furthermore, quantifying each action and movement a stroke is made up from means evaluating art in the process. This reward function must apply from the beginning to the final stage of a drawing.

I. State of the art

The journey of machines and algorithms in art creation dates back to the mid-20th century, exemplified by Jean Tinguely's "Machine à dessiner No. 3" [4], a precursor to today's GANs like Dalle-2 and Midjourney. The evolution into AI has significantly enhanced artistic possibilities, where the effectiveness of text prompts, including negative prompts, plays a crucial role in the outcome [8, 10]. Additionally, GANs ability to transform sketches or even brain waves into art marks a significant advancement [7, 11].

Looking at research considering Reinforcement Learning to draw humanly, the work of T. Zhou et al. (2018) [6] is interesting as they developed a model using DeepQ-Networks (DQN) that learned to scribble based on a given reference image. Similar to this problem, the agent faces a large action and observation space, making it hard for the model to converge in the training process. They were able to have positive results using DQN in combination with recorded stroke demonstrations of human artists. PaintBot[5] and a Model developed by Z. Huang et al. (2019) [4] can generate brush strokes to assimilate a given reference image even with colors. Instead, Algorithmic Muse is about creating artworks in a recursive manner where the observation of the canvas leads to the following action. With this method, the agent reacts to either its previous strokes or can collaborate with a human artist.

II. Training Environment

The custom environment is available on Github[5] and is compatible with the Gymnasium Env class[6]. The drawing mechanics work like this, for the Agent to make a stroke in this environment, every step and action creates a point next to the current point. This new point is generated with a vector length of 30 or 90 pixels in 12 radial directions. With the points' coordinates comes the decision of whether to

[1] https://www.nytimes.com/2022/09/02/technology/ai-artificial-intelligence-artists.html

[2] https://www.midjourney.com/home

[3] https://openai.com/dall-e-2

[4] https://www.tinguely.ch/en/tinguely-collection-coservation/collection.html?period=&material=&detail=795301f4-443e-44b7-94c9-c87f236acff8

[5] PageNotOnlineYet

[6] https://gymnasium.farama.org/api/env/

"draw" or to "move". If it is "draw" a line will be drawn from the current to the next point. In that same logic, strokes can be extended endlessly using bezier interpolation between all active points. A "move" action interrupts the active stroke and the brush moves without painting. This would give the environment a total of 48 actions (24 draw and 24 move). Tests in the learning process revealed reducing the move actions to only the smaller radius accelerated learning as it reduced the action space without loss in drawing abilities (see figure 1). For the study images were generated with 50, 100, 150 and 200 total actions.

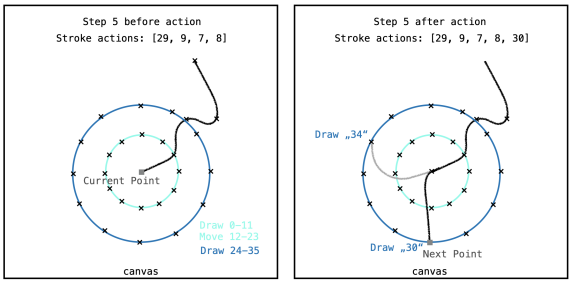


Figure 1: *Possible actions in the training environment, showing observation before and after action taken*

The observation space is a 640 x 640 pixel array in which the agent sees the current point as a rectangle of 9x9 grey pixels and the canvas with all previously drawn strokes. A frame-stacked observation of the last four steps was considered, but it did not show any improvements in the Painting Models' training.

III. Reward function

$$reward_i = \frac{(\delta_{class1} \times 20) + \delta_{class2}}{20} + coverage$$

The reward is calculated in every step after the action. It consists of a classifier giving a probability of how much it looks like a finished artistic drawing and a coverage map punishing if the canvas is not explored. For the classifier, we use the computer vision model yolov8n-cls.pt, a pre-trained image classifier trained on a custom dataset containing the following classes, Class 1: finished artwork; Class 2: snippets of artwork; Class 3: random model actions; and Class 4: bad performing painting-models (see figure 2).

Figure 2: *Custom Dataset containing following classes: Class 1: finished artwork; Class 2: snippets of artwork; Class 3: random model actions; and Class 4: bad performing painting-models*

Figure 3 shows each action's reward over the 200 steps of one drawing, and Figure 4 displays the probability of Class 1: finished artwork and Class 2: snippets of artwork. The concept is similar to a GANs architecture where a discriminator evaluates the output of a generator [3], except the classifier here is used to calculate the change in class detection from step(i-1) to step(i) (δ_{class1}). The idea is to give an appropriate value to a specific action of how much it influenced the appearance of a drawing tied to a step and observation.

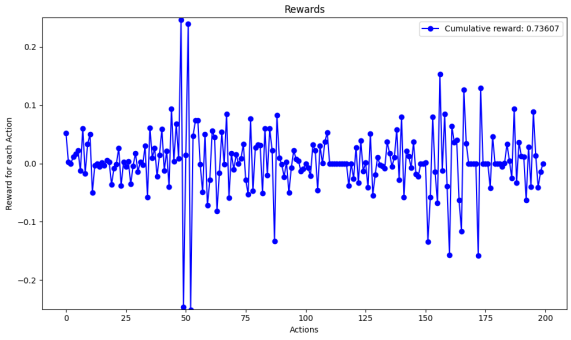


Figure 3: *Rewards of each action, the change of probability in Figure 4 prob of step(i) + prob of step(i+1) from -1.0 to 1.0*

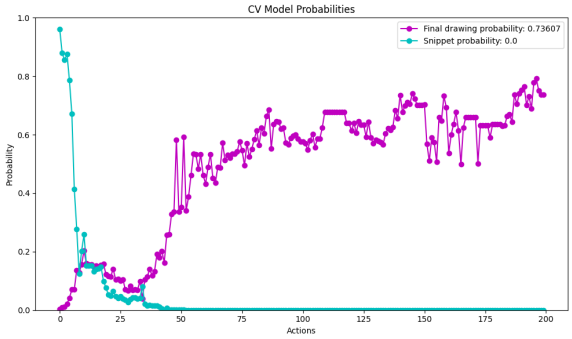


Figure 4: *Classifier calculating probability of Class 1: finished artwork (purple) and Class 2: snippets of artwork (blue) from 0.0 to 1.0*

The reward function is essential to evaluate the model's performance and primarily influences the learning process. To further emphasize the meanings of all possible final scores, the following explanation helps to understand the reward function. The final score consists of the cumulative reward in each step calculated by the reward function. The concept of score normalization Antonin Raffin, (2021) [1] was utilized and helps to evaluate a model's performance compared to other RL problems. A score of 1.0 means the model achieved its goal; in this environment, the case draws good art like a human; no higher score is achievable. A final score around 0.0 would mean the Agent either does random actions because Class3 random model actions are detected at 1.0, making Class1 = 0.0 and Class2 = 0.0. Alternatively, a bad policy, meaning Class 4, is detected at 1.0. If the final score, for example, is 0.74, it means the artwork is 74 Percent like the art from Class 1 (If we assume coverage, punishment is 0.0).

Class 1 finished artwork has a factor of 20 over Class 2 snippet of artwork. This is useful to incentivize the model at the beginning of the drawing process to reward small steps and give a hint to the right direction because the gap between random actions and a finished artwork is just too big. For a good drawing, action rewards are likely to be from 0.01 to 0.1. It's the sum of multiple actions representing several strokes that creates a finished artwork with a cumulative reward/final score of 0.8 to 1.0.

The coverage punishment is a crucial component designed to intrinsically motivate [16] the Agent during training to explore more of the canvas than just a tiny part. In the training process, it is possible to observe that the Agent learned a policy where it hits the edge or corner of the canvas and rests on the already accumulated rewards. The model can not recover from this local minimum even with a low final score of 0.02. The coverage punishment is a countermeasure that for every ten steps, the Agent needs to have explored one more of the 32 zones of the canvas—failure to do so results in a proportional punishment to the number of steps taken. The worst case would be the model always taking a "move" action, meaning the canvas is not painted, leading to a total punishment and a final score of -1.0. incentivize

IV. Optimization method(s)

I. Proximal Policy Optimization

Among RL - algorithm, PPO is favored due to its actor-critic architecture [13], where the Agent is focused on finding a policy that predicts the best action rather than only optimizing the predicted reward values for all actions as a DQN would do. As the action space consisting of 36 discrete actions is large, after 50 actions, there are 10^{77} possible actions. Other research has shown PPO to perform better than other algorithms in large action spaces by updating the weights multiple times on the same batch of collected environment samples [12]. The used implementation is the Stable-Baselines3 PPO implementation [7], with standard settings, a learning rate of 0.002, n-steps: 1024 and batch-size: 64 for a total of 400.000

[7] `https://stablebaselines3.readthedocs.io/en/master/modules/ppo.html`

timesteps.

II. Simple tree search

A tree search is also suitable for finding solutions to generate artwork in the drawing environment. This tree search randomly looks at 12 of 24 „draw“ actions and selects the one with the highest positive reward. If there are no positive rewards, with a 0.6 probability, a random „move“ action is selected, and the reward is always 0.0 because there are no changes in the observation. To prevent the tree search from getting stuck in an endless loop of ”move” actions, as all ”draw” actions lead to a negative reward, there is a 0.4 probability of a random ”draw” action being selected. Thinking about this as ”the game of drawing” holds an interesting analogy to the game of chess: The possibilities of unique drawings and unique chess games are extremely high after just a few turns. Therefore, brute forcing the problem and looking at every possible outcome isn't feasible. Alternatives like Monte Carlo Tree Search [17], and RL [15] in chess are standard practices for solving it.

V. Data

The dataset used for training the classifier (Yolo v8) Class 1, ”finished artwork,” consists of 600 sketches of human figures. Initially, we considered the huggan/wikiart dataset[8]. However, a custom dataset was created due to significant disparities between its output and the desired drawings. These custom drawings, executed by human artists using pens or small brushes, emphasize a specific style of lines that portray figurative human drawings. The raw input for this dataset was modified to ensure its adaptability to the drawing environment. It was transformed to include only black-and-white pixels, as the environment produces images exclusively with black strokes. The dataset was further augmented by adding images with four different thresholds, simulating a gradual process of strokes in the picture. This practical approach yielded 2400 training images for the Yolo v8 model, which were subsequently divided into 75% training data and 25% validation data.

Further preprocessing involved removing noise, such as points and minimal lines, simplifying the input drawings. Additionally, for Class 2, which comprises snippets of artwork, new images were generated by randomly creating boxes containing some lines. These images contained between one and five boxes (see Figure 2).

These two classes represent the criteria the Painting Model is rewarded to resemble. Lastly, a class representing random noise was created, encompassing a range of 10 to 200 actions. Another class, ”bad models,” contains the initial training iterations of the Proximal Policy Optimization (PPO) algorithm. Iteratively updating the ”bad models” class with the best version facilitates Algorithmic Muse's convergence towards the input data as it continually strives to surpass its previous iterations.

VI. Results

I. Algorithms

After training the model for 400.000 timesteps, making 80.000 drawings per run, the model stagnates at around a mean total reward of 0.18. Further testing with slight hyperparameter changes validates a stagnation around 0.1-0.2 if any learning occurs. Since RL algorithms are susceptible to hyperparameters [2] and environment settings [12] and only start to converge after a very high sample size in the millions, this research's 10 training runs of 300k timesteps is not enough data to determine whether maximum results are possible [9]. Hyperparameter tuning and more extended training could solve the issue of early stagnation.

In contrast, the results of the tree search

[8] https://huggingface.co/datasets/huggan/wikiart

show that good results are achievable with the Agent's actions and current limitations in drawing complexity. With a breadth of 12 random actions and a depth of 1 action for drawings with 50 total actions, the search gets average results of 0.22. The distribution of results varies a lot. Being able to get results from 0.8 to 1.0, there are also many results of around 0.0. The cause may be the search taking a route at the beginning of the drawing process, from which it cannot recover as it is limited to only looking at a depth of 1 branch. Subjectively good results were obtained from this approach to compare them further in the study against human artists.

II. Study

!!The study is yet not fully evaluated but will be during the next days to include 200 subjects!!

80 images generated by the tree search of Algorithmic Muse were compared in the Study[9] containing 200 subjects. Additionally, 20 images by human artists were included that resemble the stroke-based line style of Algorithmic Muse's drawing environment, and 20 images were of random actions by the Agent. The categories of the 80 AI-Images are further classified into A1, A2, N1, and N2 (A = AI and N = Random), where the first category stands for the model making drawings by continuously drawing on the canvas ("one line drawings"). The second one allows the Agent to "move" without the pen touching the canvas, resulting in multiple visible strokes. Furthermore, these categories were numbered and had an appendix of how many actions they consist of A1-XY-50, A1-XY-100, A1-XY-150, and A1-XY-200. The same logic applies to A2, N1, and N2.

In Part 1 of the study, the subjects had to choose two images out of 12 they liked the most. These 12 images were made up of 4 A1, 4 A2, 1 N1, 1 N2, and 2 human artist drawings. All 120 competed against each other in the same 10 groups.

Part 2 consisted of subjects evaluating drawings on a scale of 1 to 7. The lowest value was assigned to the word "ugly," 4 being "mid," and 7 being "beautiful." The exact half of the 120 images were being evaluated proportional to their category amount.

Part 3 was optional, and people could choose out of all 80 Algorithmic Muse images they liked the most with a color the robotic arm would paint for them as a reward for completing the study.

Questions this study might answer: What are the most liked images (win% in Part1 and evaluation Part2)? What is their favorite Algorithmic Muse style? Where AI Images can compete with human artists. Can there be associated overlapping semantic classes between AI images and Human Artist images? Images showing figurative features scored higher as people gave the feedback that they liked less abstract images. I suggest semantic categories of 1. Figurative 2. Abstract chaotic 3. Abstract with pattern, and 4. A mix of details. It would also be interesting to make hexagonal profiles of the categories for selecting categories or Quad profiles regarding semantic categories to determine whether their people responded based on a visualizable "taste."

A complete list of graphics will be added to the appendix.

VII. Where do You go from here ?

WIP: Optimize Tree search with depth and maybe MCTS or even DQN like AlphaZero. PPO algorithm further testing because it is interesting to see how far you can get this Optimization method as it decide faster than the tree search and could possibly added as a "gut" feeling enhance action selection. Human Collaboration

[9] https://www.alessandro-mac-nelly.com/umfrage

VIII. Acknowledgments

I extend my sincere gratitude to Prof. Dr. Daniel D. Hromada and Dipl. Ing. Christian Schmidts for their invaluable guidance and expertise throughout this project. Their support has been pivotal in its realization. I also wish to thank my peers in the Design Computation program for their collaborative spirit and mutual support, which have greatly enriched this experience. Their collective insight has been a cornerstone of my project's success.

References

[1] Antonin$_{R}affin$. *Rliable: Better Evaluation for Reinforcement Learning - A Visual Explanation*. Apr. 2, 2024. URL: https://araffin.github.io/post/rliable/ (visited on 04/02/2024).

[2] Theresa Eimer, Marius Lindauer, and Roberta Raileanu. *Hyperparameters in Reinforcement Learning and How To Tune Them*. 2023. arXiv: 2306.01324 [cs.LG].

[3] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[4] Zhewei Huang, Wen Heng, and Shuchang Zhou. *Learning to Paint With Model-based Deep Reinforcement Learning*. 2019. arXiv: 1903.04411 [cs.CV].

[5] Biao Jia et al. *PaintBot: A Reinforcement Learning Approach for Natural Media Painting*. 2019. arXiv: 1904.02201 [cs.CV].

[6] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[7] Bingchen Liu, Kunpeng Song, and Ahmed Elgammal. *Sketch-to-Art: Synthesizing Stylized Art Images From Sketches*. 2020. arXiv: 2002.12888 [cs.CV].

[8] Jonas Oppenlaender. "A taxonomy of prompt modifiers for text-to-image generation". In: *Behaviour amp; Information Technology* (Nov. 2023), pp. 1–14. ISSN: 1362-3001. DOI: 10.1080/0144929x.2023.2286532. URL: http://dx.doi.org/10.1080/0144929X.2023.2286532.

[9] Sohum Padhye. *Building a Reinforcement Learning Agent that can Play Rocket League*. Apr. 2, 2024. URL: https://sohum-padhye.medium.com/building-a-reinforcement-learning-agent-that-can-play-rocket-league-5df59c69b1f5 (visited on 04/02/2024).

[10] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125 [cs.CV].

[11] Ricardo Andres Diaz Rincon. *Generating Music and Generative Art from Brain activity*. 2021. arXiv: 2108.04316 [cs.HC].

[12] John Schulman. *Deep RL Bootcamp Lecture 5: Natural Policy Gradients, TRPO, PPO - lecture at Berkley*. Apr. 2, 2024. URL: https://www.youtube.com/watch?v=xvRrgxcpaHY&ab_channel=AIPrism (visited on 04/02/2024).

[13] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[14] Sakib Shahriar. *GAN Computers Generate Arts? A Survey on Visual Arts, Music, and Literary Text Generation using Generative Adversarial Network*. 2021. arXiv: 2108.03857 [cs.AI].

[15] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI].

[16] Satinder Singh et al. *Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective*. Apr. 2, 2024. URL: https://web.eecs.umich.edu/~baveja/Papers/IMRLIEEETAMDFinal.pdf (visited on 04/02/2024).

[17] Maciej Świechowski et al. "Monte Carlo Tree Search: a review of recent modifications and applications". In: *Artificial Intelligence Review* 56.3 (July 2022), pp. 2497–2562. ISSN: 1573-7462. DOI: 10.1007/s10462-022-10228-y. URL: http://dx.

doi.org/10.1007/s10462-022-10228-
y.

IX. Appendix