Generating An Understanding of Hands with Reinforcement Learning

Written by Chase Mutzig
April 29, 2024

Abstract

From its inception, AI has been able to do incredible things, from modern art, to web searching and creation, to faking an entire person into existence. However, something that has always been difficult for machines are hands, specifically of the human variety. Always too many fingers, too many segments, or no hand at all, it is incredibly rare that a hand is properly formed via a machine. In this paper, we will continue an experiment that used a variational auto encoder to generate hands by adding reinforcement learning. The goal is to generate a human hand that is anatomically accurate 70-80% of the time. To do this, we will use a dataset of around 11k hands in specific orientations and positions and feed them to the modified neural network. This paper will summarize the end results gained from using a VAE, explain how reinforcement learning is projected to help accuracy, the implementation of reinforcement learning on top of the VAE, and the results gained.

Introduction

AI mimicry in the form of vocals, photos, and videos is growing in popularity. With this growth in popularity, it is also becoming increasingly more difficult to differentiate between what is fake and real. Take a person's face as an example. In the early generations, an AI would have all the parts of the face; eyes, nose, ears, eyebrows, mouth, etc. However, these parts may have had mismatched sizes, locations, or other impossibilities (Kalpokas and Kalpokiene, 2022). In current times, these mistakes have been ironed out and it can often be very difficult to spot the malformed details. If you take a human or any humanoid figure, one of the most reliably messed up details would be the hands (Meg, 2023).

Hands are typically attached via a wrist, have a palm, and five digits. Each of these digits are made up of sections that have fingernails, fingerprints, and joints. In AI generated material, there are often incorrect numbers of such details. Currently, the main issue relates to quantity of features, rather than quality of features, as many current generators are able to make good quality and high fidelity images. In an effort to correctly generate hands 70-80% of the time, we are going to be modifying a neural network specialized in focusing on these easy-to-miss details by adding reinforcement learning. The reinforcement learning agent will work to improve the baseline understanding of the neural network in an effort to improve the results enough as to reach the 70-80% accuracy goal.

Background

To summarize the preceding experiments, a variational auto encoder is a neural network structure that forms a latent understanding of the input, then tries to recreate the input from the created understanding. In this case, the input is roughly 11,000 hands all in similar positions, and the output is a *strictly new* hand that was not previously in the input data. This newly generated hand should be visibly recognizable as a hand, and should be anatomically correct. This means that the hand should have a palm, four fingers, and a thumb. Each digit also must have the correct amount of segments, if visible.

A variational auto encoder was chosen specifically due to the nature of this task. VAEs generate probabilistic latent spaces, allowing them to generate many different samples and are thus well suited for tasks where we want to interpolate between different data; in this case, the skin tones, positions, and structure of the input hands (Pu, et. al. 2016).

A standard auto encoder has a deterministic latent space, meaning they do not capture variations in the input data well (Bank, et. al., 2023).

As for GANs, or generative adversarial networks, they implicitly learn a latent space through a generator network which may not be as understandable or controllable as a VAE latent space (El-Kaddoury, et. al., 2019).

Previous Results

The results from the VAE section will be considered the preceding and baseline results for this experiment.

The minimum goal will be to generate results better than these preceding results, and the overall goal will be to generate anatomically correct hands 70-80% of the time. To clarify, if a batch of 20 hands are created, then roughly 15 of those hands must be anatomically correct.

Figure. 1: Best baseline results from pure VAE training.

Related Work

Specifically with references to hands, there has been work mainly on reading hand motions and gestures. In one such case, Stanford University has made "[a] new AI learning scheme combined with a spray-on smart skin [that] can decipher the movements of human hands to recognize typing, sign language, and even the shape of simple familiar objects" (Patel, 2023).

In another similar project, people are using AI to learn and read gestures in an effort to make a hands-free, no external devices blackboard, similar in style to Iron-Man's holo-desk. By reading hand gestures, a person can write numbers and letters or create more blackboards, lead meetings, and many other applications (Soroni et al., 2021).

Lastly, although there are many more examples of AI using hand reading, there is a physics simulation that tries to map hand movements to a virtual environment, similar to the aforementioned project. However, this project also attempts *accurate* haptic feedback by using reinforcement and imitation learning. This means they don't want any clipping of fingers into objects, and for the person doing the manipulation to feel the opposing forces physically, from virtual objects (Garcia-Hernando, et. al., 2021).

Methodology

For this experiment, we want to change the latent space of the VAE by adjusting latent variables to improve the understanding of what a hand is, and thus improve the generated hands.

Due to its suitability, the Soft Actor-Critic (SAC) algorithm will be used. SAC is an algorithm that is well suited for continuous action spaces, which the RL agent will be dealing with since the base of this model is a VAE that generates a continuous latent space (Haarnoja, et. al., 2019). Continuing with the SAC algorithm, the agent must also have clearly defined states, actions, and rewards.

The states represent information about the environment that the agent will use to make decisions regarding any available actions it may take. Each state will be the collection of latent variables that encode information on the latent space, or understanding of the model. There will be one state per episode.

The actions will be adjustments and transformations applied to the latent variables in the VAE latent space. The agent will use SAC to learn a policy that maps the states to the actions, allowing it to modify the latent variables and hopefully improve the latent understanding of what a hand may be.

The reward for the RL agent will be based on the change in loss after each training epoch. If the loss decreases between epochs, then the agent is rewarded a positive reward in proportion to the change. If the loss instead increases, the proportional reward will be negative. The algorithm used is as follows:

$R_t = -\lambda * \Delta\ L_t$ if $t > 1$, where R_t is the reward at epoch t, $\Delta\ L_t$ is the change in loss from the previous epoch, defined as $L_t = L_t - L_{t-1}$, and λ is a positive constant scaling factor. The use of λ allows for the control of reward scaling, making rewards scale more or less depending on the loss change. λ is not strictly necessary for this algorithm to work, yet is included due to the flexibility in experimentation it provides.

SAC Algorithm

The Soft Actor-Critic algorithm has four main parts, each of which are responsible for a major portion of what makes the SAC algorithm good (Derman, et. al., 2018).

Critic Update The critic update evaluates the quality of the actions taken by the agent. This update ensures an accurate estimation of the cumulative reward, incorporating the reward, discounted future rewards, and the entropy-adjusted policy. The critic update is given as the following:

$$\mathcal{L}_Q(\phi) = E_{(s,a,r,s')\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\phi(s,a) - \right.\right.$$

$$\left.\left.\left(r + \gamma \min_{a'} Q_{\phi'}(s',a') - \alpha \log \pi_\psi(a'|s')\right)^2\right.\right. \quad (1)$$

- s: Current state
- s': Next state
- a: Taken action
- a': Next taken action
- r: Reward
- $\mathcal{D}$: Data distribution representing experiences collected from the environment
- γ: Discount factor for future rewards
- α: Temperature parameter
- $\pi_\psi(a' \mid s')$: Policy representing the probability of taking action a' in state s'

Value Function Update The value function update is responsible for estimating the cumulative reward and aligning the policy's entropy-adjusted distribution with a target entropy. This encourages a balance between trying new things (exploration) or doing something already done (exploitation), leading to diverse yet high-quality policies. The VFU is given as:

$$\mathcal{L}_V(\theta) = E_{s\sim\mathcal{D}}\left[\mathrm{KL}\left[\pi_\psi(\cdot|s)\,\middle\|\,\exp\left(\frac{Q_\phi(s,\cdot)}{\alpha}\right)\right]\right] \quad (2)$$

- $\mathcal{L}_V(\theta)$: Loss function for the value function update.
- $E_{s\sim\mathcal{D}}$: Expectation over the data distribution $\mathcal{D}$

- KL: Kullback-Leibler divergence
- $\pi_\psi(\cdot|s)$: Policy representing the probability distribution of actions given state s
- $Q_\phi(s, \cdot)$: Q-function representing the expected cumulative reward given state s and various actions
- α: Entropy temperature parameter

Policy Update The policy update updates the policy to maximize the expected reward along with the entropy term. Doing this encourages the agent to explore its environment and to take actions that yield high rewards. The policy update is given as:

$$\mathcal{L}_\pi(\psi) = E_{s\sim\mathcal{D}}\left[E_{a\sim\pi_\psi(\cdot|s)}\left[\alpha\log(\pi_\psi(a|s)) - Q_\phi(s,a)\right]\right] \tag{3}$$

- $\mathcal{L}_\pi(\psi)$: Loss function for the policy update
- $E_{s\sim\mathcal{D}}$: Expectation over the data distribution $\mathcal{D}$
- $E_{a\sim\pi_\psi(\cdot|s)}$: Expectation over the policy's action distribution given state s.
- α: Entropy temperature parameter
- $\log(\pi_\psi(a|s))$: Log probability of the action a given state s under the policy
- $Q_\phi(s, a)$: Q-function representing the expected cumulative reward given state s and action a

Temperature Entropy Update The entropy temperature update regulates exploration vs. exploitation. This update makes sure the agent maintains the wanted amount of exploration or exploitation by changing the temperature parameter based on the policy's entropy. Mathematically, the update is given as:

$$\alpha \leftarrow \text{clip}\left(\alpha + \eta * \right.$$
$$\left.\text{E}_{s\sim\mathcal{D}}\left[-\log(\pi_\psi(a|s)) - \right.\right.$$
$$\left.\left.\text{Target Entropy}, \alpha_{\text{min}}, \alpha_{\text{max}}\right. \right. \tag{4}$$

- α: Entropy temperature parameter (to be updated)
- clip$(\cdot)$: Clip function ensuring the value stays within a specified range
- η: Learning rate for the update
- $Es \sim \mathcal{D}$: Expectation over the data distribution $\mathcal{D}$
- $-\log(\pi\psi(a|s))$: Negative log probability of the action a given state s under the policy
- Target Entropy: Variable to be specified in practical application
- α_{min} and α_{max}: Minimum and maximum values for the entropy temperature

Implementation

In practical application, the SAC algorithm is not a sequence of mathematical formulas (Haarnoja, et. al., 2019) and instead uses miniature neural networks to function as actor and critic. The preceding math is used to define the theoretical application of this algorithm while in reality neural networks are used to approximate the purpose of SAC algorithmically. In the case of this paper, both the actor and critic are dense neural networks with ReLu activation applied to each dense layer allowing for non-linear learning. The input data has all RGB values normalized, meaning that the RGB scale is from zero to one, instead of 0-255. Due to this normalization, an action modifying any latent space with a Delta greater than 1 would lead to extreme changes, and thus the actor has an added dense output layer using tanh to regulate any actions to a range of [-1, 1]. The critic has an added input layer that takes the current state and actor's action and concatenates them, leading to a modified state. The critic then passes this through two dense layers, compressing it into a final single node dense layer. This process estimates the Q value of the action taken by providing a linear combination of both the input state and the action applied to it, simplified down to a single value. This single value can then be compared to other values to determine if the current state-action pair will progress or regress policy optimization. The single node output layer of the critic neural network does not have an activation function applied due to the goal being to predict a continuous Q-value rather than trying to generate a classification, in which case the non-linearity provided from activation functions would be useful. The actor's action is the prediction generated from the actor neural network given the current state.

For this project, there will be a single actor determining actions based off of training not yet implemented. There will also be two critic networks due to the use of multiple discriminators allowing for better results and decreasing the likelihood of an exploding gradient occurring. (Fujimoto, et. al., 2018). With some training, the hope is that the actor will only predict changes that are useful in policy optimization and the critics will prevent future poor decisions.

As for training the actor and critic, there will be a sample_latent function which generates a sample latent space. This sample space then gets used in making a duplicate VAE model that is nearly identical to the originally described base model. The only difference being the latent space and subsequent changes to handle a differently sized space. The environment in this case is a simple usage of the temporary model, a reset back to zero weights, tracking of actions taken and modified states, sample image generation, and giving rewards based on the loss values. Loss as stated previously is calculated using KL_reconstruction. The actor and critic networks as a whole will function as a reinforcement learning agent, and thus will be referred to as 'the agent'.

The Q-learning method for improving agents and their corresponding policies will be used, however, there is no current implementation of a training loop. There is only the described agent, temporary model, and environment. This does not mean that the agent as a whole is meaningless until it is trained, merely that it has much room for improvement.

For example, the agent is currently not basing its actions off of anything and thus any actions taken are completely random and act as adding noise to the representation. Adding noise is a common practice when preprocessing data, however, it is now being used in the model training loop, leading to a VAE that is better able to understand what is noise and what is part of the original images. There was no enhancement image preprocessing done, so even this relatively simple application of the SAC algorithm has made a large difference in results.

Issues and Solutions

When training the agent or updating a latent space, it is computationally expensive to remake a model from scratch each time if the only change is the latent space. However, this issue arises from the size of the latent space and how keras compiles models. For the first issue, when creating a model the latent space, with a dimensionality of 300, will have a shape of (0, 300). The x represents the number of active samples being understood and the y represents dimensionality. After a single epoch of training, when taking the latent space out of a model, the shape will be (num_images, 300) and thus incompatible with the original model, which was made to be compiled with a shape of (0,300). This means a new model needs to be created that is compatible with the new latent shape. Technically speaking, after the first recreation of a model, all future changes to the latent space will not make the space incompatible with the model. However, due to how keras model compiling works, of which this project uses, a models latent space cannot be 'updated'. While a model in its totality may be updated with new weights or pre-training, since the latent space is a *part* of a model and not considered weights or training, it cannot be specifically updated. To get around this issue, the 'updating' of a model is in actuality the recreation of the model using the updated latent space, leading to many models and a large use of resources.

When making changes to a latent space, taking a latent space out of a model, or putting it back in, the data type will change. When leaving or creating a model, the type will be a symbolic tensor which stores data and is not conductive to data editing. When making changes to a space or running it through the agent networks, it is necessary to have a data type that is modifiable and keeps any relevant data. To solve this, the latent space is taken via a prediction from the model *encoder*, which returns an easy to visualize and edit numpy array. After making changes to this array it is passed through a Lambda function which converts it back into a symbolic tensor and thus able to be used in a new model.

Preliminary Results

All preliminary results shown are done using a single epoch to increase the frequency at which tests can be done. Due to the following results having been generated from single epochs, the results are not expected to be any indicator of quality and instead should be seen as small-scale tests to ensure that there is improvement from the addition of an agent.

Figure. 2: VAE model with no agent, 8 training images and 2 testing images

Figure. 3: VAE model with agent, 8 training images and 2 testing images

After many quick tests using very little of the dataset, it quickly became clear that while the agent was improving the results, there was a limit to how much improvement could be gained from 10 images total. In the results gained without the agent, there are no definable features, while with the agent a vague shadow of a hand is apparent in 3/8 of the shown examples.

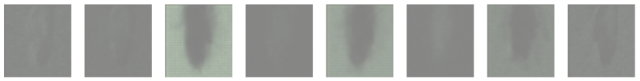

Figure. 4: VAE model with no agent, 80 training images and 20 testing images

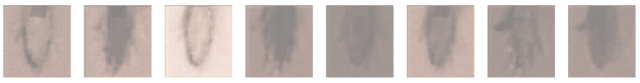

Figure. 5: VAE model with agent, 80 training images and 20 testing images

By expanding the available images by a factor of 10, the improvement from the simple and untrained agent becomes readily evident. In the example without the agent, the images still have extreme faint shadows of a hand or a green hue, while the images generated with the agent are showing progress towards a skin tone being used and zero images being mostly grey squares.

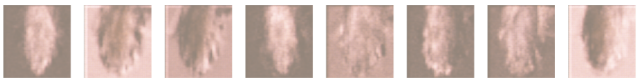

Figure. 6: VAE model with no agent, 800 training images and 200 testing images

Figure. 7: VAE model with agent, 800 training images and 200 testing images

Unfortunately, because the agent is not being trained and is simply adding in random noise, there is a turning point at which it becomes a hindrance instead of an assistance. As showcased in figures 6. and 7., there is more definition being attempted and shown in the results generated without the agent, while results with the agent almost seem to revert

back to the solid squares of color. The only notable difference between the results in favor of the agent is that the main color across all results is closer to a normal skin tone than the mostly pink hue found in the agent-less results.

Algorithm Refactoring

Due to complications from using the SAC algorithm, the agent learning and effective process was changed. The new algorithm being used is known as the Self-Correcting Q-learning algorithm, or SCQ for short. While SAC aims to maximize expected rewards, encourages exploration, is efficient even with few examples, is known for its stable and fast convergence, and is specifically suited for continuous action spaces, the implementation of SAC is extremely complex and the policy it generates is typically of a Gaussian distribution over actions (Chien and Yang, 2022). For this project specifically, a policy as such would be less beneficial than the type of policy a Q-learning algorithm produces which represents the expected cumulative reward gained based on an action and the following state. Therefore, SCQ will be used because it is a Q-learning algorithm that aims to learn the optimal action-value function by lowering the difference between the generated and expected outputs. SCQ is also capable of exploration, when given sufficient examples it is able to generate high-quality policies, is able to converge to variable policies instead of a single distribution, and it is able to be applied to both discrete and continuous action spaces (Kumar, et. al., 2020).

Self-Correcting Q-learning uses two tables two store values representing actions, the rewards, and many other factors if necessary. When updating the values, instead of using a soft-actor critic form, the tables are updated based on the error found between the generated and expected outputs. This two table approach is standard for double q-learning, and SCQ adds onto this update a correction variable to prevent over and under fitting. The effect of hyper-parameters and variables are the same, as well as the contest of exploration vs. exploitation, and therefore will not be reiterated. The correcting modification leads to a more stable and reliable training process (Zhang, et. al., 2021). With the correcting variable added, it is better for this agent if the environment it works in and the states it sees are constantly changing, and therefore the process of training the agent has also changed.

New Process

The changing of algorithm as well as the limitations with the previous algorithm have necessitated a change to the overall training process. The previous method was to train the network a little, edit the latent representation with the agent, and build a new temporary network with the new understanding, and repeat the loop. This was time and resource intensive as well as relying on the network to form connections. This dependency will lead to the same problem encountered in the past with the network reaching its capacity to learn and understand, ultimately leading to the same quality of results.

The new and current method is to train the neural network to capacity one time. Then, with a prediction from the model's encoder to obtain the latent representation, the agent is given its environment. The representation is changed from a symbolic tensor to a numpy array so changes can be made and then converted back, similar to the previous method. The change occurs after this step, where the latent representation is passed to the network's decoder to generate images. By making the agent be the step in between encoder and decoder, the process is no longer fully reliant on the network's understanding of the latent space to improve and instead relies on the agent to generate an optimal policy.

Now, before the representation is given to the decoder, the agent must act upon it. Since the latent representation is in the form of a 300 dimension array, the agent is capable of affecting every single index within the limits of the action space. For this project specifically, the action space is defined as discrete values between -1 and 1, inclusively. The limits of -1 and 1 are so that any single change is small, as all modifications directly change the latent representation that the decoder must make into an image. The amount of discrete values between -1 and 1 is effectively infinite, which would be much too large of an action space. Therefore, the action space is defined with a granularity variable that determines how many even divisions there should be. By gradually increasing granularity, the action space grows and the difference between each unique action decreases. Eventually an optimal granularity of 10,001 was chosen. A granularity of 10,001 means there are 5,000 negative options, 5,000 positive options, and the singular option to do nothing if an action of '0' is taken. This also means there is a difference of 0.0002 between each action, making anything above 10,001 granularity extremely small and not worth the extra resources and time it would take.

The modified latent representation is then sent to the decoder, where its output is then compared to a random sample taken from the testing split of the dataset. Both images are then flattened to be single dimension vectors and directly compared using mean squared error to obtain an error value. The closer the value is to zero, the closer the output and dataset resemble each other. Due to this, the inverse of the error is the reward, making the reward increase as the error decreases. A direct comparison as described will not actively seek out nuances such as where shadows are applied, however, it will be a good metric for the agent to learn from. After a reward is determined, one of the two Q-tables is randomly chosen to be updated. The update is based on the action taken and the degree of the reward. The change decided uses the values from the other q-table, summed with the correction variable, and then applied to the chosen q-table. The main benefit of splitting updates across two q-tables is that it decreases the likely-hood of over and under estimating if the agent explored too much or too little, leading to a much more stable learning.

Once the q-table is updated, the encoder makes another prediction on the VAE latent representation, which has not been changed or updated, and replaces the current state that the agent is looking at. Since the VAE latent space is continuous, every prediction the encoder makes on it will not be identical, effectively setting the agent back to the start of the training process. By constantly changing the representation

and thus the actions needed to make the output resemble the dataset, the agent must also constantly change the values and importance of which actions to take. This leads to the agent never over generalizing and has lead to vastly improved outcomes.

Refactored Output

Before continuing, one may ask why the agent did not build upon the initial representation predicted by the network's encoder. The representation is changed every episode, resulting in the initial representation eventually having been manipulated excessively and having a high contrast, saturation, sharpness, and low accuracy.

Figure. 8: Decoded Image with Continuous Processing on Initial Representation

By using the constant change, new method, and variable tuning, the following was achieved with a relatively average amount of 1,000 episodes. For comparison, the left half of the 20 batch output will show the output from decoding the neural network, and the right half will show the output from decoding the agent after both have been trained. The training and testing data for the neural network will always be in an 80/20 split. Shorthand notation of resources allocated will be as follows: A (T) for total amount of data, a (E) for epoch amount, and a (P) for episode count.

Figure. 9: Initial Output Comparison of Network and Agent. T:1000, E:20, P:1000

The improvement shown in Figure 9 is vast, with the network only receiving 20 epochs of training. The above took the agent about 2 hours to complete. To show that the agent does not only converge onto one hand, a test that gave the network significantly more epochs is shown in Figure 10.

After a few more tests, it was determined 1000 epochs was not beneficial enough to warrant the extra time required, so future tests had a range of 100-400 epochs. A test was conducted where the agent was given 10,000 episodes, however, with each consecutive episode the prediction from the encoder took longer to generate, leading the program to automatically exit after 72 hours of run time. Due to this mistake, early stopping was implemented where the agent would end training early if the error did not improve a certain amount after a specified episode count.

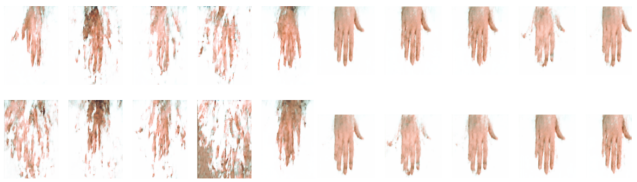

Figure. 10: Output Comparison of Network and Agent Given More Epochs. T:1000, E:1000, P:1000

Figure. 11: Output Comparison of Network and Agent. T:1000, E:400 stopped at 358, P:3000 stopped at 13

During testing it was noted that the agent was converging to a single hand as shown in Figures 10 and 11, even when the amount of data was fully increased to 11,076. After looking into the problem, it was determined that the agent tended to converge onto a specific hand in the dataset, typically the 3rd or fourth. After more looking, this problem only occurred while running the code on a CPU, or Central Processing Unit. This is the most common type of computation resource and is very deterministic (Yıldız, et. al., 2018). When running the code instead on Google Colab's GPU, or Graphical Processing Unit, the agent was found to converge to different hands. This phenomena was found to be caused by the extra variable of 'GPURNG'. GPUs are primarily used for graphics rendering, however, they can also be used for parallel computing tasks and random number generation (Brodtkorb, et. al., 2012). By running the code on a GPU, the parallel processing made the output non-deterministic and thus variable. This variation is why the initial results in Figure 9, in which the code was able to be run on a GPU, are different from most other test runs.

Before learning about the GPURNG variable, some tests were done in an attempt to force a different hand, and also to see how the agent handled darker skintones. The dataset was decreased from 11,076 images of mixed skintone hands down to 659 dark skintone hands. Due to the significant reduction in available data, the generated hands are blurry and not of as high quality, however, the results do show that the agent has no issue dealing with a large range of human skintones. Upon inspection, the quality is still at an acceptable level with each hand still having correct anatomy and coloration, albeit a higher amount of visual distortion.

Figure. 12: Output Comparison of Network and Agent on Darker Skintones. T:659, E:300, P:1000 stopped at 147

Evaluation

Aside from being anatomically correct, the hand must be the correct color (within any reasonable *human* skin tone range), have the correct orientation, be a clear image with little blur, etc. Essentially, the evaluation will be on a visual basis of whether the output is closer to what a hand is than the previous output. Out of the roughly 20 images per output, about 15 of them need to be highly accurate for this project to be deemed a success and reach the 70-80% benchmark.

When viewing the latest results from the new method of reinforcement learning agent, it is clear that the hands are of a much higher quality than in previous tests. Observe the hands generated by the agent in Figure 11. Each hand has a palm, thumb, and four fingers. Upon closer inspection, each hand also has slight discoloration where knuckles, shadows, and veins would appear. These hands are mostly in the same position with the same skin tone, and all exhibit the added detail of fingernails. While the fingers on each hand could be considered abnormally long, the output is anatomically correct. The color of the hand certainly falls within the human skin tone range, and any visual distortions are small and not obstructive to the overall structure of the hand. When viewing a full collection of 20 hands, ĩ7 are without harsh distortion, with the rest having distortions ranging from mild to severe. Since the results match the evaluation criteria, and are proven to be repeatable, this experiment will be marked a success.

Conclusion

In this paper we talked about the continuation of a neural network experimentation that tried to generate images of hands and how we will work to improve it using reinforcement learning. This previous experimentation used a variational auto encoder and got poor results in color. The final agent applied utilized the self-correcting q-learning algorithm in a practical environment. The agent was able to improve the latent understanding and output generation of the base VAE model to the point of achieving the goal of 70̃-80% accurate and visually acceptable results.

References

Matthias, Meg. "Why does AI art screw up hands and fingers?". Encyclopedia Britannica, 25 Aug. 2023

Kalpokas, Ignas, and Julija Kalpokiene. "From Gans to Deepfakes: Getting the Characteristics Right." SpringerLink, Springer International Publishing, 2022

Prachi Patel. "Spray-on Smart Skin Reads Typing and Hand Gestures." IEEE Spectrum, IEEE Spectrum, 3 Mar. 2023

F. Soroni, S. a. Sajid, M. N. H. Bhuiyan, J. Iqbal and M. M. Khan, Hand Gesture Based Virtual Blackboard Using Webcam, 06 Dec. 2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 2021

Guillermo Garcia-Hernando, Edward Johns, & Tae-Kyun Kim, Physics-Based Dexterous Manipulations with Estimated Hand Poses and Residual Reinforcement Learning. IEEE Spectrum, IEEE Spectrum, 2021

Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., & Carin, L. (n.d.). Variational Autoencoder for Deep Learning of Images, Labels, and Captions, NeurIPS Proceedings, 2016

Bank, Dor, Noam Koenigstein, and Raja Giryes. Autoencoders. Machine learning for data science handbook: data mining and knowledge discovery handbook Feb. 2023

El-Kaddoury, M., Mahmoudi, A., Himmi, M.M. "Deep Generative Models... and Generative Adversarial Networks", Spinger, 2019.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic Algorithms and Applications". Cornell University arXiv, 2019

Esther Derman, Daniel J. Mankowitz, Timothy A. Mann, and Shie Mannor. "Soft-Robust Actor-Critic Policy Gradient". Cornell University arXiv, 2018

Scott Fujimoto, Herke van Hoof, David Meger, "Addressing Function Approxiamtion Error in Actor-Critic Methods", Proceedings of Machine Learning Reasearch, 22 Oct. 2018

Jen-Tzung Chien and Shu-Hsiang Yang. "Model-Based Soft Actor-Critic". IEEE, 3 Feb. 2022

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. "Conservative Q-Learning for Offline Reinforcement Learning". NeurIPS Proceedings, 2020

Lieping Zhang, Liu Tang, Shenglan Zhang, Zhengzhong Wang, Xianhao Shen, and Zuqiong Zhang. "A Self-Adaptive Reinforcement-Exploration Q-Learning Algorithm". Symmetry, 11 June. 2021

Abdullah Yıldız, H. Fatih Ugurdag, Barış Aktemur,

Deniz İskender, and Sezer Gören. ”CPU design simplified”. IEEE, 9 Dec. 2018

André R. Brodtkorb, Trond R. Hagen, and Martin L. Sætra. ”Graphics processing unit (GPU) programming strategies and trends in GPU computing”. ELSEVIER, 16 Apr. 2012