A Realistic, Reinforcement Learning Painter

Peter Schaldenbrand
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
`pschalde@andrew.cmu.edu`

Jean Oh
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
`jeanoh@nrec.ri.cmu.edu`

Abstract

Prior work in converting images into an ordered sequence of brush stroke instructions has produced incredibly accurate results in computer simulated settings, however, these simulations do not translate to real world painting. We modified the existing state-of-the-art reinforcement learning model[5] in order to make it more realistic – as in the output brush stroke instructions can be accurately performed by a painting agent, and the instruction sequence resembles the method that a human painter would plan. We made the individual brush strokes more realistic by adding constraints to the RL model's Neural Renderer such as a maximum brush stroke length and a fixed width brush size. The stroke sequence produced from the model was made more realistic by using Content Masked Reward whereby the model was incentivized to paint regions of the canvas that would make the target object recognizable. Our model generates output that when fed to a robot painting agent generates paintings that look like the target image, while prior work does not. Additionally, our model generates a sequence of strokes that look more like a human produced them than related work without the use of human produced data for training.

1 Introduction

Our research goal was to create a realistic model for translating images into a sequence of brush strokes. We define **realistic** in this usage as the feasibility with which the brush strokes can be performed by a painting agent as well as the sequence of the brush strokes resembling that of a human painter.

Prior work in converting images into brush strokes has been very successful in accuracy, especially in using Reinforcement Learning. Most notably, Huang *et al.* [5] uses a RL model and given enough paint strokes, can almost perfectly reproduce the given image as seen in Figure 1. Inspecting the intermediate strokes in Figure 1, many of the strokes could not be realistically painted by an agent. For instance, there is no constraint on brush size, and the paint is not necessarily opaque. These are two limitations that would impede a painting agent that utilized the stroke output of the Huang *et al.* [5] model.

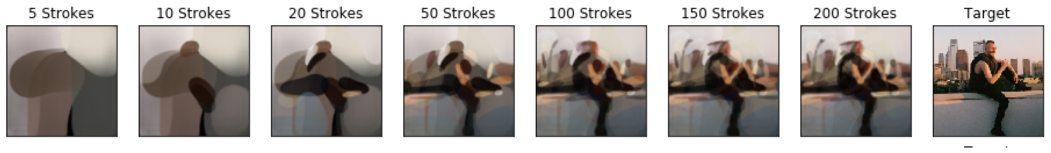

Figure 1: Series of strokes from the Huang *et al.* [5] model.

Constraint	Reason	Implementation Details
Opaque Strokes	The Huang *et al.* [5] model allows for any amount of opaqueness, however, acrylic paint generally masks whatever content is beneath it.	We fixed the opaqueness of each stroke to be completely solid.
Maximum Stroke Length	A brush can only hold so much paint, and in the duration of a stroke, the brush will run out of paint. The Huang *et al.* model's stroke length limit is roughly the size of the canvas.	We set a stroke length limit to be one third of the canvas width.
Canvas Color	The Huang *et al.* model assumes that the blank, initial canvas is black.	White was assumed to be the starting canvas color.
Fixed Brush Width	The Huang *et al.* model allows for brush widths up to the size of the canvas and any granularity smaller. Realistically, the painting agent would only have a finite amount of fixed width brushes.	We assume that the brush width is 5% of the canvas width. This can be made to be smaller by running the model on a subsection of the image.
Discrete Paint Colors	The Huang *et al.* model allows for any colored paint. Our painting agent uses a discrete number of paints.	We use K-Means clustering on the target image to determine which paint colors to use, then discretize the output brush strokes using a color similarity algorithm. This does not affect the training of the algorithm.

Table 1: Constraints added to the [5] World Model to make the Neural Renderer behave closer to real painting.

The Huang *et al.* [5] model uses a Neural Renderer to simulate the painting environment. This makes training much faster than waiting for a separate computer program to renderer a brush stroke onto a digital canvas. The Neural Renderer is differentiable, which also improves training speed. The Neural Renderer is an example of using a World Model [3] in Reinforcement Learning. This world model is extremely effective for training the model quickly, however, the world model in this case is vastly different than the real world. In this paper, we alter the Neural Renderer in the Huang *et al.* [5] model to behave more similarly to how real painting with brushes and acrylic paint work. Table 1 outlines the constraints added to the Neural Renderer.

In order to test whether the constraints we added made the painting simulation more realistic, we used an Arduino Braccio[1] robot arm as a painting agent. The robot can take the instructions produced from the painting model and paint them onto a small canvas with rough accuracy. The robot uses a discrete number of pre-mixed paints and can paint on a canvas up to 20cm$\times$20cm.

The constraints in Table 1 affect the individual strokes for the model. We also wanted to make the collective stroke sequence output to be closer to realistic painting methods. As a human painter works, the object that they are painting is apparent early on in the process. Often the painter will sketch the subject of the painting with a pencil first or they will do a very rough preliminary painting that lacks detail. We introduce a concept we call Content Masked Loss to give the model more reward and incentive to paint regions of the painting that are important for recognizing the image. The VGG-16 [11] convolutional neural network model that was pre-trained for object recognition on the ImageNet Large Scale Visual Recognition Challenge [9] dataset was used to extract features from the target image. The extracted features were used to weight the regions of the image that were important for object recognition in the model's reward function. The model therefore spent more and earlier strokes painting objects like mouths, eyes, and wheels instead of background, pedestrian objects such as sky or walls. The model painted the important regions first, as they would give the model the most reward.

[1]`https://store.arduino.cc/usa/tinkerkit-braccio`

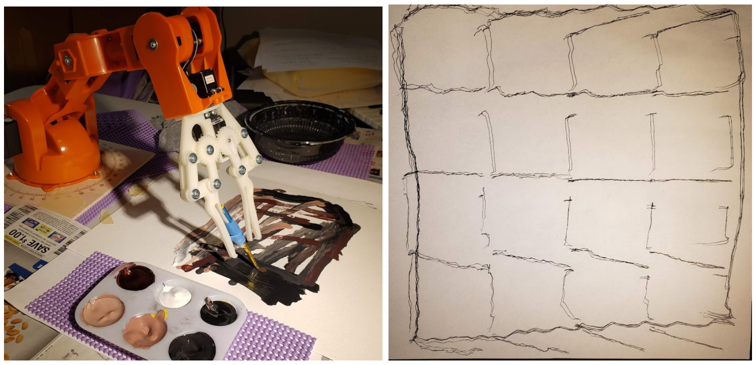

Figure 2: (Left) The Arduino Braccio used to paint the strokes produced from the painting models. (Right) A grid that the robot was instructed to draw with a pen attached to it to show the robot's inaccuracies.

2 Related Work

Stroke-based Rendering (SBR) is the process of layering individual elements, such as shapes or brush strokes, onto a digital canvas in order to reproduce a given image. The goal of SBR may be to perfectly recreate an image with a given set of tools, or it may be to add creative abstractions while rendering.

Recent methods attempt to tackle SBR using deep learning [5, 1, 8, 13], though the field has existed long before these tools became popular [4]. "Artist Agent" [12] was an approach to modelling brush strokes on paper using a reinforcement learning agent. The agent designed brush strokes and received feedback as to whether the strokes were believably human or not.

Other SBR work attempts to create an agent that can produce whole paintings rather than single brush strokes. SPIRAL [1] used reinforcement learning to generate brush strokes that would render into a painting that minimized the difference between the canvas and the given image. In SPIRAL, the agent would predict a stroke, a renderer would add this stroke to the canvas, and then the loss could be computed. The rendering in this manor is slow, and it is not differentiable. So, it is inefficient for the algorithm to optimize the parameters in the SPIRAL model.

Work that builds off of SPIRAL [8, 13, 5] often uses the world model concept [3] in order to speed up training. In the case of painting, the world model is called a Neural Renderer. The Neural Renderer is implemented with a differentiable neural network so that the loss can be back-propagated through it. Neural Painter [8] builds off of SPIRAL using the world model concept as well as uses a Variational Autoencoder and a Generative Adversarial Network to render the brush strokes into an image. Using a VAE or GAN allows the model to learn the distribution of brush strokes which enables the authors to sample paintings from a category's distribution. The Neural Painter model is not nearly as accurate at recreating an image using brush strokes as Huang *et al.*, an RL approach, though.

StrokeNet [13] uses the world model, but generates strokes using a Recurrent Neural Network rather than reinforcement learning. StrokeNet is successful at decomposing MNIST digits into strokes but fails to generalize to full, complex images.

Huang *et al.* [5] achieves the best results at decomposing an image into a series of strokes. The authors use the SPIRAL architecture with the world concept model. Additionally, they use a continuous action space which allows for more flexibility in their model. The objective of the reinforcement learning algorithm is to minimize the difference between the generated painting and the given painting. The authors find that using WGAN loss known as *Wasserstein-l* distance, or *Earth-Mover* distance, generates better results than the L_2 distance. The results from this paper are very good, for with enough strokes, the model is able to almost completely recreate the input image.

It may seem intuitive to train the model to paint using painting data, but there aren't many data sets containing images and their corresponding strokes that could be used to train a model in that supervised fashion. The data sets that do exist with this information are often not very practical. For instance, the `QuickDraw` [2] dataset, which contains cartoon sketches that users create while trying to

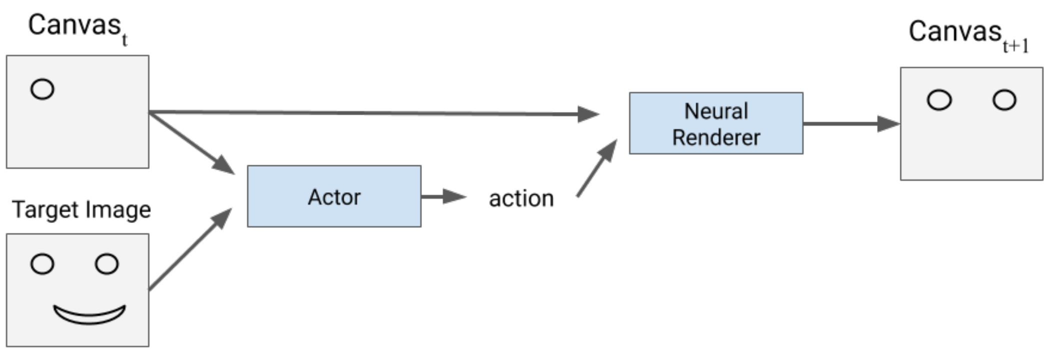

Figure 3: The Neural Renderer is used to model the application of a brush stroke onto a canvas

render a given image or noun, contain both the brush strokes and image data, however, these sketches are too simple for learning real painting. SketchRNN[2] does use paired data, but it can only draw objects that have been labelled in its training data. Because of the lack of quality data sets with paired examples, there is a great need to use unsupervised methods to train the model.

Reinforcement Learning models such as Huang *et al.* [5] and SPIRAL[1] perform SBR in an unsupervised fashion. The model learns through trial and error how to paint images. The reward functions are designed to maximize the difference in loss between the current canvas and target image and the canvas with an additional stroke and target image. These methods are capable of recreating the target image very accurately. Because theses models learn in an unsupervised fashion, they do not encode how a human painter would paint. Additionally, the strokes allowed by models such as Huang *et al.* [5] are unrealistic to what is possible to paint using a standard brush. In this paper we introduce the first Reinforcement Learning method of Stroke-Based Rendering that is capable of producing instructions that can easily be fed to a painting agent to be painted accurately.

3 Methods

3.1 The Model

We employ the model used in Huang *et al.* [5] which uses a deep reinforcement learning framework. The action space describes brush strokes in the form of Bezier curves. There are values for three position coordinates, thickness of the brush stroke at both ends, opacity of the paint at both ends, and color. As the action space is continuous, Huang *et al.* adopts the Deep Deterministic Policy Gradient (DDPG) [6]. A Neural Renderer is used to model the application of a brush stroke onto a canvas. The Neural Renderer is differentiable neural network and can be seen in Figure 3.

The reward function is calculated by computing the difference the the loss at time t and $t+1$ as seen in Equation 1. Huang *et al.* uses an adversarial model to compute the loss function (Equation 2). The model based DDPG critic predicts an expected reward function:

$$V(c_t) = r(c_t, a_t) + \gamma V(c_{t+1})$$

where $r(c_t, a_t)$ is the reward of performing action a_t on canvas c_t. c_{t+1} can be renderered using the Neural Renderer to apply the brush stroke a_t to the canvas c_{t+1}, seen below as $trans(c_t, a_t)$. The actor in the DDPG, $\pi(c_t)$, is trained to maximize the expected reward:

$$r(c_t, \pi(c_t)) + V(trans(c_t, \pi(c_t)))$$

3.2 Methods: Constraints

We updated the World Model (Neural Renderer) of the painter model to constrain the individual strokes to be more realistic. The process for adapting the individual strokes is outlined in Table 1. Using these constraints, the Neural Renderer is forced to behave more closely to real painting. The Neural Renderer is trained independently, and learns to paint a stroke onto a canvas. In Huang *et al.*, the strokes were unconstrained Bezier curves. When we trained the Neural Renderer, the strokes were limited as shown in Table 1.

$$Reward = mean\left[L(c_t, y) - L(c_{t+1}, y)\right] \quad (1)$$
$$L_{GAN}(c, y) = 1 - Discriminator(c, y) \quad (2)$$
$$L_{L2}(c, y) = (c - y)^2 \quad (3)$$
$$L_{CL}(c, y) = (VGG_l(c) - VGG_l(y))^2 \quad (4)$$
$$L_{CM}(c, y) = (c - y)^2 * normalize(VGG_l(y) \quad (5)$$
$$L_{L1^*}(c, y) = min(|c - y|, \lambda) \quad (6)$$
$$L_{CM+L1^*}(c, y) = min(|c - y|, \lambda) * normalize(VGG_l(y)) \quad (7)$$

Figure 4: The reward function is the per-pixel average of the loss between the canvas at time t and the canvas with an additional stroke at time $t + 1$. The various loss functions used in this paper follow, where c is the canvas and y is the target image. The $Discrimanator$ is a model used to differentiate paintings of an image and the actual image.

3.3 Methods: Reward Functions

The stroke planning was affected by manipulating the loss function used in the reward function of the model. The Huang *et al.* [5] model used the adversarial loss as seen in Equation 2. The $Discriminator$ model outputs 1 if the canvas and target image are the same and 0 if they are completely different. This loss function is agnostic to the abstract content in the image and only computes the pixel difference between the target image and the current canvas.

Painters take an abstract approach when painting a given image and consider the subject of the image. As a human painter paints, an onlooker should be able to recognize the image they are painting early on in the process. We took two different approaches to increasing the recognizability of the painting strokes: Content Loss Reward and Content Masked Reward.

3.3.1 Content Loss Reward

In Content Loss (Equation 4), we used the $L2$ loss on the features extracted from the canvas and target images instead of the images themselves. We used the VGG-16 [11] model to extract the features. The VGG-16 parameters are fixed. Using content loss in the reward function, the model generates brush strokes that minimize the loss between the features from the target and the features from the current canvas instead of directly using the target and canvas.

3.3.2 Content Masked Loss Reward

The intuition behind Content Masked Loss is to increase the weight of the per-pixel loss in "important" regions of the canvas. The "importance" of a region is determined by extracting features using the VGG-16 [11] model. As seen in Figure 5, the Content Masked Loss is equal to the weighted L2 distance between a canvas and the target image. The features are computed by feeding the target image into the VGG-16 model, taking the output of an intermediate layer, averaging the image across the channels, and then 0-1 normalizing it. We experimented using the output from different layers in the VGG-16 model in Appendix Figures 13 and 14.

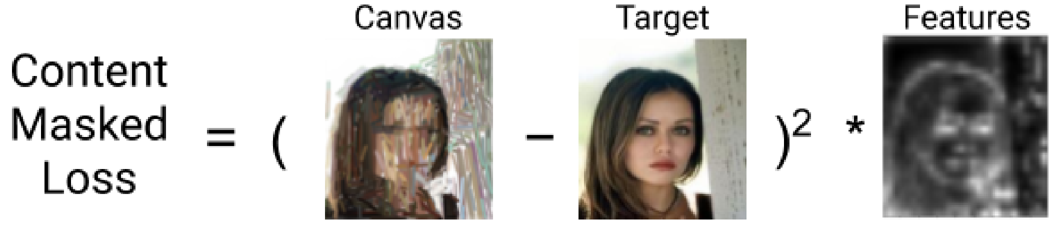

Figure 5: Content Masked Loss is the L2 distance between a canvas and the target image weighted by the features extracted from the target image.

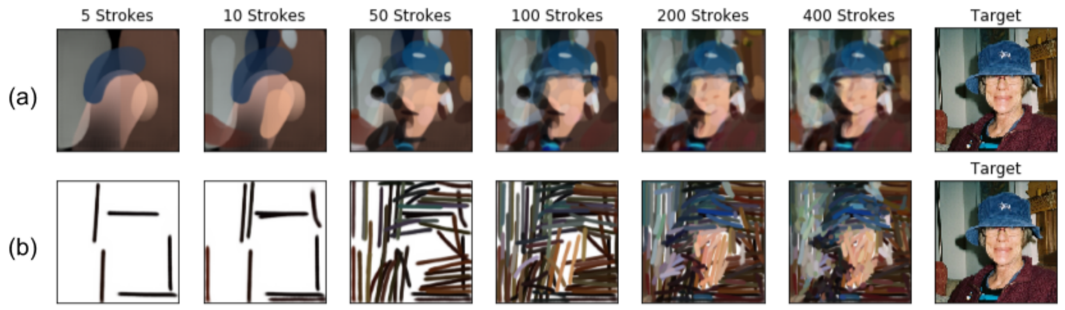

Figure 6: The effects of the constraints from Table 1 (except the color constraint). Row (a) is the Huang *et al.* [5] model at various points in the painting process while row (b) is the same model using the constrained Neural Renderer.

3.3.3 L1 with Loss Clipping

The reward functions all rely on measures of distance between the canvas and the target image. Since the images are in RGB format, the largest distance would be between a white and black pixel. The canvas is initially white, so the model will have a bias towards painting darker colors since it can get more reward with them. We reduce this bias by experimenting with $L1$ instead of $L2$ losses. We also set a constant maximum value for the per-pixel loss. Loss Clipping used in a loss function can be seen in Figure 4 with $L1$ ($L1*$) and in conjunction with Content Masked Loss ($CM + L1*$).

3.4 Methods: Robot Arm Use Case

We programmed an Arduino Braccio robot arm to paint the strokes generated by the model. The Braccio is a low-end robot and therefore has some inherit inaccuracies. We attached a pen to the robot and instructed it to paint a grid. In Figure 2, the inaccuracies of the robot can be seen as the robot struggles on the edges of the canvas.

The robot painted strokes generated by the Huang *et al.* [5] model and our models to test if our modifications actually made the brush stroke sequence more realistic.

4 Results

4.1 Results: Constraints

The effects of all the constraints in Table 1 except for the color constraint are in Figure 6. The constrained model is able to reproduce the target image but not with as high fidelity as the Huang *et al.* [5] model. The strokes are now fixed width and considerably smaller than what the Huang *et al.* [5] model allows, so it takes many more strokes to reproduce the target image in the constrained model. The fixed width strokes are too big to make out details such as mouth and eyes. The Huang *et al.* [5] paper details a method to get finer details of images: the canvas is divided up into subsections, then the model is run over each subsection. So if we divide an image into quadrants and run the constrained model over each, this is equivalent to painting with a brush half the width of fixed width constraint.

The last canvases in Figure 6 show a significant difference in fidelity between the constrained model and the Huang *et al.* [5] model. One of the most impressive aspects of the Huang *et al.* [5] model was that it could nearly replicate an image given enough strokes. We tested whether the constrained model could also achieve this accuracy. In Figure 7, the models generated 125 strokes using the entire image as input, then 125 strokes for each non-overlapping 1/25th of the image. The constrained model is still noticeably less accurate than the Huang *et al.* [5] model at recreating the target image, however, the constrained model's output looks more like a real painting and is still very accurate.

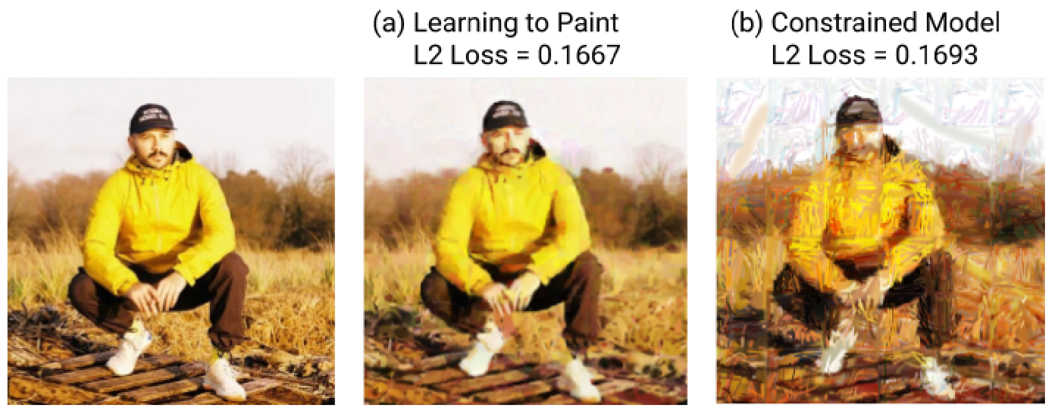

Figure 7: On the left is the target image. The paintings were created by the model generating 125 strokes looking at the whole canvas and then 125 strokes in each of 25 evenly distributed subsections of the canvas. (a) was created using the Huang *et al.* [5] model as is while (b) used the constraints in table 1.

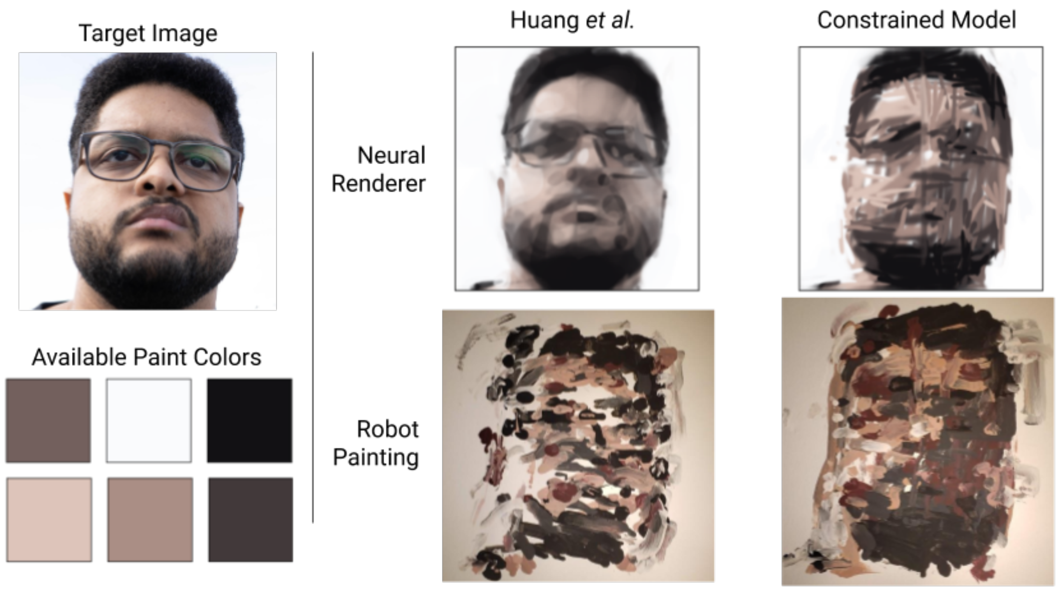

Figure 8: Paintings made from the strokes of the Huang *et al.* [5] and that same model with the constraints in Table 1. The six colors of paint displayed were all that was available and were determined using K-means clustering on the target image. The model was run over the whole target image to produce 150 strokes which were painted with a 1cm brush. Then, the model was run in each quadrant producing 75 strokes in each which were painted with a 0.5cm brush. There were 450 total brush strokes on the 20cm×20cm canvas.

4.2 Results: Robot Painter

A robot painting agent was used to test whether the constraints in Table 1 made the model more realistic in terms of individual strokes. An Arduino Braccio robot arm was used to paint strokes produced by both the Huang *et al.* [5] as is and that model with the constraints. Figure 8 shows the neural renderer for each model's interpretation of the strokes generated by the model and the robot painter's rendering of the strokes. Neither robot painting is particularly true to the target image, however, the Constrained Model's painting looks more complete than that of Huang *et al.*. Comparing the robot painting to the neural rendering: The Constrained Model's painting looks far more similar to its neural rendering than Huang *et al.*.

4.3 Results: Content Loss Reward

Using the the $L2$ distances of the features extracted from the canvas and target image in a reward function is far more difficult for a model to learn than the image and canvas themselves. The gradient when comparing the difference of features extracted is highly abstract and too difficult for the model to navigate. We tried to use this content loss (Equation 4) reward function, but the model would only go to a local minimum and produce blank canvases no matter what parameters we used. We tried to combine the Content Loss Reward with the L2 Reward and weight them in different ways, but the model still could not converge.

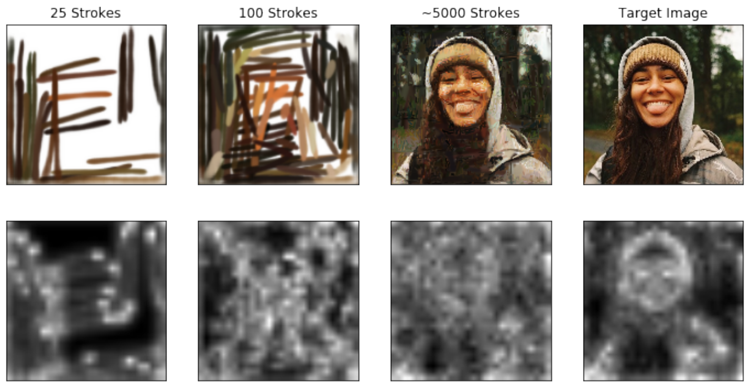

Figure 9: The top left three images are output from the model painting the target image in the top right. The second row are the features extracted from their corresponding image above them using the first 17 layers of the VGG-16 model.

The feature extraction process does not translate well to the paintings. The VGG-16 model [11] was trained on photographs. The canvases (especially with only a few strokes) are so vastly different from photographs that the extracted features are relatively random. As seen in Figure 9, even when the painting is extremely similar to the target image, the features extracted are very different.

4.4 Results: Content Masked Reward

The VGG-16 model has 31 layers in the feature extraction portion of the architecture. To compute the feature mask, the output of a layer is averaged across channels and 0-1 normalized. The output of each of the first 29 layers of the pretrained VGG-16 model are shown in Figure 13. The initial layers extract features such as edges and texture, while the later layers extract higher level information such as eyes and mouths.

We trained the constrained model using the Content Masked Loss (Equation 5) in the reward function using various layers from the VGG-16 model. After the models were trained, the painting process for each model was tested in Figure 14. The lower level layers produced models that focused more broadly on the subjects face while higher layers were focused on abstract features such as eyes. All models performed very similar in their final results, however, the intermediate strokes were strongly affected. The models that used higher layers performed more human-like since they focused on the high level features of the image.

Layer 17 was chosen for future experiments since it was a good balance between using high and low level features.

4.5 Results: Human-Like Painting

Part of making the painting model more realistic was to make the model paint more like a human would. The reward function was altered to incentivize the model to paint in the process a human would. As a human painter paints, the subject of the painting should be apparent. The intermediate canvases of models trained with different loss functions can be seen in Figure 10.

In order to test if the stroke sequence produced from our model created images in a human-like process, we used a facial detection model to determine how early in the painting process each model

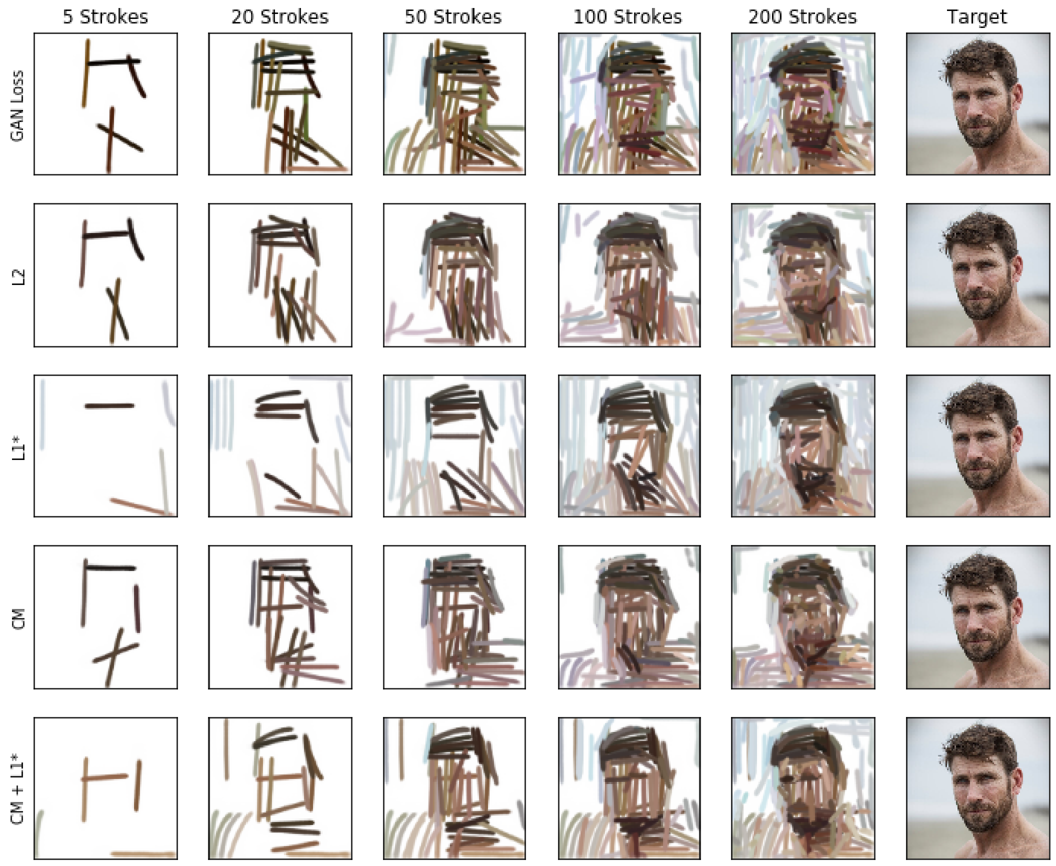

Figure 10: The painting process for models trained using different loss functions in their reward functions.

was capable of producing distinguishable faces. Faces were chosen for this experiment, since the models were trained on the CelebA[7] dataset. The model that paints the most human-like would paint recognizable facial features earliest in the process rather than the canvas looking random until the end. We used FaceNet[10] as a face detection model.

We tested five models each varying with only the loss function used to compute the reward. All models use the constraints detailed in Table 1. The five models:

- GAN - Adversarial Loss. Equivalent to Huang *et al.* [5] but with the constraints in Table 1. Equation 2
- L2 - $L2$ Loss. Equation 3
- L1* - $L1$ with Clipping Loss. Equation 6
- CM - Content Masked Loss. Equation 5
- CM + L1* - Content Masked, $L1$ with Clipping Loss. Equation 7

In the test, we ran each painting algorithm on an image until the face detection model detected a face with greater than 50% certainty. We then recorded the number of strokes it took to trigger the detection. We tested on 2000 held out images from the CelebA[7] dataset and set a maximum number of strokes to be 750. After 750 strokes, there is very little chance for facial detection. The average number of strokes necessary for each of the models to produce a detected face and the percentage of times that a face was not detected within 750 strokes is found in Table 2. Histograms comparing the number of strokes for face detections are found in Figure 11. The cumulative density of the histogram information is plotted in Figure 12.

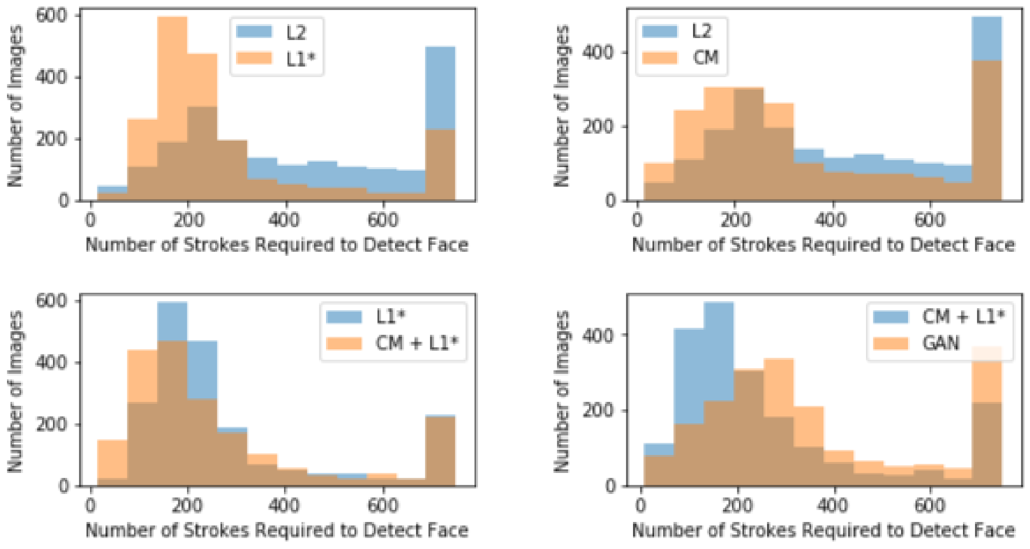

Figure 11: Comparing the rewards used to train the painting model. Histograms display how many strokes it took for FaceNet[10] to detect a face in the painting process for each model on 2000 held out images from the CelebA[7] dataset.

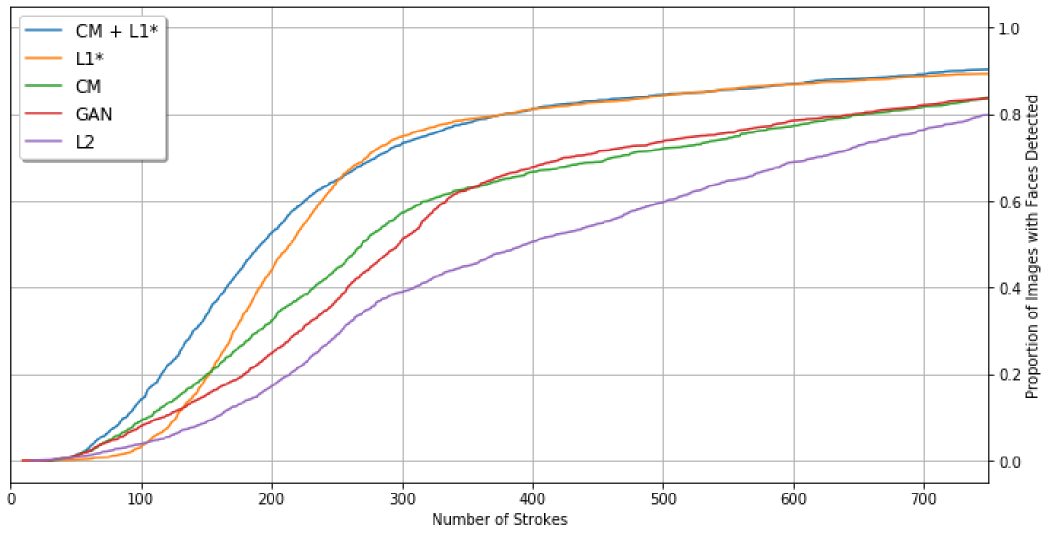

Figure 12: Plotting the proportion of test images with faces detected versus the number of strokes for painter models trained with various loss functions used to compute the reward.

	GAN	L2	L1*	CM	CM + L1*
Avg. # of strokes to face detection	365	435	287	355	269
% times a face couldn't be detected within 750 strokes	16.3%	20.1%	10.6%	16.1%	9.5%

Table 2: Models trained with various reward functions. Each ran on 2000 held out images for up to 750 strokes or until a face was detected with >50% certainty.

5 Discussion

The brush strokes produced from the state-of-the-art image to instructions model, Huang *et al.* [5] were clearly not realistic as seen in Figure 6 (b). The constraints in Table 1 were designed to force the Neural Renderer used in the Deep Reinforcement Learning model to behave more closely to how real painting works. In Figure 8, the Neural Renderer output from the Huang *et al.* model appears to be far more accurate than the constrained model. But when the robot agent painted the instructions, the constrained model's output looked more similar to the target image than Huang *et al.*. The constrained model's painting also looked more similar to it's corresponding Neural Renderer output proving that the constraints allow the model to know more accurately how the strokes will actually behave on a real canvas.

The model was biased towards painting darker colors as this was the darker colors granted more reward. $L1$ with loss clipping reduced this bias. In Figure 10, the model that used $L1$ with clipping ($L1*$) was less biased towards painting the dark hair in the image and would even paint the light background earlier than all the other models.

The constrained model's output looks more accurate compared to the state-of-the-art model, however, its planning process is not very human-like. In Figure 1, the strokes look very random until the very end of the process. A human painter's painting would resemble the subject of the painting early on in the process. Looking at Figure 11 and Table 2, both $L1*$ and CM models show great improvements over their $L2$ basis. Their stroke planning led to far faster face detection. $L1*$ provides a much steeper improvement to $L2$ than Content Masked Loss does. When adding Content Masking to $L1*$ there is still a significant improvement to early face detection. In Figure 12, the CM + L1* model is almost always the top most line. This indicates that this model's output is the easiest to recognize as painting faces early in the painting process.

The bottom right histogram of Figure 11 shows the Huang *et al.* [5] model with constraints versus our best model (using Content Masked $L1$ loss with clipping). Content Masked Loss and $L1$ with clipping provide a huge boost to human-like performance of the model as the paintings are far more recognizable with fewer brush strokes.

6 Acknowledgements

We thank Zhewei Huang, Wen Heng, and Shuchang Zhou [5] who open sourced their code[2] allowing us to modify it for this paper.

Thank you to Nicholas Eisley[3] for the photographs used in Figures 1, 6, 7, 9, and 10.

References

[1] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. *CoRR*, abs/1804.01118, 2018.

[2] David Ha and Douglas Eck. A neural representation of sketch drawings. *CoRR*, abs/1704.03477, 2017.

[3] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *CoRR*, abs/1809.01999, 2018.

[2] `https://github.com/hzwer/ICCV2019-LearningToPaintf`

[3] `http://neisley.squarespace.com/`

[4] Aaron Hertzmann. A survey of stroke-based rendering. *IEEE Computer Graphics and Applications*, 23(4):70–81, July/August 2003. Special Issue on Non-Photorealistic Rendering.

[5] Zhewei Huang, Wen Heng, and Shuchang Zhou. Learning to paint with model-based deep reinforcement learning. *CoRR*, abs/1903.04411, 2019.

[6] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2016.

[7] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[8] Reiichiro Nakano. Neural painters: A learned differentiable constraint for generating brushstroke paintings. *CoRR*, abs/1904.08410, 2019.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[10] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, pages 815–823. IEEE Computer Society, 2015.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[12] Ning Xie, Hirotaka Hachiya, and Masashi Sugiyama. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. *CoRR*, abs/1206.4634, 2012.

[13] Ningyuan Zheng, Yifan Jiang, and Dingjiang Huang. Strokenet: A neural painting environment. In *International Conference on Learning Representations*, 2019.

A VGG-16 Feature Extraction

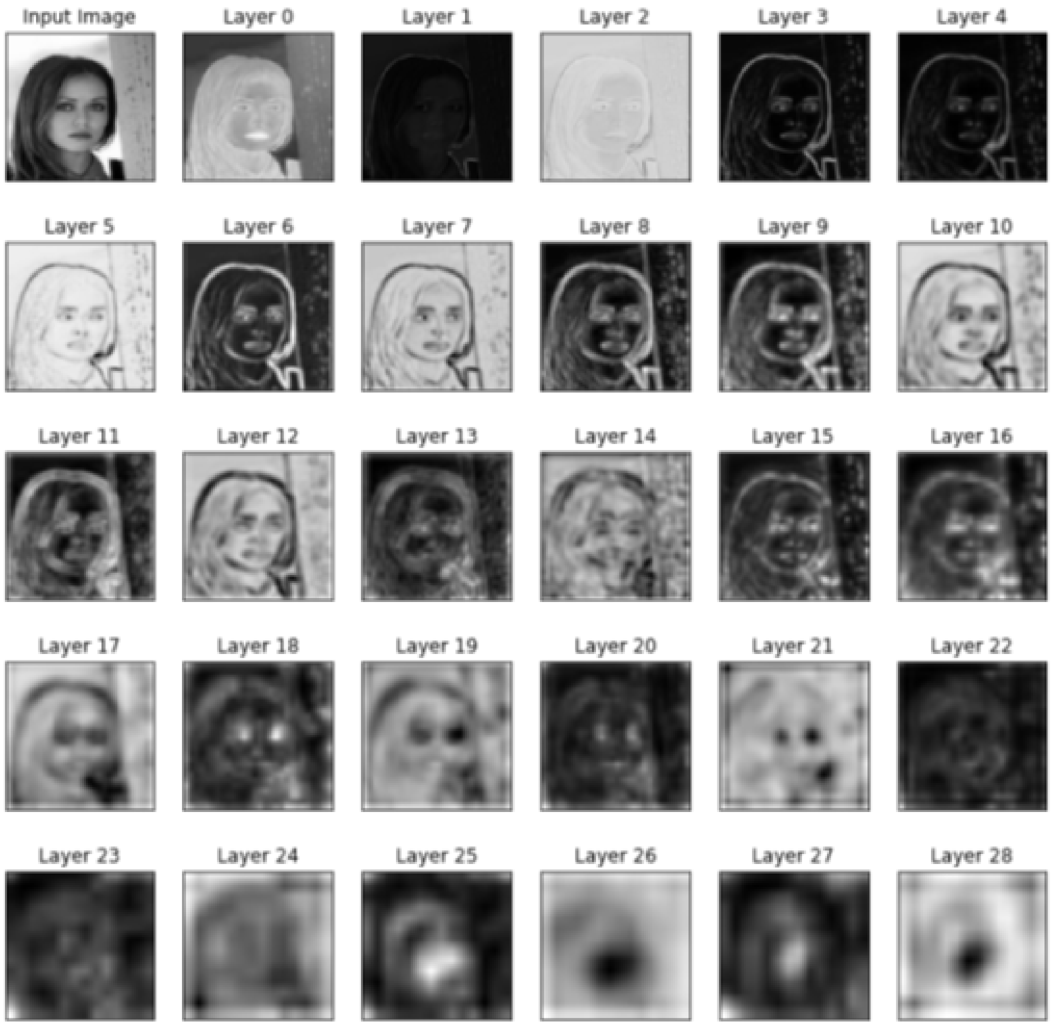

Figure 13: Features extracted from a given image at various layers within the VGG-16 [11] model.

B Effect of VGG-16 Layers in Content Masked Reward

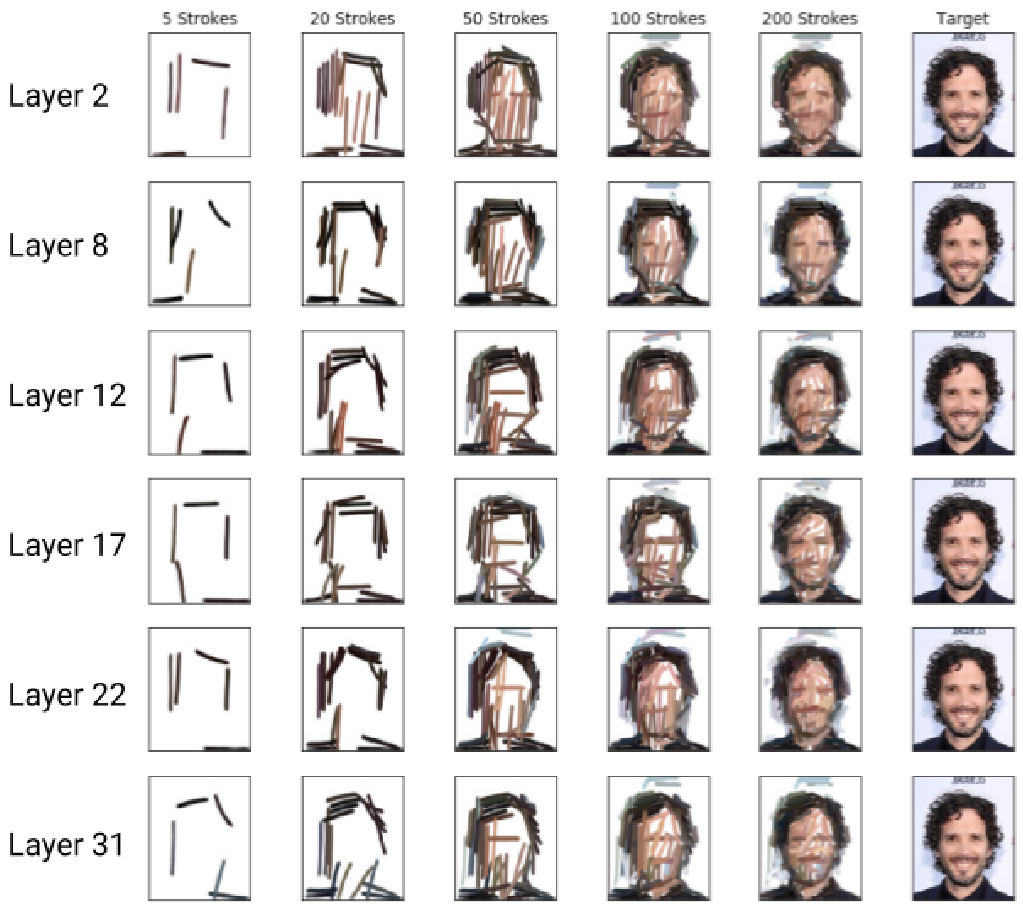

Figure 14: The painting process exposed. The model used was the Huang *et al.* [5] model with the constraints in Table 1. The models were trained using Content Masked Reward with various layers in the VGG-16 model used to compute the weighting features.