Neural Painters: A learned differentiable constraint for generating brushstroke paintings

Reiichiro Nakano[*]
Tokyo, Japan
Independent Researcher
reiichiro.s.nakano@gmail.com

Abstract

We explore neural painters, a generative model for brushstrokes learned from a real non-differentiable and non-deterministic painting program. We show that when training an agent to "paint" images using brushstrokes, using a differentiable neural painter leads to much faster convergence. We propose a method for encouraging this agent to follow human-like strokes when reconstructing digits. We also explore the use of a neural painter as a differentiable image parameterization. By directly optimizing brushstrokes to activate neurons in a pre-trained convolutional network, we can directly visualize ImageNet categories and generate "ideal" paintings of each class. Finally, we present a new concept called intrinsic style transfer. By minimizing only the content loss from neural style transfer, we allow the artistic medium, in this case, brushstrokes, to naturally dictate the resulting style.[2]

1 Introduction

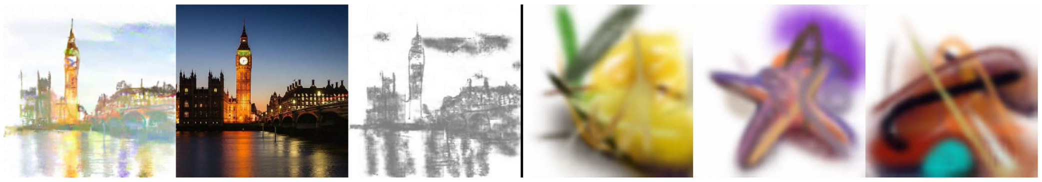

Figure 1: Using a neural painter as a differentiable image parameterization [22], we are able to directly optimize brushstrokes to minimize style transfer's content loss (left), or visualize ImageNet classes (right; pineapple, starfish, and violin categories).

There has been much work on using neural networks to generate painting-like images, some of the most notable being style transfer [7] and GANs [9], and all their many variations[4, 8, 18]. Most of these techniques generate images by having a network directly calculate the RGB value of each pixel.

However, artists don't paint by generating each individual pixel, they paint by generating *brushstrokes*.

There is a substantial body of research [12, 14, 23, 26] on *stroke-based rendering* (SBR), the field of digitally generating paintings by arranging brushstrokes on a canvas according to some optimization goal [15]. Recently, there has been significant progress on getting deep learning [20] to produce paintings by generating brushstrokes [5, 6, 27] instead of pixels.

[*]https://reiinakano.github.io

[2]We have provided Google Colaboratory notebooks to fully reproduce our experiments from scratch at https://github.com/reiinakano/neural-painters/tree/master/notebooks

Inspired by SBR and recent work on differentiable "world models" [11, 13], we perform various experiments with *neural painters*, which are differentiable simulations of a non-differentiable painting program.

2 Training Neural Painters

The main role of a neural painter is to serve as a fully differentiable simulation of a particular painting program. In this paper, we use an open source non-deterministic painting program called MyPaint[3]. We propose two different architectures for a neural painter, one based on variational autoencoders [19] and one based on generative adversarial networks [9]. A VAE neural painter generates smoothed-out brushstrokes, while a GAN neural painter generates "rougher" and more realistic brushstrokes.

3 Recreating SPIRAL Results

Ganin et al. [6] trained a neural agent called SPIRAL to learn to “paint” images by using the constrained action space of a real painting program. We use neural painters to recreate these results on three datasets: MNIST, KMNIST [2], and CelebA. The details and results for our agent are in the supplementary section. A significant advantage of this approach over SPIRAL is the amount of computing resources needed to achieve these results. Training SPIRAL involves using several multi-CPU/multi-GPU computers over the course of a few days. Our experiments can be reproduced entirely within the single GPU-environment of Google Colaboratory.

4 Towards Learning Human Strokes

In this section, we try a very simple but effective method to bias the agent to learn human strokes: *preconditioning*. Instead of training the agent from a randomly initialized state, we precondition the agent by forcing it to generate a particular set of strokes for each class. We test our process on MNIST. The experiment details and results are provided in the supplementary section.

5 As a Differentiable Image Parameterization

Differentiable image parameterizations are a technique developed by Mordvintsev et al. [22] to generate visualizations and art from pre-trained neural networks. Since neural painters are differentiable mappings from the brushstroke action space to a 2D image, they can be used directly as a differentiable image parameterization. We show two applications of neural painters as differentiable image parameterizations: visualizing ImageNet classes, and intrinsic neural style transfer. Some examples are given in Figure 1. Detailed setups and further results are provided in the supplementary section.

6 Conclusions

Constraints are a key element of creativity. The natural constraints that an artistic medium imposes upon the artist give a piece of art a distinct look from others. An artist attempting to paint a scene using oil paints will get a remarkably different result from someone trying to sketch the same scene with a pencil. In the same sense, using a neural painter leads to creative ways to achieve an objective - whether it be recreating digits, painting faces, or maximizing a pre-trained network’s activations. This paper explored the power of neural painters - a differentiable constraint learned from a non-differentiable real-life constraint. We believe this concept could be extended to different artistic mediums, such as splatter painting, or even 3D sculptures.

Acknowledgments

We would like to thank David Ha for his detailed feedback and encouragement throughout this work. We are also grateful to Ludwig Schubert and the Distill community for providing support for the diagrams in this paper. Many of our diagrams were repurposed from the article Differentiable Image Parameterizations by Mordvintsev et al. [22]

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[2] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

[3] MyPaint contributors. Mypaint, 2019. URL `https://github.com/mypaint/mypaint`.

[4] Ahmed Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. Can: Creative adversarial networks, generating" art" by learning about styles and deviating from style norms. *arXiv preprint arXiv:1706.07068*, 2017.

[5] Kevin Frans and Chin-Yi Cheng. Unsupervised image to sequence translation with canvas-drawer networks. *arXiv preprint arXiv:1809.08340*, 2018.

[6] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*, 2018.

[7] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. URL `http://arxiv.org/abs/1508.06576`.

[8] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[10] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[11] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462, 2018. URL `https://worldmodels.github.io/`.

[12] Paul Haeberli. Paint by numbers: Abstract image representations. In *ACM SIGGRAPH computer graphics*, volume 24, pages 207–214. ACM, 1990.

[13] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

[14] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460. ACM, 1998.

[15] Aaron Hertzmann. A survey of stroke-based rendering. *IEEE Computer Graphics and Applications*, (4):70–81, 2003.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[17] Zhewei Huang, Wen Heng, and Shuchang Zhou. Stroke-based artistic rendering agent with deep reinforcement learning. *arXiv preprint arXiv:1903.04411*, 2019.

[18] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural style transfer: A review. *arXiv preprint arXiv:1705.04058*, 2017.

[19] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[21] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[22] Alexander Mordvintsev, Nicola Pezzotti, Ludwig Schubert, and Chris Olah. Differentiable image parameterizations. *Distill*, 3(7):e12, 2018.

[23] Michio Shiraishi and Yasushi Yamaguchi. An algorithm for automatic painterly rendering based on local source image approximation. In *NPAR*, pages 53–58. Citeseer, 2000.

[24] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007. URL `http://eplex.cs.ucf.edu/papers/stanley_gpem07.pdf`.

[25] Tom White. Perception engines, 2018. URL `https://medium.com/artists-and-machine-intelligence/perception-engines-8a46bc598d57`.

[26] Georges Winkenbach and David H Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 91–100. ACM, 1994.

[27] Ning Xie, Hirotaka Hachiya, and Masashi Sugiyama. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. *IEICE TRANSACTIONS on Information and Systems*, 96(5):1134–1144, 2013.

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

A Related Literature

As early as 1990 [12], long before the sudden popularity of deep learning [20], there has been extensive work on automatically finding a set of brushstrokes to "paint" images [12, 14, 23, 26]. *Stroke-based rendering* (SBR) is the field of digitally generating paintings by arranging brushstrokes on a canvas according to some optimization goal [15]. The simple idea of using actual brushstrokes gives the outputs of SBR algorithms a very real painting-like texture.

Recently, there has been significant progress on getting neural networks to produce paintings by generating brushstrokes [5, 6, 27] instead of pixels. An example is the work done by Ganin et al. [6] on SPIRAL, a reinforcement learning agent trained adversarially to learn how to use a real painting program to generate images. A neural agent called SPIRAL learns to reconstruct images from MNIST, Omniglot, and CelebA by defining brushstrokes on a canvas.

Though impressive, the reinforcement adversarial learning framework used by SPIRAL is complex and requires significant computational resources [6]. Recent work by Ha and Schmidhuber [11] and Hafner et al. [13] have shown that building a differentiable world model of an environment can drastically reduce computational requirements for reinforcement learning algorithms. Learning a differentiable model for brushstrokes is the core idea underlying neural painters. Independent work using a similar idea was done by Zheng et al. [28] on StrokeNet and Huang et al. [17] in Learning to Paint. Our approach differs in that it models a non-deterministic painting program and uses a different network architecture and training methodology. Furthermore, we diverge from the goal of reconstructing a target image by exploring the use of a neural painter as a differentiable image parameterization, which, to the best of our knowledge, is a fully novel concept.

B Training Neural Painters

The main role of a neural painter is to serve as a fully differentiable simulation of a particular painting program. In this paper, we use an open source non-deterministic painting program called MyPaint[3], which is the same program used by SPIRAL. There are two main considerations for training a neural painter, the painting program's action space and the neural painter's architecture.

B.1 The Action Space

The action space defines the set of parameters that are used as control inputs for the painting environment. It serves as the interface that an agent can use to generate a painting.

For the experiments in this paper, we use a slight variation of the action space used by SPIRAL [6]. This action space maps a single action to a single brushstroke in the MyPaint program. An agent "paints" by successively generating actions and applying full brushstrokes on a canvas.

The action space consists of the following variables:

- Start and end pressure - Two variables that define the pressure applied to the brush at the beginning and end of the stroke.
- Brush size - Determines the radius of the generated brushstroke.
- Color - 3D integer vector determining the RGB color of the brushstroke.
- Brush coordinates - Three Cartesian coordinates on a 2D canvas, defining the brushstroke's shape. The coordinates define a starting point, end point, and an intermediate control point, constituting a quadratic Bezier curve.

The second consideration in training a neural painter is the architecture. We need an appropriate architecture and training paradigm to learn an accurate mapping from a point in the action space to the corresponding brushstroke. In this paper, we consider 2 approaches for training a neural painter.

B.2 Training a VAE Neural Painter

Our first approach is inspired by the two-stage method used by Ha and Schmidhuber [11] to learn a world model for a particular environment. [3]

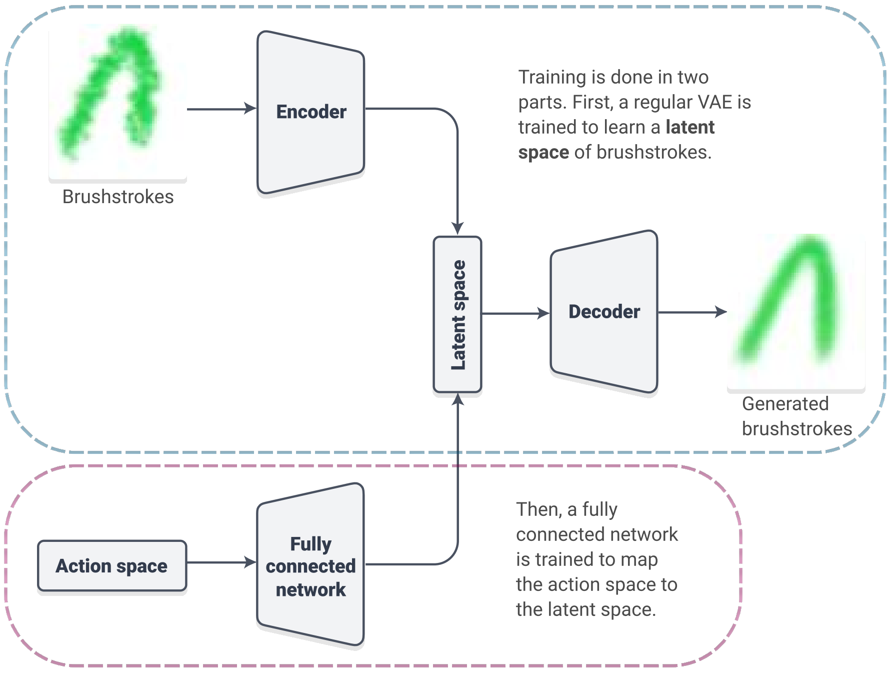
Brushstrokes
Encoder
Training is done in two parts. First, a regular VAE is trained to learn a **latent space** of brushstrokes.
Latent space
Decoder
Generated brushstrokes
Action space
Fully connected network
Then, a fully connected network is trained to map the action space to the latent space.

Figure 2: VAE neural painter training process

A variational autoencoder (VAE) [19] is trained to learn a latent space of brushstrokes. We then train a separate network to map an action to the point in latent space corresponding to the expected brushstroke. Unlike the approach in [11], we do not need a recurrent neural network (RNN) to map from actions to brushstrokes as there is no relationship between a brushstroke and the previous actions performed on the painting program. Figure 2 shows the training process for a VAE neural painter.

Figure 3 compares the results of a trained VAE neural painter with the real output of MyPaint.

[3]In the original paper, a VAE [19] is used to learn a latent space for all possible frames in an environment. An RNN is then used to predict the next frame of an environment given an action.

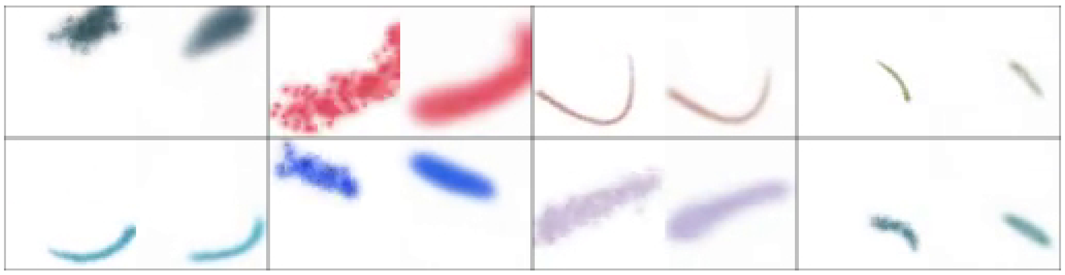
Figure 3: Pairs of real brushstrokes (left) and the corresponding VAE neural painter outputs (right). Notice how the VAE outputs are “smudged” versions of the ground truth.

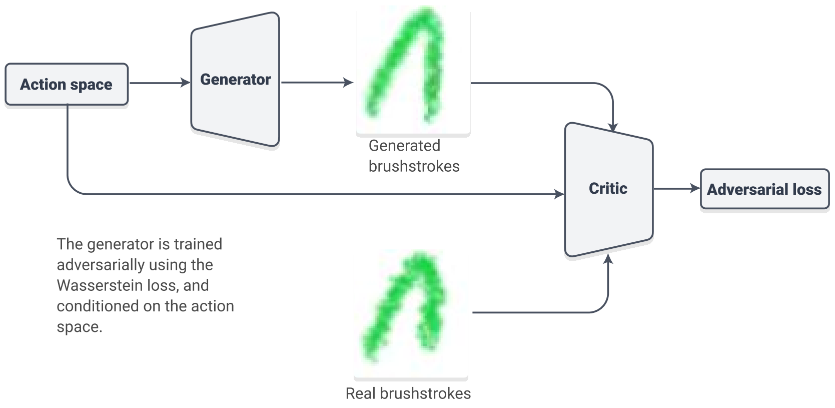
Action space
Generator
Generated brushstrokes
Critic
Adversarial loss
The generator is trained adversarially using the Wasserstein loss, and conditioned on the action space.
Real brushstrokes

Figure 4: GAN neural painter training process

The biggest weakness of the VAE neural painter is its “smudging” effect on the brushstrokes. Instead of accurately recreating the dotted texture of the larger brushstrokes, the VAE chooses to smoothen them out instead. Depending on the task, this inaccuracy could lead to less than ideal results when we transfer an agent from a neural painter to the real painting program.

B.3 Training a GAN Neural Painter

To solve this problem, we turn to another widely popular family of generative models, generative adversarial networks [9]. GANs have been shown to produce sharper images than VAEs, and this property could help the neural painter produce accurate brushstrokes.

Instead of relying on the reconstruction and KL divergence loss used by VAEs, we use an adversarial loss function to directly learn a mapping from actions to brushstrokes. Unlike a regular GAN, we do not inject noise into the input of the generator. Instead, we feed the generator the input action and have it map directly to a brushstroke. The discriminator is given real and generated action-brushstroke pairs and tries to distinguish whether the pair is valid or not. In this way, it is similar to a conditional GAN [21]. The training process, which uses Wasserstein loss [1, 10], is illustrated in Figure 4.

The results of training this network is shown in Figure 5. Although the recreation isn’t perfect, we can see that the outputs of the neural painter are “rougher” and more realistic as opposed to the smoothed out VAE brushstrokes.

How well an agent’s actions transfer from a neural painter to the real painting program depend directly on how accurate the neural painter’s outputs are. In this section we explored only two possible approaches, and there remains a lot of room for improvement in this area. We have provided accompanying notebooks for training our VAE and GAN neural painters to serve as a good starting point for anyone interested in training their own.

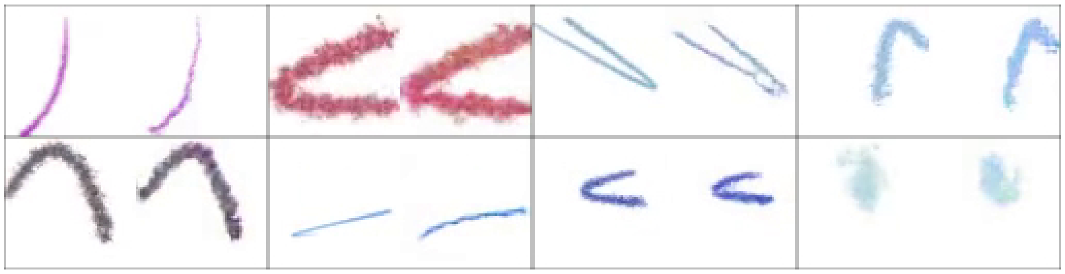

Figure 5: Pairs of real brushstrokes (left) and the corresponding GAN neural painter outputs (right). Notice how the GAN is able to capture the irregularities of real brushstrokes.

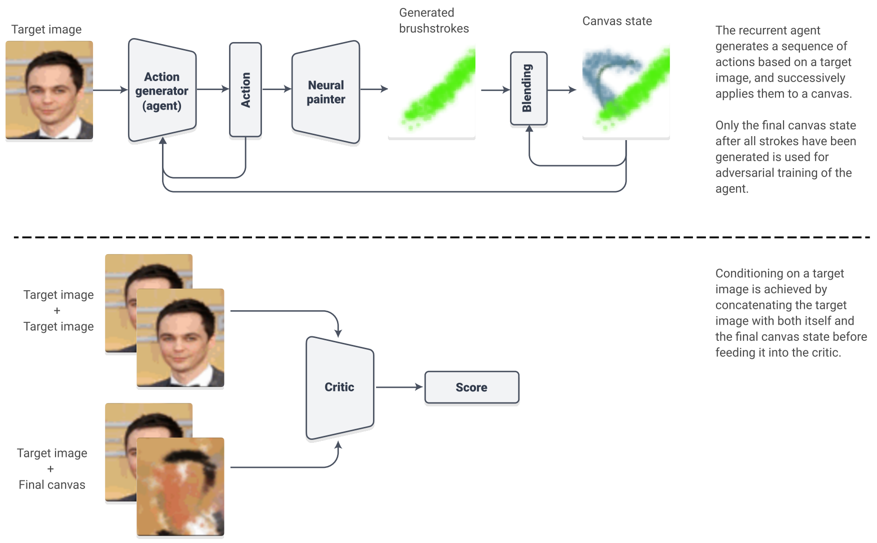
Target image
Action generator (agent)
Action
Neural painter
Generated brushstrokes
Blending
Canvas state
The recurrent agent generates a sequence of actions based on a target image, and successively applies them to a canvas.
Only the final canvas state after all strokes have been generated is used for adversarial training of the agent.
Target image + Target image
Critic
Score
Target image + Final canvas
Conditioning on a target image is achieved by concatenating the target image with both itself and the final canvas state before feeding it into the critic.

Figure 6: Adversarial training of agent for the purpose of reconstruction

C Recreating SPIRAL Results

Ganin et al. [6] trained a neural agent called SPIRAL to learn to "paint" images by using the constrained action space of a real painting program. In the paper, an agent was trained to paint images from three different datasets: MNIST, Omniglot, and CelebA. As the painting program is non-differentiable, the agent was trained using adversarial reinforcement learning.

Since a neural painter is fully differentiable, we don't need reinforcement learning techniques to perform the same experiments. We can simply train the agent using regular adversarial methods.

Our LSTM-based[16] neural agent is designed to take an input target image and output a set of actions. These output actions are connected directly to the neural painter and mapped to brushstrokes on a canvas. The agent's goal is to recreate the input image on the canvas using the constraints that the neural painter imposes. The idea is that the agent's outputs can be transferred directly to the real painting program, despite seeing only the neural painter during training.

When training this agent, instead of directly optimizing a pixelwise loss like L2 distance, we use an adversarial loss. As discussed in the SPIRAL paper, this leads to better and more well-behaved gradients that the agent can use to learn. Figure 6 shows the full training setup for our agent.

We test this approach's performance on three datasets: MNIST, KMNIST [2], and CelebA. You can explore the results for our agent in Figure 7.

A significant advantage of this approach over SPIRAL is the amount of computing resources needed to achieve these results. Training SPIRAL involves using several multi-CPU/multi-GPU computers

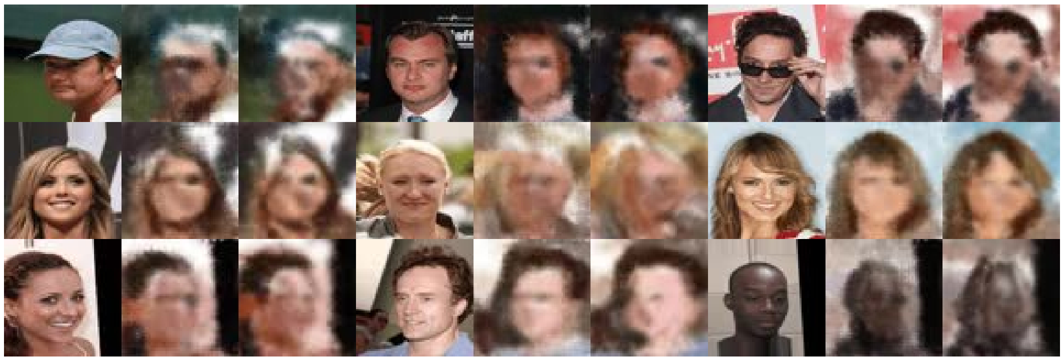

Figure 7: Each group shows three images: the target image (left), the neural painter output (center), and the generated brushstrokes transferred back to MyPaint (right).

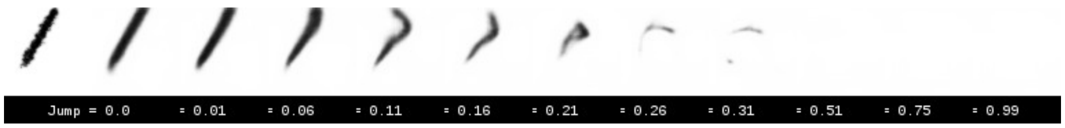

Figure 8: This image shows the effect of lifting the brush partially (i.e. a value between 0 and 1). Values between 0 and 1 are undefined and do not exist in the real environment, however, a neural painter must still "dream" up an output.

over the course of a few days. Our experiments can be reproduced entirely within the single GPU-environment of Google Colaboratory.

We believe this quick convergence can be partially attributed to the ease of credit assignment. When training our agent, the full gradients from each stroke are available and can be used directly in backpropagation, as opposed to the reinforcement learning paradigm used by SPIRAL, where only the reward at the end of painting is taken into account. This can be observed qualitatively by comparing the strokes produced by these agents on the CelebA dataset. SPIRAL's first few strokes are completely occluded by future strokes, while this does not happen with our approach.

C.1 The effect of discrete actions

In the previous section on training neural painters, we mentioned how we removed discrete variables from the action space that SPIRAL used. Why? It is not impossible to train a neural painter on a discrete action space. Our methods for training a neural painter work just as well for a continuous action space as it does with an action space with discrete variables [4].

However, there is one very important distinction: neural networks take continuous inputs. Even if a neural painter perfectly recreates the painting program's outputs when given a valid discrete action, its output between two discrete values is completely undefined. The neural painter must bridge this gap, and it is free to do it however it wants.

Figure 8 shows the output of the neural painter as the lift variable is moved continuously from 0 to 1. It shows an interesting effect. As the brush is lifted, the produced stroke seems to "flicker" until eventually disappearing completely. Somehow, the network has decided that these random flickers were the easiest way to represent a lift value between 0 and 1.

When the goal is to simply recreate a painting program's output, this behavior is not a problem. After all, a user can simply constrain the inputs to use only valid values. Unfortunately, this matters very much when we try to use the gradients from this neural painter to train an agent.

At best, this behavior makes the agent think it can produce impossible strokes, causing a discrepancy between the outputs of the neural painter and the painting program. One specific example is when the

[4]Brush size and stroke pressure are discrete variables with 10 levels. An extra binary flag is used to determine whether or not a brush is lifted i.e. a lifted brush produces no stroke.

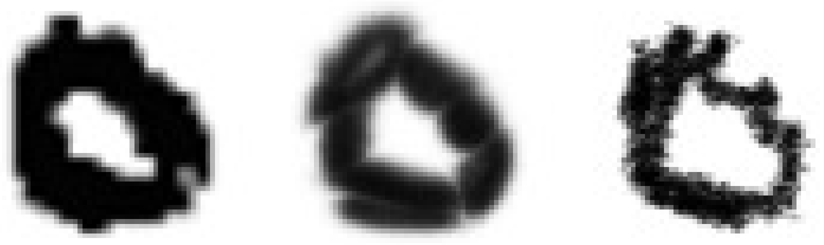

Figure 9: The figure shows three images: the target image (left), the neural painter output (center), and the generated brushstrokes transferred back to MyPaint (right). This result shows the effect of an agent thinking it can use a thicker brush than it actually can.

neural painter interpolates a stroke thickness between discrete values, which is rounded to the nearest brushstroke size when transferred to the painting program. This is shown in Figure 9.

At worst, it can kill training due to bad gradients. When the lift variable is used, the agent quickly gets into a state where it produces only invisible strokes. At this point, moving the agent's variables slightly in any direction would still produce invisible strokes. Essentially, there is no gradient for the agent to learn anything and training is stuck.

Figuring out how to handle these discrete actions will be an interesting research direction moving forward. Unfortunately, we cannot always side step this issue as we have done in this case by completely ignoring discrete variables. Many interesting environments (including the MuJoCo Scenes environment solved by SPIRAL) will have unavoidable discrete actions, and if we want to apply neural painters to those tasks, handling a discrete action space will be necessary.

D Towards Learning Human Strokes

In the previous section, you may have observed an interesting thing about the stroke order used by the painting agent. For any given agent, the stroke order generated for all target images will be similar.

For example, in the MNIST case, an agent may choose to draw all digits using a bottom-to-top, counter-clockwise approach. The agent does not bother using different strokes for different digits. Notice how 8's are usually treated as 3's with closed loops.

For CelebA, an agent might follow a certain set of steps for every target image e.g. shade the background, fill in the face shape, add hair, then add a stroke for the eyes.

In these experiments, the painting agent was trained with only one goal: recreate a given target image using brushstrokes. Since neural networks are "lazy learners" that converge to the nearest local minimum, the agent learns the simplest possible solution: use similar strokes everywhere. There's no reason to favor dissimilar, let alone human, stroke orders, as long as the final painted image looks as much like the target image as possible.

Of course, there is value in an agent that tries to draw like a human. First, we might be able to learn a model that accurately converts pixel character images to stroke vector data. Second, it might actually improve the original goal of reconstruction. Since digits in the MNIST dataset were actually drawn using a human order, an agent will likely find it easier to reconstruct some of the finer details of an image if it understands how a digit is usually drawn.

In this section, we try a very simple but effective method to bias the agent to learn human strokes: *preconditioning*. Instead of starting adversarial training with the agent's variables initialized randomly, we precondition the agent by forcing it to generate a particular set of strokes for each class. Our process is as follows:

- We begin by manually generating a single example for each class, with strokes we think are representative for the entire class. e.g. We can decide to draw 0's counter-clockwise, 1's top-to-bottom.

- Train the agent to reconstruct our manual strokes for each class via mean squared error, disregarding adversarial loss. Of course, we do not train the discriminator at this point.
- After the agent has started producing our manual strokes for every class, we can consider it preconditioned.
- Reintroduce adversarial loss. At this point, we can either completely drop off stroke reconstruction loss, or reduce its effect at a scheduled interval.
- Train normally.

We test our process on MNIST. Note that the approach requires us to provide only a *single* human example for each class. After training, the agent has more or less learned to stick with our original stroke order, with slight variations to make sure the target image is reconstructed. All 0's are drawn counter-clockwise, and all 1's are drawn top-to-bottom.

One way to explain why preconditioning works is that it changes the basins of attraction for the optimization problem. For an untrained, randomly initialized agent, the nearest local minimum for the adversarial loss is likely one that keeps stroke order similar, regardless of the target character. However, once it has been preconditioned to produce a certain set of strokes for each class, the closest local minimum becomes one that is as similar as possible to those strokes.

Although the approach shows promise, there is much room for improvement. Preconditioning relies on us knowing the correct class label for each image. Without this information, we will not be able to tell the agent to use a certain stroke order for a particular digit, simply because we do not know what digit it is.

Another weakness of the approach is its inability to properly handle multimodal classes that can be written in different ways. An example is the MNIST 7. 7's can be drawn either with or without the horizontal bar at the center, which are two different stroke orders. To apply the same technique, we need to condition the agent in a way that it distinguishes different modes of the class, and have it apply the correct stroke order. Solving these problems are a good direction for future research.

E As a Differentiable Image Parameterization

Differentiable image parameterizations are a technique developed by Mordvintsev et al. [22] to generate visualizations and art from pre-trained neural networks. Given a network trained on images (usually a convolutional network), we attempt to find a 2D image that maximally activates a particular neuron in the network. Instead of directly optimizing the individual RGB values of each pixel, we try different image generation processes that map some set of parameters to a 2D image. As long as this process is differentiable, we can directly optimize the parameters via backpropagation. Depending on the image generation process, the results can be strikingly different and beautiful. Various parameterizations such as CPPNs [24], Fourier transforms, and 3D to 2D mappings were demonstrated in [22].

Since neural painters are differentiable mappings from the brushstroke action space to a 2D image, they can be used directly as a differentiable image parameterization.

E.1 Visualizing ImageNet classes

We focus on visualizing the final layer of the pre-trained networks, corresponding to specific ImageNet classes. We find that a good way to improve the outputs is to optimize more than one pre-trained network at the same time[5]. This helps the outputs generalize better by reducing the effect of each individual network's imperfections. Figure 10 shows how to use a neural painter as a differentiable image parameterization to visualize ImageNet classes.

A fun way to interpret the results of a neural painter used as a differentiable image parameterization is as the answer to the question:

If you gave a pre-trained network a brush and asked it to paint a picture of the optimal panda, what would it paint?

[5]As done by Tom White in his work on Synthetic Abstractions [25]

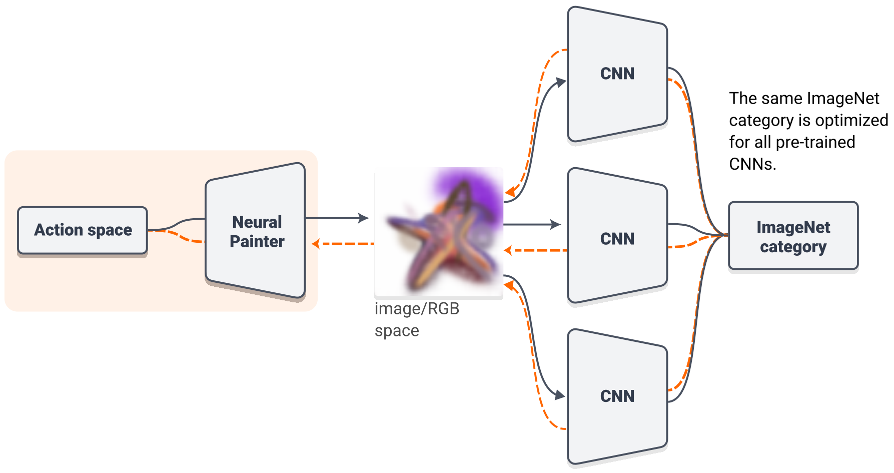

Figure 10: A neural painter as a differentiable image parameterization.

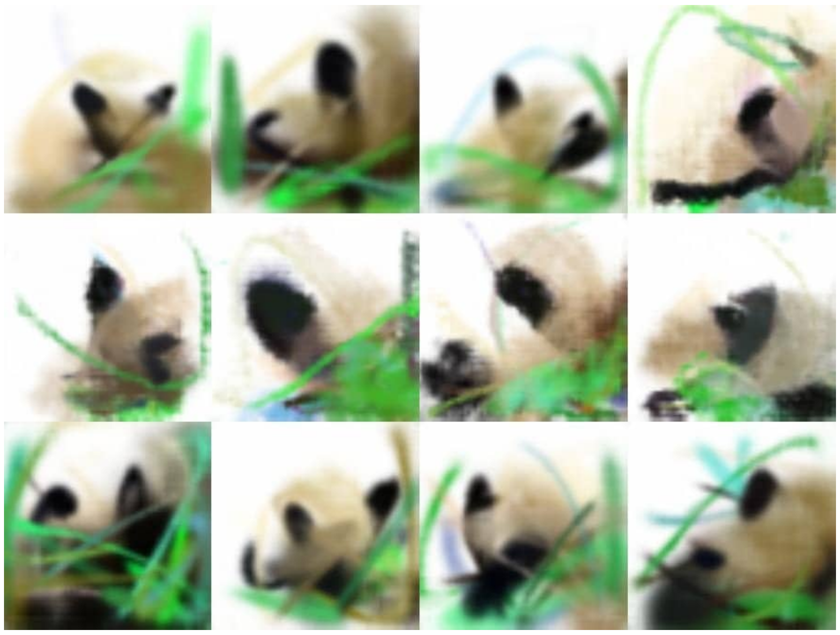

Figure 11: A neural network's paintings of the "optimal" pandas

Figure 11 shows examples of outputs generated by optimizing different ImageNet classes. The results show just how diverse the generated outputs can be for any given class, by simply tweaking the number of strokes, changing the neural painter, or using different pre-trained networks.

E.2 Intrinsic Style Transfer

As a differentiable image parameterization, neural painters are not limited to visualizing layers of a pre-trained neural network. We can produce various interesting effects depending on the loss we are optimizing for. One such effect is stroke-based painterly rendering [15] of a target image. With this method, we optimize brushstrokes to minimize only the content loss[6] in neural style transfer [7]. This setup is shown in Figure 12.

[6]To calculate the content loss between a content image and an output image, we take the mean squared error of their respective activations for a particular layer in a pre-trained neural network.

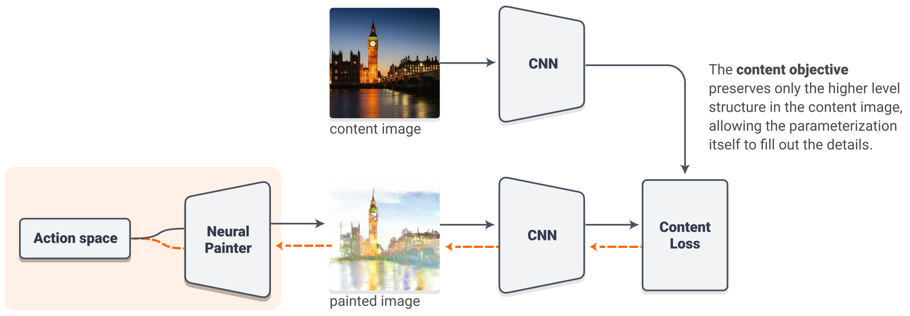
content image
CNN
The **content objective** preserves only the higher level structure in the content image, allowing the parameterization itself to fill out the details.
Action space
Neural Painter
painted image
CNN
Content Loss

Figure 12: Intrinsic style transfer using a neural painter

Intuitively, this technique lets us find brushstrokes that preserve only the higher-level content in the target image. The effect produced is that of painting only the meaningful parts of a target image, without caring about pixel-level reconstruction. The *style* is an intrinsic property dictated purely by the artistic medium, in this case, brushstrokes. Figure 13 shows some results of intrinsic style transfer.

By manually changing the primitives of the brushstroke, we can achieve vastly different styles. Note the difference in outputs by simply constraining the brush to use only grayscale values. Finding new styles by applying different constraints or using different artistic mediums[7] will be an exciting research path forward.

[7]Constraints could be simple modifications such as changing the color palette or making brushstrokes thinner. A good place to start looking for interesting brushstroke constraints is the extensive literature available on stroke-based rendering [15]. We can also completely change the medium and try differentiable parameterizations like CPPNs, which have been compared to light paintings[22].

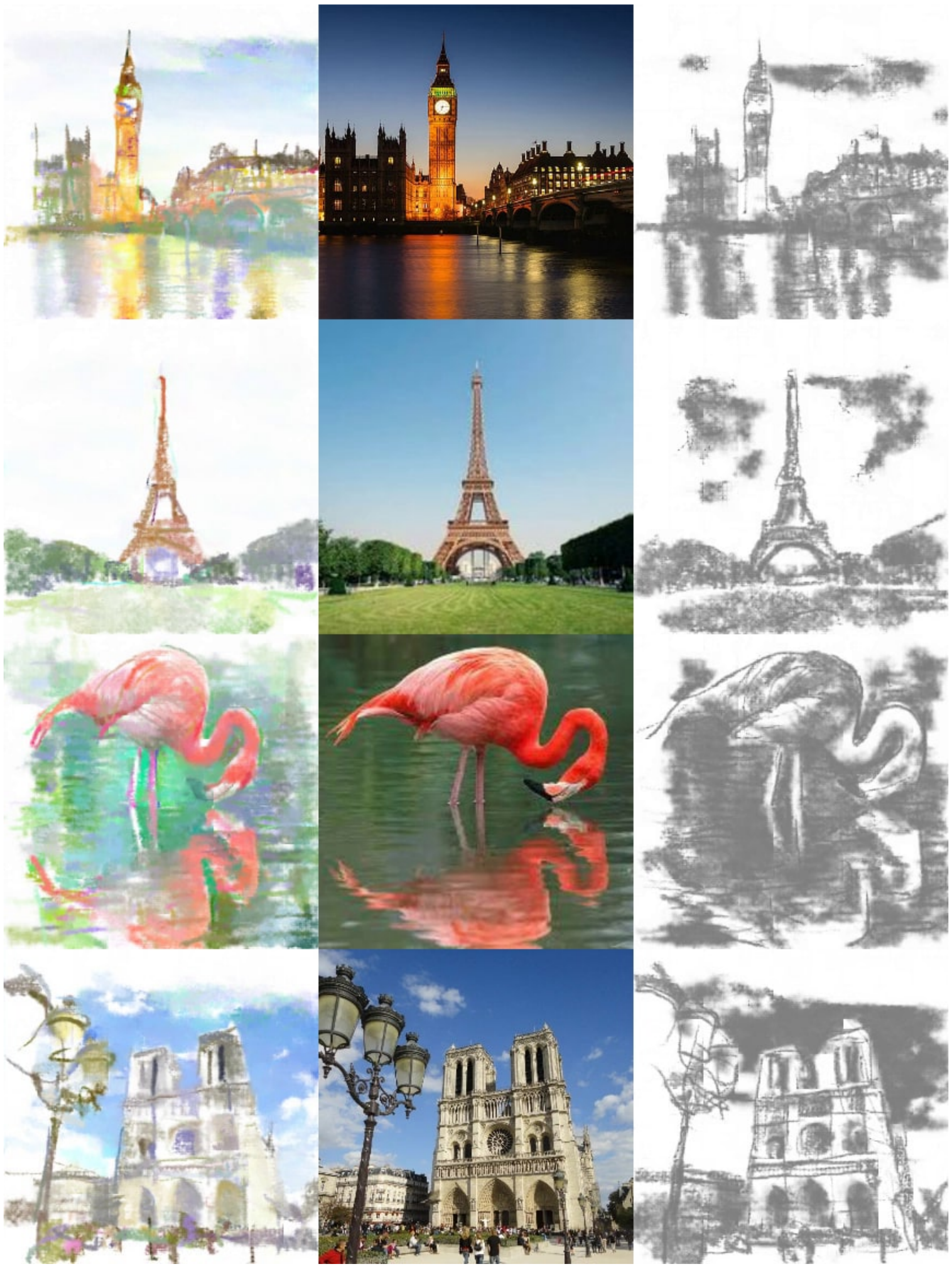

Figure 13: Results for intrinsic style transfer using a GAN neural painter for both colored and grayscale brushstrokes. The strokes are generated on multiple overlapping grids. Although the neural painter is only designed to output 64x64 pixels on a canvas, we can stitch multiple canvases together to achieve an arbitrary resolution, limited only by GPU memory.

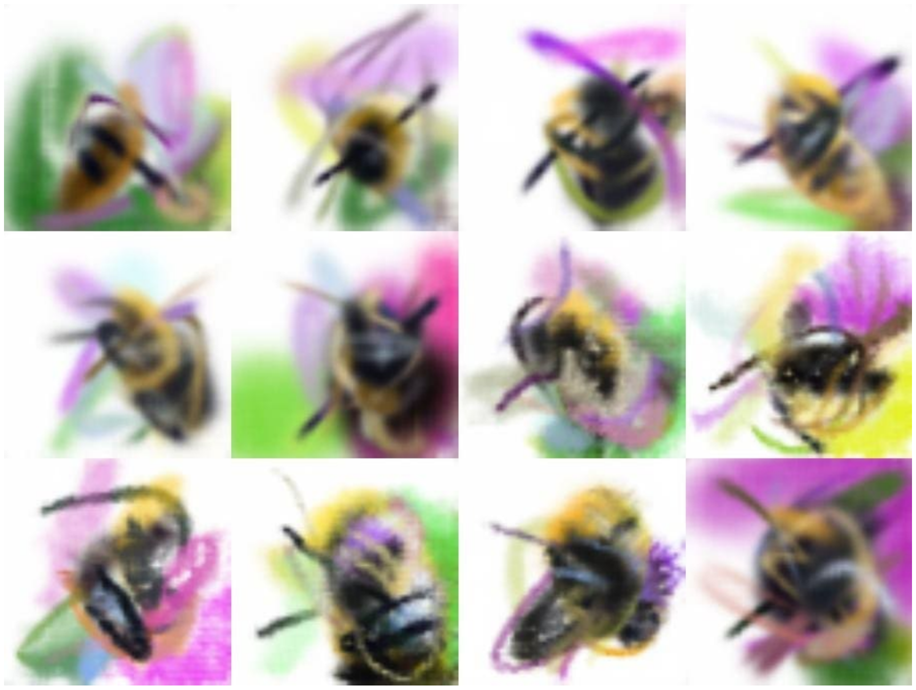
Figure 14: A neural network's paintings of the *optimal* bees

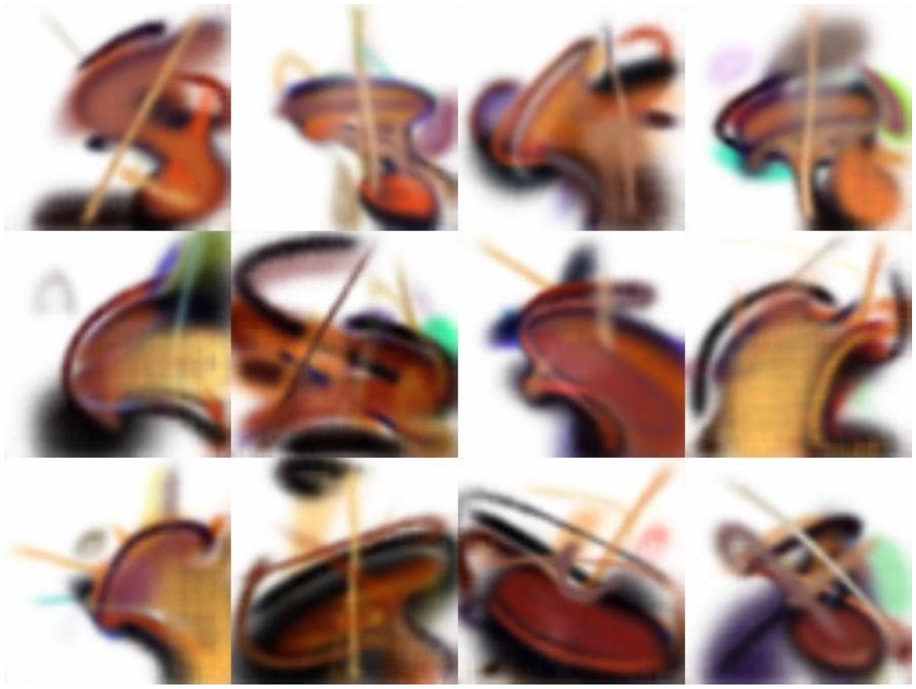
Figure 15: A neural network's paintings of the *optimal* violins

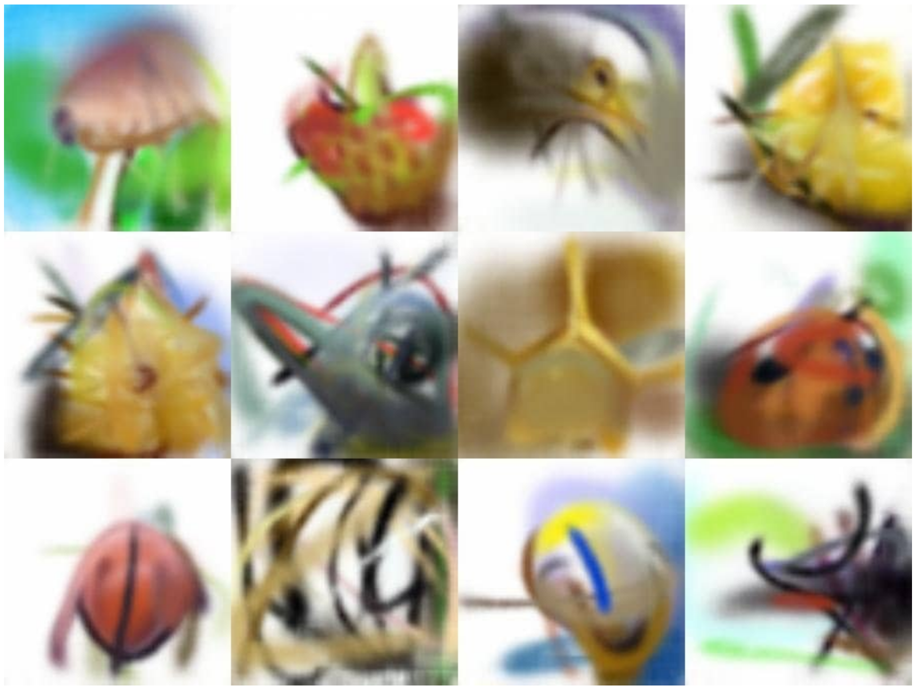

Figure 16: Other ImageNet class visualizations