Improving Badly Drawn Bunnies

Stuart Fong
Queen's University
stuart.fong@queensu.ca

Lazar Paroski
Queen's University
lazar.paroski@queensu.ca

Elijah Bajec Dickinson
Queen's University
elijah.b@queensu.ca

Kevin Wang
Queen's University
22wsl3@queensu.ca

Sofia Gerretsen
Queen's University
sofia.gerretsen@queensu.ca

Lucy Wu
Queen's University
wu.lucy@queensu.ca

Abstract—**Bad drawings often fail to convey an artist's true intentions. We present a class-conditioned variational autoencoder (VAE) that makes vectorized sketch images better resemble their intended class. The model is trained on a dataset of hand-drawn images from different classes and, given a user's drawing, can either complete it or improve upon it. We show that conditioning using the class label not only helps the model distinguish between different drawings but also improves its performance when trained on multiple classes.**

I. Introduction

A. Motivation

Human sketches are used as a way of communication and expression, able to capture emotions and moments using a sequence of strokes. While there have been many famous artists across history, countless others have not been fortunate enough to possess such talents. Many people wish to be able to draw, but their lack of skill can make it difficult to express their creativity. Bunnies, for example, are known to be beautiful and majestic creatures, but misrepresenting some of their features can cause the sketch to be recognized as another animal or even render it completely unintelligible [Yang et al., 2022]. While previous works [Yang et al., 2022], [Ha and Eck, 2018], [Wang et al., 2023], [Das et al., 2023] have developed methods to improve the quality of sketches, the proposed models do not have any way of knowing what is depicted in the input sketch. This means that any ambiguous strokes can cause the model to recognize the sketch as something different, misconstruing the author's original intentions. We attempt to remedy the issue of bad drawing skills by exploring methods for conditioning on generative models for vectorized sketch generation. In doing this, we provide a method to make sketch images look more like their intended class.

B. Problem Definition

The problem that we aim to solve is as follows: We are given an input sketch as a sequence of tuples of the form $(\Delta x, \Delta y, p)$, which represent the location and state of the pen. The first two values give the displacement of the pen after a certain time interval, while the third indicates whether the pen is currently touching the paper. Given an input sketch and its class, our goal is to output another sketch with features similar to the input but modified to better match images of the same class seen in the training data.

The first step is to encode a collection of sketches belonging to different classes into a lower-dimensional latent space. This is done to extract the features that are present in each class. Sampling from this space yields a vector representing the features of some sketch in the class. Decoding this vector, we obtain a new sketch that contains the features of the class seen in training.

In comparison to other approaches, our method uses the drawing's class to generate a sketch. We show that this better addresses the limitations of previous approaches in vectorized sketch generation to better handle the presence of multiple classes. We hypothesize that the addition of the class label allows the model to focus more on the features of a specific sketch drawing rather than the characteristics of drawings in its class.

We test the effectiveness of our proposed method using an autoregressive mixture-density variational autoencoder (VAE) similar to that of sketch-rnn [Ha and Eck, 2018]. We will show that the addition of a class label to the base sketch-rnn model results in better-quality sketches when trained on multiple classes.

II. Related Work

Many significant advancements have been made in the area of sketch generation. The goal of this field is to automate the process of creating intricate sketches that resemble those of human drawings. We focus on vectorized sketch generation, which aims to represent the sketches using geometric lines and shapes. One prominent approach is sketch-rnn [Ha and Eck, 2018] which uses a sequence-to-sequence variational autoencoder (VAE). It takes in a sequence of strokes, encodes it using a bidirectional recurrent neural network (RNN), and then decodes a sample using an autoregressive mixture-density RNN. Recent works [Wang et al., 2023], [Das et al., 2023] apply diffusion probabilistic models to vectorized sketch generation. They show how to represent the task of generating a sketch as a denoising diffusion process, where the recognizability of the sketch is gradually improved over several iterations.

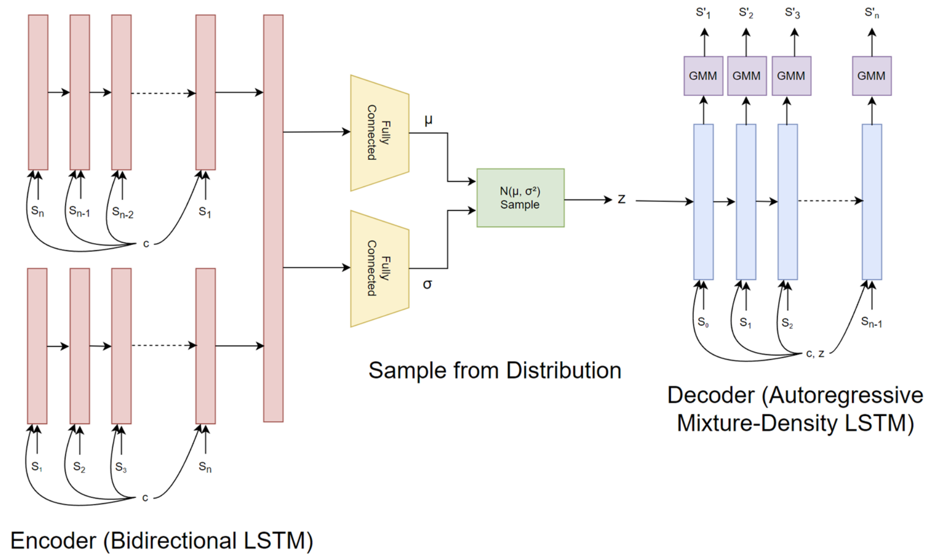

Fig. 1: The model architecture for our proposed model. Note that some features are excluded for simplicity. Refer to [Ha and Eck, 2018] for the full details.

III. Methodology

A. Dataset

The Quick, Draw! Dataset [Ha and Eck, 2018] contains a total of 50 million vectorized sketches across 345 classes that were created by players of the game Quick, Draw![1]. The data is preprocessed as follows:

- The drawing is aligned to the top-left corner, meaning that the minimum value is 0.
- The drawing is scaled uniformly to have a maximum value of 255.
- Strokes have a 1-pixel spacing.
- Strokes are simplified using the Ramer-Douglas-Peucker algorithm [Douglas and Peucker, 1973] with $\epsilon = 2$.

Each sketch is represented as a list of triples $(\Delta x, \Delta y, p)$ where Δx and Δy are the offset from the previous point with positive directions are down and right, and p is the pen state where 0 represents "pen down" and 1 represents "pen up".

We performed some additional preprocessing on the dataset by pruning sketches that either had too little or too many strokes. In our experiments, we removed all sketches whose number of strokes were more than 3 standard deviations from the mean for each class. This is to ensure that the drawings in the dataset are interesting enough to capture the drawing category, but simple enough that it can be learned by the model. Additionally, since we extend all input sketches to the length of the largest in the dataset, pruning long sketches results in a dramatic decrease in training time. Like [Ha and Eck, 2018], we normalize the offsets of each sketch by dividing them by the standard deviation of the offsets in each class.

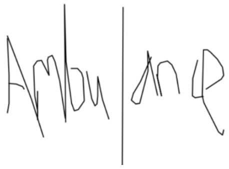

Fig. 2: An example of a training sketch that does not accurately represent ambulances. This sketch possesses an abnormally low number of strokes and would thus be filtered out during preprocessing.

B. Experiment Setup

Given n classes, we extend the points in the dataset into an $n+5$ element tuple $(\Delta x, \Delta y, p_1, p_2, p_3, c_1, c_2, \ldots, c_n)$ where the first two elements Δx and Δy are the same as in the Quick, Draw! Dataset, p_1, p_2, p_3 represent the possible pen states, and $c_1, c_2, \ldots, c_n$ are one-hot encoded variables representing the class. p_1 represents if the pen is currently touching the paper, drawing a line from the last point to the current point. p_2 represents if the pen is lifted from the paper, not drawing a connective line between the last and current point. p_3 represents if the drawing has ended, no longer rendering any points including and after the current one. We use this logic to both convert images to sketches and sketches to images.

We base our model on sketch-rnn [Ha and Eck, 2018], a variational autoencoder (VAE) that uses long short-term memory (LSTM) networks for its encoder and decoder. LSTMs are a type of neural network that excels at processing sequential data and reduces the impact of the vanishing gradient problem [Semeniuta et al., 2016]. sketch-rnn is a simple yet effective way of vectorized sketch generation, which we use as a proof of concept.

[1] quickdraw.withgoogle.com

1) Encoder: We first pass the input sketch as a sequence of points into the encoder, along with a class label. Our encoder is a bidirectional LSTM that takes the sequence and outputs a latent vector of size N_z. Specifically, an input sketch represented using a sequence of points S, and the same sequence in reverse, S_{reverse}, are fed into two LSTMs that comprise the bidirectional LSTM, $\text{LSTM}_{\leftarrow}$ and $\text{LSTM}_{\rightarrow}$, to obtain two hidden states: h_{forward} and h_{backward}.

$$\begin{aligned} h_{\text{forward}} &= \text{LSTM}_{\rightarrow}(S) \\ h_{\text{backward}} &= \text{LSTM}_{\leftarrow}(S_{\text{reverse}}) \\ h &= [h_{\text{forward}}; h_{\text{backward}}] \end{aligned}$$

We take the concatenated final hidden state, h, and pass it through a fully connected layer to get vectors μ and $\hat{\sigma}$ of size N_z. We convert $\hat{\sigma}$ to a positive standard deviation vector σ using the formula:

$$\sigma = \exp\left(\frac{\hat{\sigma}}{2}\right)$$

We then sample the latent vector z, using the formula:

$$z = \mu + \sigma \mathcal{N}(0, I)$$

where $\mathcal{N}(0, I)$ is a vector of independent and identically distributed (IID) Gaussian variables of size N_z and $z \in \mathbb{R}^{N_z}$ is the random vector used to produce the multivariate Gaussian distribution.

2) Decoder: A fully connected layer produces the initial hidden and cell states for the decoder, using $\tanh(z)$ as input. We then concatenate z and our class label to each stroke of the input sketch, using the resulting sequence as input to the decoder. This generates a sequence of parameters $(\Pi, \mu_x, \mu_y, \sigma_x, \sigma_y, \rho_{xy})$ for the Gaussian mixture model (GMM), and (q_1, q_2, q_3) which are probabilities for each pen state p_1, p_2, and p_3 respectively. Here, Π is a vector of mixture weights for the M normal distributions with $\sum_{i=1}^{M} \Pi_i = 1$, μ_x and μ_y are the means of each normal distribution with standard deviations σ_x and σ_y, and ρ_{xy} is the correlation coefficient. We use the $\exp$ operation to ensure that the standard deviations are positive, the $\tanh$ operation to ensure that the correlation coefficient is in $[-1, 1]$, and the softmax function so that Π and (q_1, q_2, q_3) both sum to 1. The probability of sampling stroke $(\Delta x, \Delta y)$ is

$$P(\Delta x, \Delta y) = \sum_{i=1}^{M} \Pi_i \mathcal{N}(\Delta x, \Delta y \mid \mu_{xi}, \mu_{yi}, \sigma_{xi}, \sigma_{yi}, \rho_{xyi})$$

We sample the output strokes from the GMM by first randomly choosing one of the normal distributions with probabilities from Π, then transforming two samples z_x and z_y from the standard normal distribution according to the following equations:

$$\begin{aligned} \Delta x &= \mu_x + \sigma_x z_x \\ \Delta y &= \mu_y + \sigma_y \left(\rho_{xy} \Delta x + z_y \sqrt{1 - \rho_{xy}^2}\right) \end{aligned}$$

The pen state is obtained by randomly sampling one of the states with probabilities q_1, q_2, and q_3.

After training for sketch reconstruction, we generate sketches conditioned on a latent representation, as described in the original sketch-rnn paper [Ha and Eck, 2018]. The previous approach computes the initial hidden and cell states using the latent vector z, then uses $S_0 = [0, 0, 1, 0, 0]$ concatenated with z as the initial input to the decoder. The next input S_{i+1} is obtained by sampling from the GMM parameters produced by the previous input S_i. In our approach, we additionally concatenate the class label to the input at each timestep.

3) Training: We train our model to minimize a loss function composed of the weighted sum of a Kullback-Leibler (KL) divergence term L_{KL} and reconstruction loss term L_R. As in the original sketch-rnn model, we additionally introduce a hyperparameter w_{KL} to control the contribution of L_{KL}.

The KL divergence is calculated for the latent vector distribution with respect to an IID Gaussian vector with zero mean and unit variance [Ha and Eck, 2018]. This loss function aims to shape the latent space so that a vector sampled from $\mathcal{N}(0, I)$ can be used by the decoder to generate a recognizable image.

$$L_{KL} = -\frac{1}{2N_z}(1 + \hat{\sigma} - \mu^2 - \exp(\hat{\sigma}))$$

The reconstruction loss L_R is computed as a sum of the log loss L_s of the offset terms $(\Delta x, \Delta y)$, and the log loss L_p of the pen states (p_1, p_2, p_3), to maximize our model's ability to predict the training data [Ha and Eck, 2018]. These are defined as follows:

$$\begin{aligned} L_s &= -\frac{1}{N_{\max}} \sum_{i=1}^{N_s} \log(\sum_{j=1}^{M} \Pi_{j,i} \mathcal{N}(\Delta x_i, \Delta y_i | \mu_{x,j,i}, \mu_{y,j,i}, \\ &\qquad \sigma_{x,j,i}, \sigma_{y,j,i}, \rho_{xy,j,i})) \\ L_p &= -\frac{1}{N_{\max}} \sum_{i=1}^{N_{\max}} \sum_{k=1}^{3} p_{k,i} \log(q_{k,i}) \end{aligned}$$

where $N_{\max}$ is the size of the longest sequence in the batch and N_s is a parameter where all points beyond index N_s when calculating L_s are discarded.

As done in [Ha and Eck, 2018], we anneal the L_{KL} term to optimize our model for reconstruction during early training stages. This is done as follows, where step is the training step, $n_{\min}$ is set close to 0, and $\mathcal{R}$ is set close to 1.

$$\begin{aligned} n_{\text{step}} &= 1 - (1 - n_{\min})\mathcal{R}^{\text{step}} \\ \text{Loss} &= L_R + w_{KL} n_{\text{step}} \max(L_{KL}, KL_{\min}) \end{aligned}$$

The result is that n_{step} converges to 1 as training progresses. The $KL_{\min}$ term is typically set to a low value, such as 0.10 or 0.50. This term will encourage the optimizer to put less focus on optimizing for the KL loss term L_{KL} once it is low enough, which empirically results in better metrics for the reconstruction loss L_R.

C. Experiments

In our research, we trained and tested our model on four classes: apples, flowers, cacti, and carrots. We create two separate versions: one using class labels, and one identical to the original sketch-rnn, with both sharing the same parameters. All classes consist of 70000 training images, 2500 testing images, and 2500 validation images. We trained the model for 10 epochs at 14000 steps each and a batch size of 10, alongside an Adam optimizer with a learning rate of 10^{-4} during this process. We decay the learning rate after each epoch by multiplying it by 0.92 each time. Values of n_{min}, $\mathcal{R}$, KL_{min}, and w_{KL} were 0.01, 0.9999, 0.3, and 1.0 respectively. Our encoder consisted of 512 nodes, and our decoder consisted of 2048 nodes, with the latent vector z having $N_z = 128$ dimensions. We also experimented with using a HyperLSTM [Ha et al., 2016] for the decoder, with 32 feature dimensions, 64 hyperdimensions, and 2048 hidden dimensions, keeping all other parameters unchanged. This was suggested in [Ha and Eck, 2018], and we found it gave a significant increase in model performance. Training was performed on an RTX 2070 GPU and i7 10700k CPU.

Our adjustments to the original sketch-rnn model allow us to experiment with fixing a latent vector while varying the class label, forcing our model to generate a specific class of sketch across a wide range of inputs. Figure 3 showcases the result of this process.

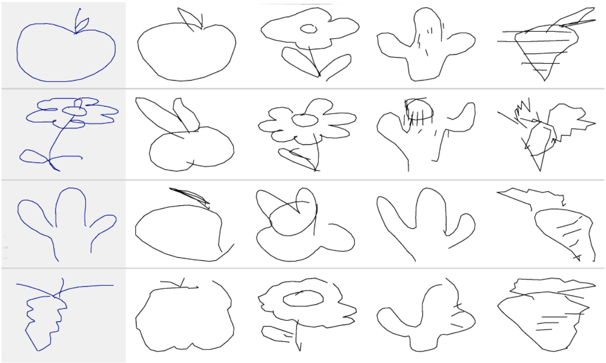

Fig. 3: Variation of class labels for fixed inputs of apples, flowers, cacti, and carrots. The leftmost column contains the original sketches (blue), with subsequent columns corresponding to a particular class. Each row corresponds to a single input.

Perhaps the most interesting use of this technique is to complete user-drawn sketches, illustrated in Figures 4 and 5. We achieve this by passing the incomplete sketch and its latent vector to the decoder, then using the resulting hidden and cell states of the LSTM to generate the remaining strokes. When trained on multiple classes, the original sketch-rnn may struggle to infer the class and produce inconsistent results during this process (Figure 4). With the conditional version, even simple shapes consistently admit completions resembling the intended class (Figure 5).

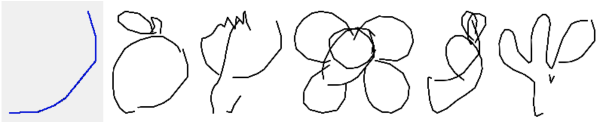

Fig. 4: Completions of the blue curve (left) by the unconditioned version.

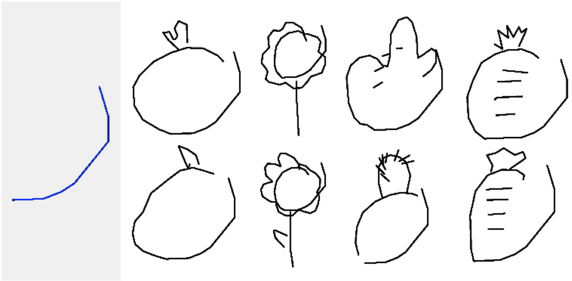

Fig. 5: Completions of the blue curve (left) by the conditioned version. Each column corresponds to a fixed label.

D. Evaluation Methods

We evaluate our model by using the Fréchet Inception Distance (FID) score [Heusel et al., 2017] for each class. This score measures the distance between two sets of images by comparing the means and variances of their features. These features are obtained from the Inception v3 model [Szegedy et al., 2016], trained on the ImageNet dataset [Deng et al., 2009]. This score is commonly used for assessing the quality of generated images and has been found to correlate with the recognizability of an image.

IV. Results and Discussion

We obtain the FID score from our model by first running the sketch images from our test set through our model and then calculating the score based on the test images and the generated sketch images. A lower FID score indicates a stronger similarity between the data set and generated images. The results are shown in Table I.

TABLE I: Results of FID score analysis.

Model	Apple	Cactus	Carrot	Flower	Average
Class label	39.81	35.92	**19.44**	**25.41**	**31.05**
No class label	**38.99**	**34.80**	48.42	29.20	37.10

As seen in the results, our class-conditioned version of sketch-rnn scores better on average than the unconditioned version of sketch-rnn when trained on multiple classes. While the original scores marginally better than the conditioned version on apples and cacti, we notice a large discrepancy between the flowers and carrots. When analyzing the outputs, the reason for this seems to be the original model's tendency to incorrectly guess the class and fail to reproduce fine detail. In Figures 6 and 7, we showcase this tendency with some randomly selected sketches from the output of each version, where each row corresponds to a different class.

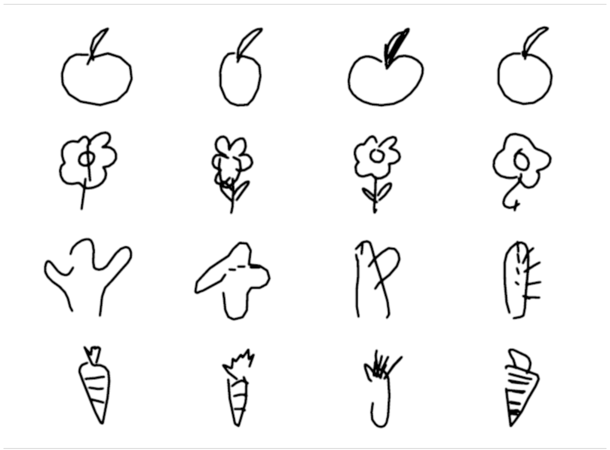

Fig. 6: Output of the conditional model on apples, cacti, flowers, and carrots. Each row corresponds to a different class.

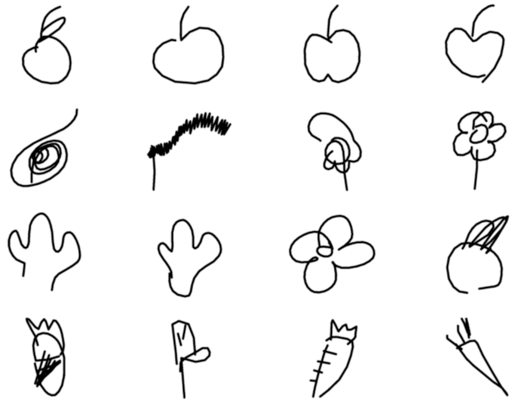

Fig. 7: Output of the unconditional model on apples, cacti, flowers, and carrots. Each row corresponds to a different class.

A. Ethical Considerations

This project aims to help those who want to improve their drawings. Although the intended purpose is positive, it can be subjective as to whether the returned output is an improvement from the original. In some use cases, the model output may be seen as offensive.

Sequence-to-sequence models transform one sequence to another and are frequently used in natural language processing (NLP) applications. Although the current purpose of our model is for improving sketches, it may be adapted to perform malicious activities such as forgery. If trained on handwritten signatures, the sequence-to-sequence model may be able to improve fake signatures to look more authentic.

B. Replication Package

A copy of our code and experiments can be found on our Github repository:
github.com/stuartfong1/improving-badly-drawn-bunnies

V. Conclusion

We have presented a class-conditioned model for vectorized sketch generation based on sketch-rnn. Given an input drawing and a class label, our model is able to construct a sketch image that belongs to the specified class but borrows features from the original image. The variational autoencoder encodes the input sketch into the latent space and then decodes a latent vector to produce a new sketch image. The random sampling in the Gaussian mixture model and by the decoder allows for generating a wide variety of images. In our evaluations, we show that our proposed model improves on the base sketch-rnn model when trained on multiple classes. The unique structure of VAEs opens up different use cases, including improving sketch drawings and sketch completion.

VI. Future Work

In this work we show that adding a class label to the sketch generation model helps to improve the recognizability of the output sketches. We have shown a proof of concept using sketch-rnn [Ha and Eck, 2018], which we hope will inspire further research in vectorized sketch generation. The following are a few suggestions:

1) Quality-Guided Generation: The goal of this project is to improve on a given vectorized sketch image. While generative models can correct a sketch to better match the distribution of the training data, the training data itself can be highly varied in terms of quality [Yang et al., 2022]. Having a method to assess the quality of a given sketch can help to guide the model to produce better looking sketches.

2) Other Generative Models: While variational autoencoders are able to generate high-quality and varied data, there are other generative models, such as generative adversarial networks (GANs) and diffusion probabilistic models that can perform this task better in terms of the quality of generated outputs [Wang et al., 2023], [Das et al., 2023].

3) Zero-Shot Generation: Investigating other ways to condition on generative models can enable generation on a wider variety of sketches. The recent CLIP [Radford et al., 2021] and prompt learning paradigm [Sain et al., 2023] has shown great success in sketching and may be used to condition on unseen classes.

VII. Limitations

Due to resource limitations, it was not possible to conduct a thorough study about how the performance of the model compares to the performance of the base sketch-rnn model as the number of classes increases. While there is a difference between the two models when using four classes, it is unknown if the performance of the two models will coincide with a greater number of classes.

Additionally, the limited resources restricted the ability to investigate how the two models compare on a wider selection of classes. Currently, the experiments use a small subset of the 345 available classes in the Quick, Draw! dataset. While the proposed model performs well on the chosen classes, the model may perform better or worse on other classes.

References

[Das et al., 2023] Das, A., Yang, Y., Hospedales, T. M., Xiang, T., and Song, Y. (2023). Chirodiff: Modelling chirographic data with diffusion models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society.

[Douglas and Peucker, 1973] Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122.

[Ha et al., 2016] Ha, D., Dai, A., and Le, Q. V. (2016). Hypernetworks.

[Ha and Eck, 2018] Ha, D. and Eck, D. (2018). A neural representation of sketch drawings. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[Heusel et al., 2017] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6626–6637.

[Radford et al., 2021] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.

[Sain et al., 2023] Sain, A., Bhunia, A. K., Chowdhury, P. N., Koley, S., Xiang, T., and Song, Y. (2023). CLIP for all things zero-shot sketch-based image retrieval, fine-grained or not. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 2765–2775. IEEE.

[Semeniuta et al., 2016] Semeniuta, S., Severyn, A., and Barth, E. (2016). Recurrent dropout without memory loss.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826. IEEE Computer Society.

[Wang et al., 2023] Wang, Q., Deng, H., Qi, Y., Li, D., and Song, Y. (2023). Sketchknitter: Vectorized sketch generation with diffusion models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

[Yang et al., 2022] Yang, L., Pang, K., Zhang, H., and Song, Y. (2022). Finding badly drawn bunnies. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 7472–7481. IEEE.