Teaching GANs to Sketch in Vector Format*

Varshaneya V
SSSIHL, Prashanti Nilayam Campus
Puttaparthi, Andhra Pradesh, India
varshaneya.v@gmail.com

Balasubramanian S
SSSIHL, Prashanti Nilayam Campus
Puttaparthi, Andhra Pradesh, India
sbalasubramanian@sssihl.edu.in

Vineeth Balasubramanian
IIT - Hyderabad
Kandi, Telangana, India
vineethnb@cse.iith.ac.in

ABSTRACT

Sketching is a fundamental human cognitive ability. Deep Neural Networks (DNNs) have achieved the state-of-the-art performance in recognition tasks like image recognition, speech recognition etc. but have not made significant progress in generating stroke-based sketches a.k.a sketches in vector format. Though there are Variational Auto Encoders (VAEs) for generating sketches in vector format, there is no Generative Adversarial Network (GAN) architecture for the same. In this paper, we propose a standalone GAN architecture called SkeGAN and a hybrid VAE-GAN architecture called VASkeGAN, for sketch generation in vector format. SkeGAN is a stochastic policy in Reinforcement Learning (RL), capable of generating both multidimensional continuous and discrete outputs. VASkeGAN draws sketches by coupling the efficient representation of data by VAE with the powerful generating capabilities of GAN. We have validated that SkeGAN and VASkeGAN generate visually appealing sketches with minimal scribble effect and is comparable to a recent work titled Sketch-RNN.

CCS CONCEPTS

• **Computing methodologies** → **Sequential decision making**; *Image representations*.

KEYWORDS

GANs, Sketch generation, Vector art, Policy gradients

ACM Reference Format:
Varshaneya V, Balasubramanian S, and Vineeth Balasubramanian. 2021. Teaching GANs to Sketch in Vector Format. In *Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP '21), December 19–22, 2021, Jodhpur, India,* Chetan Arora, Parag Chaudhuri, and Subhransu Maji (Eds.). ACM, New York, NY, USA, Article 01, 8 pages. https://doi.org/10.1145/3490035.3490258

1 INTRODUCTION

Since times immemorial, sketching has played an important role in human communication, much before languages were developed. To this day, sketches are used to represent abstract concepts including circuit diagrams, parse trees, and architectural designs [3]. Sketching involves visual, spatial and conceptual knowledge; understanding the process of sketching provides an insight into human cognition [3], beyond being useful for sketch-based applications.

Deep Learning (DL) is the most sought after paradigm recently for generative modelling. Some of the popular generative models in DL are VAEs [13] and GANs [5]. VAEs tend to maximize the likelihood of the generated data coming from the actual distribution while assuming a Gaussian prior. Different VAE architectures have performed well in generating various types of images ranging from handwritten digits [6, 13] to faces [13] to house numbers [6], CIFAR images [6] etc. On the other hand, in a GAN, the generator and discriminator play a minmax game until they reach an equilibrium. At this point, the distribution of generator is close to that of the original data. GANs are known to have been used for simple image generation in [19] as well as to more sophisticated tasks such as style-transfer [30, 34], super-resolution [15], image-to-image translation [11, 29], image in-painting [18] etc. All of the aforesaid works are limited to only raster images.

Vector graphics were introduced in computer displays in the 1960s. Since then vector graphics have been studied very intensely. These images do not degrade when transformations are applied and they require minimal amount of space to be stored and transferred. Most importantly, they can be rescaled infinitely. They are represented as curves and strokes.

Today, DNNs perform the state-of-art in image recognition as well as image generation tasks but the generative ability of DNNs is limited to raster images.There are only a handful of works that discuss sketch generation in vector format, let alone vector image generation in the wild. As of today, there are few works [2, 7, 9, 33] based on VAE for sketch generation in vector format. Currently there are no GAN architectures for sketch generation in vector format. In this work, we focus on developing GANs that can generate sketches in vector format. We consider a sketch as a tuple consisting of 2-D continous offsets and 3-D discrete pen states. This representation is also called as the "stroke-based format".

2 RELATED WORK

Literature on sketch generation is scarce, particularly with respect to vector images. Sketches have been used as templates for problems including recognition [20, 21, 23], eye-fixation or saliency [24], guessing a sketch being drawn [25] and parsing [22]. Interpretation of sketches and further refinement using hidden Markov models is presented in [26]. Sketch drawing by robot using pure image processing techniques is elicited in [28]. [31] uses GANs to generate sketches of human faces given digital portraits of their faces. This work is in rasterized version. There are also works such as [1, 17] which discuss the approaches to convert rasterized sketches into realistic images. Recent works such as DoodlerGAN[4] also consider sketches in raster format which aids them in modelling the process of sketching as generation of parts/portions which make up the whole sketch. Since the parts are fixed to 7 for birds (head, body, beak, tail, mouth, legs, wings) and 16 for terrestrial, aquatic, and

*The entire work was done during the period between Jun 2017 – Apr 2019.

ICVGIP'21, December 2021, Jodhpur, India

ACM ISBN 978-1-4503-7596-2…$15.00
https://doi.org/10.1145/3490035.3490258

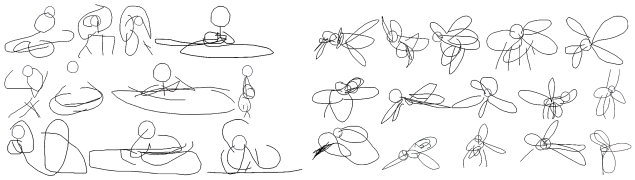

Figure 1: Scribble effect of [9] with yoga pose sketches (left) and mosquito sketches (right).

aerial creatures (e.g., paws, horn, fins, wings), it would become mandatory to specify the parts for generating sketch from any other category. This requires some manual intervention thus making it hard to generalize across different categories.

The first attempt in generating vector images is by [7] to generate Kanji (Chinese alphabet) characters using a two-layered LSTM, where each Kanji character is represented in a stroke-based format. Following [7], D. Ha et al. proposed a VAE model called the Sketch-RNN [9] for vector sketch generation. This model is trained on the QuickDraw dataset [12]. The sketches are represented in the stroke-based format. This work from Google Brain is state-of-the-art in unconditional generation, conditional reconstruction, latent space representation completing incomplete sketches etc. It produced visually appealing sketches when trained with a single category of sketch. The sketches are not visually appealing when a mix of category is used for training. Hence, in order to overcome this difficulty, [2] replaced the encoder of Sketch-RNN with a Convolution Neural Network(CNN) and eliminated the KL-Divergence loss. Since the convolution is spatial, the input to the this model is rasterized format of sketches from the QuickDraw dataset. Based on the Turing Test, the authors of [2] conclude that the models with CNN encoders outperformed those with RNN encoders in generating human-style sketches. K. Zhong in [33] extended the VAE proposed in [9] to create an end-to-end pipeline which takes in fonts in vector format to learn and generate novel fonts.

All the architectures mentioned for sketch generation in vector format are VAEs. A well known disadvantage with VAEs is that they tend to produce blurred images in case of raster images. Since there is no concept of blurring in vector images, stroke-based sketches produced by VAEs like Sketch-RNN [9] tend to draw smoother and more circular line segments that resemble an averaging of many sketches in the training set. This has been discussed under *Model Limitations* heading in the supplementary material of [9]. We would like to call this as the "*scribble effect*". Figure 1 shows this effect in the sketches of "yoga poses" and "mosquitos". Images in Figure 1 are generated using the pretrained models and code from one of the official repositories of Sketch-RNN[1].

Since GANs are known to produce sharp raster images, it is quite logical to expect that they alleviate the scribble effect in vector images. Towards this end, we propose a stand-alone GAN architecture called SkeGAN for generating sketches in vector format. To the best of our knowledge, this is the first GAN proposed for sketching in vector format. For fair comparison, we also propose a hybrid VAE-GAN architecture called VASkeGAN. We also compare our models with Sketch-RNN [9] on visual quality, training complexity, scribble effect and other ablation studies. Since there exists no standard measure to evaluate visual quality, we use human evaluation for assessing visual quality. Also both SkeGAN and VASkeGAN generalizes pretty well across different categories of sketches without requiring any manual intervention unlike [4].

[1]Link to the repository is here.

3 OUR CONTRIBUTIONS

3.1 Problem Setup

Sketches are considered to be a collection of 5-tuple $(\Delta x, \Delta y, q_1, q_2, q_3)$, where $(\Delta x, \Delta y)$ are the offsets to be moved along X and Y axes respectively and (q_1, q_2, q_3) are the pen-states. $(q_1, q_2, q_3) = (1, 0, 0)$ indicates that pen is on the paper, $(q_1, q_2, q_3) = (0, 1, 0)$ indicates it is lifted and $(q_1, q_2, q_3) = (0, 0, 1)$ indicates that the drawing has ended. Pen-state is modeled as a categorical random variable. All the drawings are assumed to start from origin. This is done by prepending the sketch with the start-of-sequence symbol $S_0 = (0, 0, 1, 0, 0)$. The offsets are modeled as a Gaussian Mixture Model (GMM) in the case of SkeGAN and as IID normal variable in the case of VASkeGAN. In both the models, we incorporate the parameter $\tau \in [0, 1]$ as defined in [9], to control the randomness or the variety in the generated samples. Sketch generation is done tuple-by-tuple until q_3 is not 1 or until the maximum length N_{max} is reached. A sketch is generated stroke by stroke, wherein the stroke at time-step i depends on all of the strokes at previous time-steps. In order to model this dependency, the generator is an auto-regressive model like LSTM or GRU. We have used an LSTM unlike Sketch-RNN [9], which has a HyperLSTM [8] as the auto-regressor. The discriminator distinguishes whether a batch of sketches has come from the dataset or from the generator. So it must understand the dependency between strokes of different time-steps in order to distinguish the sketches. Therefore, the discriminator is also an LSTM.

3.2 SkeGAN: A Sequential GAN for Vector Images

In a GAN architecture, the weights of the generator are updated based on the signal/reward from the discriminator. GANs have a limitation when there is a need to generate discrete tokens. The discrete outputs pose a difficulty for the gradient updates to be passed from discriminator to generator. In our case, the offsets are continuous random variables whereas the pen-states are discrete random variables. So, given a conventional GAN architecture, during the back-propagation, the gradient updates are passed without any difficulty for the offsets but not for pen-states. Also, any discriminator can guide a generator only when a complete sequence is given to it. This means that the discriminator cannot guide the generator while it is in the process of generating a sequence. Therefore, we propose a coupled GAN architecture with a combination of policy gradient and standard adversarial losses to generate both multi-dimension discrete and continuous tokens. The generator G in SkeGAN is a stochastic policy in RL which can sample tuples for the Monte Carlo search. By performing a Monte Carlo search, the reward signal from discriminator D is passed back to G even at its intermediate action value. Further, policy gradients are used for updating the weights of G via gradient ascent.

In the natural sketching process, the current position of the pen dictates whether it must be on the paper or must be lifted when it

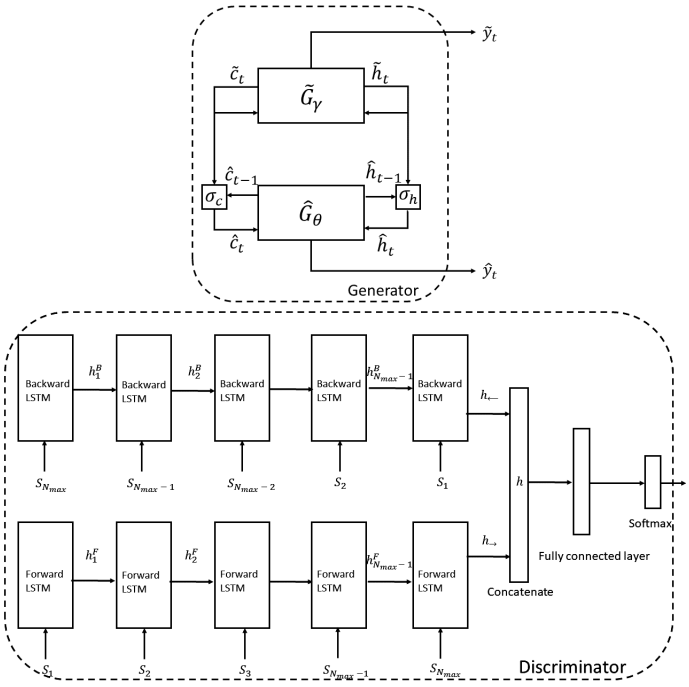

Figure 2: Generator (Top) and Discriminator (Bottom) of SkeGAN

is to be moved to the next coordinate. In other words, the offsets influence the pen-states. In addition to this, the previous pen-state influences the next pen-state. So, the current pen-state depends both on its previous state and the current offset. To model this coupling, we propose a fused generator G consisting of two generators viz. $\tilde{G_\gamma}$ for generating offsets and $\hat{G_\theta}$ for generating pen-states. Each of $\tilde{G_\gamma}$ and $\hat{G_\theta}$ is an LSTM with a hidden size of 512. The hidden and cell states of $\tilde{G_\gamma}$ at time-step t are denoted as $\tilde{h}_t$ and $\tilde{c}_t$, respectively. And that of $\hat{G_\theta}$ are denoted as $\hat{h}_t$ and $\hat{c}_t$, respectively. The coupling is achieved by having two update gates σ_c and σ_h with learnable parameters. The coupling effect can be mathematically described in the following equations:

$$\hat{h} = \sigma_h(W_h[\tilde{h}_t, \hat{h}_{t-1}] + b_h) \quad (1)$$

$$\hat{h}_t = \hat{h} \odot \tilde{h}_t + (1 - \hat{h}) \odot \hat{h}_{t-1} \quad (2)$$

$$\hat{c} = \sigma_c(W_c[\tilde{c}_t, \hat{c}_{t-1}] + b_c) \quad (3)$$

$$\hat{c}_t = \hat{c} \odot \tilde{c}_t + (1 - \hat{c}) \odot \hat{c}_{t-1} \quad (4)$$

where $\odot$ refers to element-wise multiplication and W_h, W_c, b_h and b_c are learnable parameters. The Generator of the proposed architecture is shown in the top portion of Figure 2. At each time-step t, $\tilde{G}$ generates $\tilde{y}_t$ and $\hat{G}$ generates $\hat{y}_t$. The parameters for the distribution of offsets are estimated from $\tilde{y}_t$ while those for the distribution of pen-states are estimated from $\hat{y}_t$ as given in [9].

The discriminator D is a Bidirectional LSTM with a hidden size of 256. A batch with one half containing generated sketches and another half from the dataset is shuffled and given to it. The forward and the backward hidden states of the LSTM are concatenated and mapped to a vector of dimension 2 followed by softmax activation to predict the probability of each sequence being real or fake. The discriminator is shown in the bottom portion of Figure 2.

Policy gradient based formulation: Let $G = (\tilde{G_\gamma}, \hat{G_\theta})$ and $D = D_\phi$. Since this policy gradient based formulation is meaningful only for discrete tokens, the following discussion pertains to $\hat{G_\theta}$ alone. Given a sequence $Y = (y_1, y_2, ..., y_T)$, where each y_t is a 3-tuple consisting of a valid pen-state i.e. $y_t \in \mathcal{Y}$ and $\mathcal{Y} = \{(1,0,0),(0,1,0),(0,0,1)\}$. At a time-step t, the state s of $\hat{G_\theta}$ is the sequence of produced tokens, which is $(y_1, y_2, ..., y_{t-1})$ and its action a is to select the next token y_t. It can be observed that though the policy model $\hat{G_\theta}(y_t \mid Y_{1:t-1})$ is stochastic, the transition state is deterministic after an action. In other words, if $s = Y_{1:t-1}$ is the current state and the action is y_t, then the next state s' equals $Y_{1:t}$. $D_\phi(1 : Y_t)$ is the probability of the sequence generated until time step t being real or not. Since there is no intermediate reward for an incomplete sequence, the objective of $\hat{G_\theta}$ is to generate a sequence from the start state s_0 which maximizes the expected end reward as given by:

$$J(\theta) = \mathbb{E}[R_T \mid s_0, \theta] = \sum_{y' \in \mathcal{Y}} \hat{G}_\theta(y' \mid s_0) \cdot Q_{D_\phi}^{\hat{G_\theta}}(s_0, y') \quad (5)$$

where $Q_{D_\phi}^{\hat{G_\theta}}(s_0, y')$ is the action-value function of the sequence and R_T is the reward of the complete sequence. $Q_{D_\phi}^{\hat{G_\theta}}(s_0, y')$ is the expected accumulative reward starting from state s, taking action a by following the policy $\hat{G_\theta}$. The next step is to estimate the action-value function. The estimated probability of a sequence being real, i.e. $D_\phi(Y_{1:T}^n)$, is considered to be the reward for $\hat{G_\theta}$. D_ϕ can provide the reward for a complete sequence only. Also, one must look for maximizing the long-term rewards. Therefore, to evaluate every intermediary step t, Monte Carlo search with a rollout policy $\hat{G_\beta}$ is used to sample the rest of the $T - t$ tokens.

Let the output of an N-time Monte Carlo search be represented as:

$$MC^{\hat{G_\beta}}(Y_{1:t}; N) = \{Y_{1:T}^1, ..., Y_{1:T}^N\} \quad (6)$$

where $Y_{1:t}^n = (y_1, ..., y_t)$ and $Y_{t+1:T}^n$ is sampled based on rollout policy and the current state. Here $\hat{G_\beta}$ is set to $\hat{G_\theta}$ itself for simplicity and speed. Thus, the action value function for $\hat{G_\theta}$ is defined as:

$$Q_{D_\phi}^{\hat{G_\theta}}(s = Y_{1:t-1}, a = y_t) = \begin{cases} D_\phi(Y_{1:T}^n), & Y_{1:T}^n \in MC^{\hat{G_\beta}}(Y_{1:t}; N) \quad \text{for t} \leq \text{T} \\ D_\phi(Y_{1:t}), & \text{for t} = \text{T} \end{cases} \quad (7)$$

The advantage of using D_ϕ as the reward function is that, since it is updated by the adversarial loss at every iteration, it improves in its capability to distinguish between real and fake. Due to this, it can provide better feedback to $\hat{G_\theta}$. The gradient of J(θ) with respect to θ is given by [27]:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{Y_{1:t-1} \sim \hat{G_\theta}} \left[\sum_{y_t \in \mathcal{Y}} \nabla_\theta \hat{G}_\theta(y_t \mid Y_{1:t-1}) \cdot Q_{D_\phi}^{\hat{G_\theta}}(Y_{1:t-1}, y_t) \right] \quad (8)$$

Using likelihood ratios [32], $\nabla_\theta J(\theta)$ becomes:

$$\nabla_\theta J(\theta) \simeq \sum_{t=1}^{T} \mathbb{E}_{y_t \sim \hat{G_\theta}(y_t \mid Y_{1:t-1})} [\nabla_\theta \hat{G}_\theta(y_t \mid Y_{1:t-1}) \cdot Q_{D_\phi}^{\hat{G_\theta}}(Y_{1:t-1}, y_t)] \quad (9)$$

The parameters of $\hat{G_\theta}$ are updated by the gradient ascent rule:

$$\theta \leftarrow \theta + \alpha_h \nabla_\theta J(\theta) \tag{10}$$

where $\alpha_h \in \mathbb{R}^+$ is the learning rate at the h^{th} iteration.

Apart from policy gradient rule to update θ, we also have the standard adversarial loss to guide the generator $\tilde{G_\gamma}$. It is to be noted that, as in [9], offsets are modeled as a GMM. We also scale the pen-state outputs, the mixing weights and variance parameters of the GMM by a temperature parameter denoted as τ to control the level of randomness in generation. τ lies in [0, 1]. Further, in order to ensure stability and faster convergence, we pre-train both the generator and discriminator. Complete details regarding training of SkeGAN is provided in the supplementary material.

3.3 VASkeGAN: VAE-GAN for Sketch Generation

Since VAEs are good at representing data in the latent space and GANs at generating data, we propose a VAE-GAN [14] based architecture *VASkeGAN* for sketch generation. The VAE in VAE-GAN [14] produces meaningful representation of the data, which helps the generator in generating data close to its actual distribution. [14] has shown that VAE-GAN architecture alleviates blurring for *CelebA* [16] and *Labeled Faces in the Wild* (LFW) [10] datasets. Due to improved generating capabilities over a conventional VAE, we relax the assumption used in [9] that the offsets are sampled from a GMM, and assume that offsets are sampled as IID from $\mathcal{N}(0, 1)$. In other words, our inductive bias is simpler. The proposed architecture is shown in Figures 3 and 4.

The encoder is a bi-directional LSTM with 256 hidden units that encodes the given sketch in to a latent vector z of size N_z, as in a standard VAE. The decoder of the VAE doubles up to be the generator of the GAN. It is an LSTM with 512 hidden units that samples sketches conditioned on z. The initial hidden state h_0 and cell state c_0 are derived from z via the following equation: $[h_0; c_0] = tanh(W_z z + b_z)$, where W_z and b_z are learnable parameters. The input x_i to the decoder at each time-step i is the previous point S_{i-1} concatenated with z. The output of decoder $y_i \in \mathbb{R}^7$ at every time-step i, can be split as:

$$[\mu_{x,i}, \mu_{y,i}, \tilde{\sigma}_{x,i}, \tilde{\sigma}_{y,i}, (\hat{q}_{1,i}, \hat{q}_{2,i}, \hat{q}_{3,i})] = y_i \tag{11}$$

Exponential operation is applied to $\tilde{\sigma}_{x,i}$ $\tilde{\sigma}_{y,i}$ so that the standard deviations are non-negative. Softmax is used on $\hat{q}$'s to calculate the probabilities of the pen-states. The sampling of Δx_i and Δy_i is akin to the reparametrization trick. In other words, $\Delta\tilde{x}_i \sim \mathcal{N}(0, 1)$ and $\Delta\tilde{y}_i \sim \mathcal{N}(0, 1)$ are independently sampled and Δx_i and Δy_i are calculated as:

$$\Delta x_i = \Delta\tilde{x}_i \cdot \sigma_{x,i} + \mu_{x,i} \quad \Delta y_i = \Delta\tilde{y}_i \cdot \sigma_{y,i} + \mu_{y,i} \tag{12}$$

For discriminator, we experimented with both LSTM and GRU with 512 number of hidden units.

Encoder and decoder (generator) are trained with standard KL loss (L_{KL}) and reconstruction loss (L_R). Further, generator is also guided by the standard adversarial loss. As in SkeGAN, temperature τ is used here as well. Further details regarding training are provided in the supplementary material.

4 EXPERIMENTS AND RESULTS

4.1 Datasets, Baselines and Performance Metric

We have used the QuickDraw Dataset created in [9] for training and experimentation. QuickDraw consists of sketches belonging to 345 different categories. Each category consists of 75000 sketches for training, 2500 for validation and 2500 for testing. As of today, QuickDraw is the only dataset with a large number of sketches in vector format for training and testing. We trained VASkeGAN and SkeGAN on the categories of sketches such as cat, firetruck, mosquito and yoga poses. Since there is huge variety of sketches in QuickDraw, we chose these categories because they represent the sketches of humans, animals, insects and non-living things, and capture the diversity of the dataset. We have trained a separate model for each of the categories for both VASkeGAN and SkeGAN architectures, as done for Sketch-RNN in [9]. VASkeGAN was trained for 200000 iterations on the aforesaid sketch categories. The total number of training rounds for cat, mosquito, yoga and firetruck sketches are 4, 3, 6 and 4 respectively. The results of SkeGAN and VASkeGAN along their implications are discussed subsequently. We have quantitatively assessed the visual appeal of the sketches by performing a human evaluation with a group of 45volunteers, to rate the sketches on a scale of 1 – 5 on the categories such as clarity, drawing skill and naturalness. The experiments conducted are in line with those done on Sketch-RNN and for results of Sketch-RNN on the corresponding experiments, due to paucity of space, the readers are requested to refer [9].

4.2 Results

Unconditional Generation of SkeGAN: All of the sketches are generated with a single starting tuple S_0. Subsequently tuples are generated until the pen-state q_3 equals 1 or the number of tuples generated becomes N_{max}. Figure 5 shows some of the sketches generated for the aforesaid categories. Note that the sketches to the left of the separating line in Figure 5 are generated just after the pre-training of the G_θ. The sketches to the right of the separating line are generated by the trained model. The visual appeal of the generated images on the right favours the combination of policy gradients and adversarial loss for generating sketches. The images on the left of separating line indicates that pre-training is essential but not sufficient to generate good sketches. It is very clear from Figure 5 that the scribble effect as seen in Figure 1 is absent.

Sketch Completion by SkeGAN: In order to test the extrapolative abilities of SkeGAN, we feed a partially drawn sketch and observe how it can figure out various endings for the incomplete sketch. The generator which is trained with sketches of a particular category, is conditioned with an incomplete sketch from that category. The hidden state of the generator after this conditioning is h, which contains the semantic information of the incomplete sketch. Using this information, the remainder of the tuples for the sketch are sampled from the generator, with h as its initial hidden state. Figure 6 shows various completions for the same input sketch at a temperature τ of 0.25. The completed sketches shown are indeed meaningful and visually appealing, which highlights the creative aspect of SkeGAN.

Sketches Generated by VASkeGAN: We allow the trained model to generate sketches after being conditioned by a sketch from a particular class. A sample of the generated images are shown in Figure

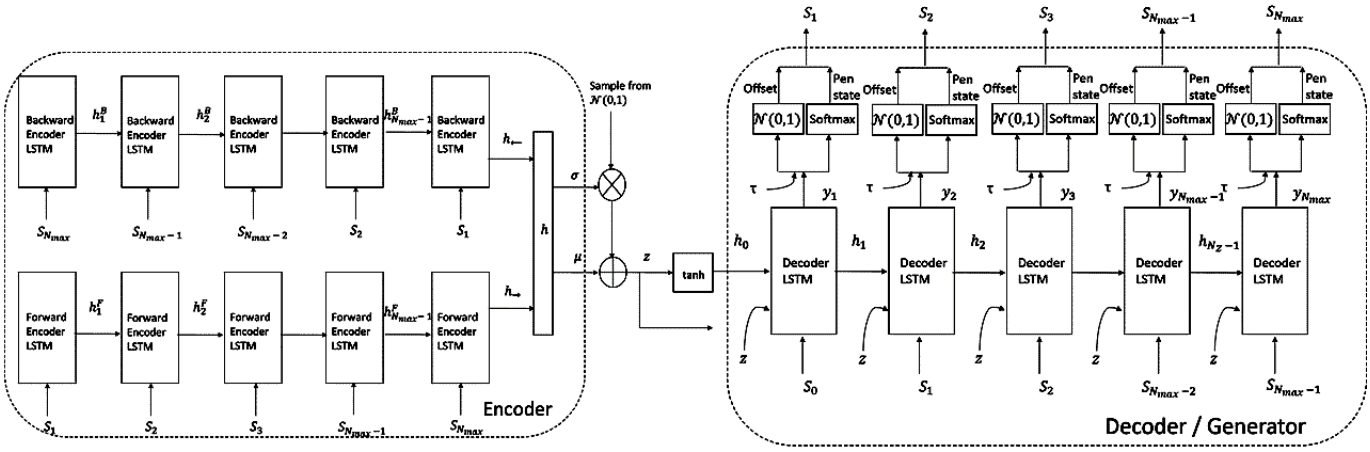

Figure 3: Encoder and Decoder of VASkeGAN architecture.

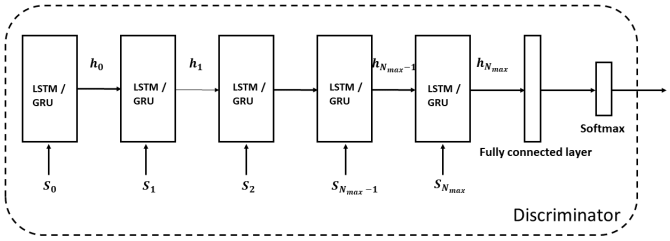

Figure 4: Discriminator of VASkeGAN architecture.

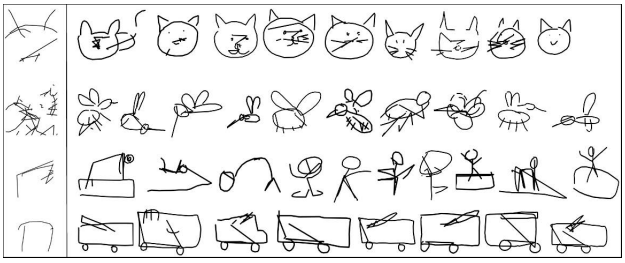

Figure 5: Sketches generated by SkeGAN after pre-training (left) and after the actual training (right).

7. This confirms that the model is indeed generating meaningful sketches and not random strokes. Here too, the scribble effect is absent.

Following this, we experimented by training VASkeGAN as a standalone GAN wherein only the generator (decoder) and the discriminator are retained. We found that the sketches generated in this case for all the categories are just doodles without any discernible entity. This experimentally validates the fact that discrete outputs pose a difficulty for the gradient updates to be passed from discriminator to generator for the weight update. Hence empirically strengthening the argument in favour of the formulation of SkeGAN.

Transfer Learning by VASkeGAN: The weights of the model trained on cat sketches for 200000 iterations are transferred to train

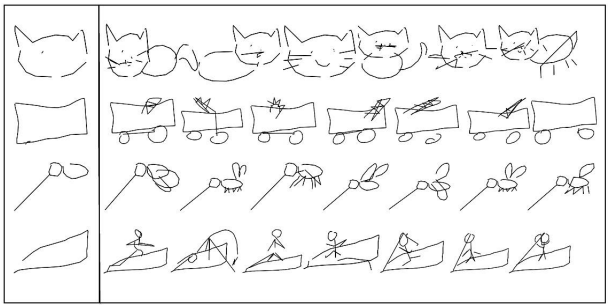

Figure 6: Partially drawn sketches (Left). Completed sketches by SkeGAN (Right).

two different models on pig and aeroplane sketches. In this case, the training is done for only 100000 iterations. Figure 8 shows the pig and aeroplane sketches generated by transferring the learnt representations across categories. This shows that VASkeGAN generalizes well and is able to transfer the knowledge across categories. Transfer learning on SkeGAN led to a mode collapse, which is a direction of future investigation.

Visual Appeal: The average of scores for a particular criterion across different categories are tabulated in Table 1. Sketch-RNN [9] performs closest to the groundtruth while SkeGAN is not too far away. Further, SkeGAN outperforms VASkeGAN, again emphasizing the importance of policy gradients to model pen states. Overall, SkeGAN generates sketches that are clear, artistic and natural as those generated by Sketch-RNN and those in the dataset.

5 DISCUSSIONS

We now present some ablation studies related to our models and also compare the training times of SkeGAN with VASkeGAN.

Effect of Temperature τ:

Conditional Generation of VASkeGAN: We fix a sketch from each category and vary the temperature τ to see its effect on the reconstruction. The effect of temperature on sketches trained with GRU

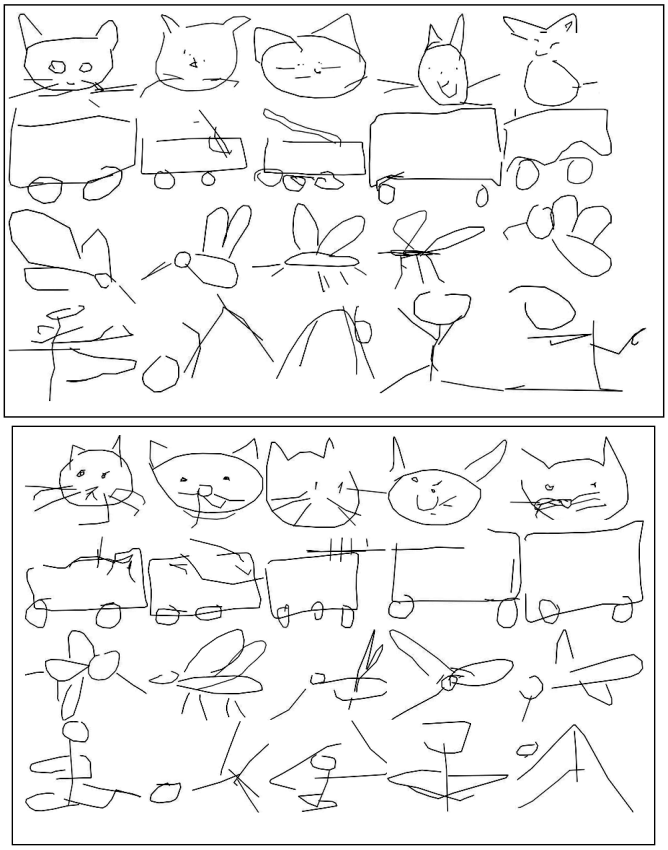

Figure 7: Sketches generated by VASkeGAN with GRU discriminator (Top) and LSTM discriminator (Bottom).

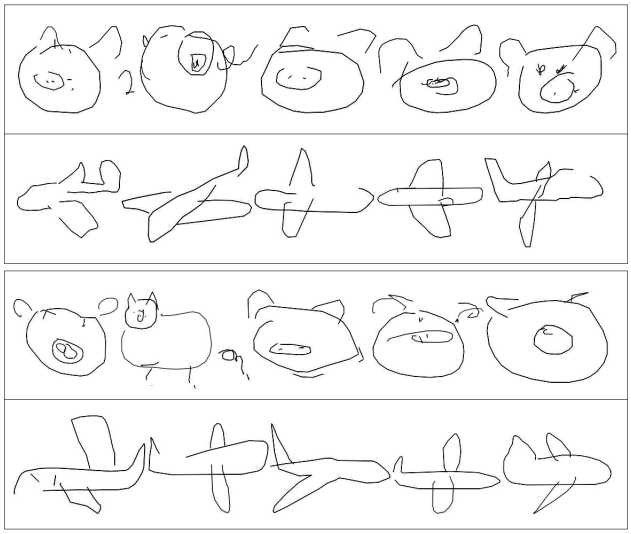

Figure 8: Transfer learning with GRU discriminator (Top) and LSTM discriminator (Bottom).

Model	Clarity	Drawing Skill	Naturalness
Dataset	2.79 ± 1.01	2.71 ± 1.07	2.6 ± 1.12
SkeGAN	**2.25 ± 1.1**	**2.25 ± 1.04**	**2.24 ± 1.21**
Sketch-RNN	2.76 ± 1.02	2.61 ± 1.03	2.5 ± 1.06
VASkeGAN(GRU)	1.8 ± 0.86	1.74 ± 0.91	1.87 ± 1.12
VASkeGAN(LSTM)	1.87 ± 0.93	1.68 ± 0.79	1.83 ± 1.04

Table 1: Human Evaluation for Visual Appeal

and LSTM discriminators is shown in Figure 9. The images to the left of the separating line in the top and bottom subfigures are the human input to the trained model. To the right of the separating line in both the subfigures are the ones that are conditionally generated by the VASkeGAN model at the temperatures of 0.2, 0.4, 0.6, 0.8 and 1.0 respectively. From the Figure 9, it can be noticed that as the temperature increases the "randomness" increases. Under the influence of τ, σ_x^2, σ_y^2 and $\hat{q}_k$ are replaced by $\sigma_x^2\tau$, $\sigma_y^2\tau$ and $\frac{\hat{q}_k}{\tau}$ respectively. Therefore, higher the value of τ more is the influence of variance which is translated into variations in the sketch generation. This is consistent with what is observered with Sketch-RNN when there is a variation in temperature τ [9].

Another observation is that the generated sketches of a particular category has the best visual appeal for a particular temperature. It is also at this temperature that along with reconstruction of sketches, extra visually appealing features (not present in the input image) are generated. For example, in Figure 9, the change in position of whiskers of cats in the top subfigure and the generation of whiskers on the cat's face in the bottom subfigure, are not present in the human input but are generated by reasoning out as to make the generated sketch more natural.

Unconditional Generation of SkeGAN: The effect of temperature τ for unconditional generation is similar to its effect in conditional generation in the case of VASkeGAN. As τ increases the randomness of the generated sketches also increases. Since we are investigating its effect on unconditional generation, there is no ground-truth to compare the randomness. Instead, a group of sketches generated with a particular τ must be compared with those generated with a different value of τ. The influence of τ on σ_x, σ_y and $\hat{q}_k$ is same for those in conditional generation of VASkeGAN. In addition to this, it influences the mixing weights of GMM by acting as its inverse multiplier as in [9]. In Figure 10, there are 5 rows each depicting the sketches generated by SkeGAN at τ values of 0.2, 0.4, 0.6, 0.8 and 1.0. We find here that τ value of 0.4 is ideal for sketch generation based on visual appeal.

Weighting Policy Gradient Loss: In order to understand the effect of policy gradient loss on the training, we multiply the loss with different weights and analyze its effect on the sketches generated. Figure 11 shows the effect of multiplying weights to policy gradient loss on the sketches generated. One can observe that the weightage given to both policy gradient loss and adversarial loss must be equal in order to generate visually appealing sketches. Therefore, the ideal weight for both adversarial loss and policy gradient loss is 1.0 respectively.

Weighting KL Divergence Loss for VASkeGAN: To understand the effect of weighting KL Divergence loss, we assign different values to w_{KL} such as 0.25, 0.5 and 1.0 and analyze the quality of

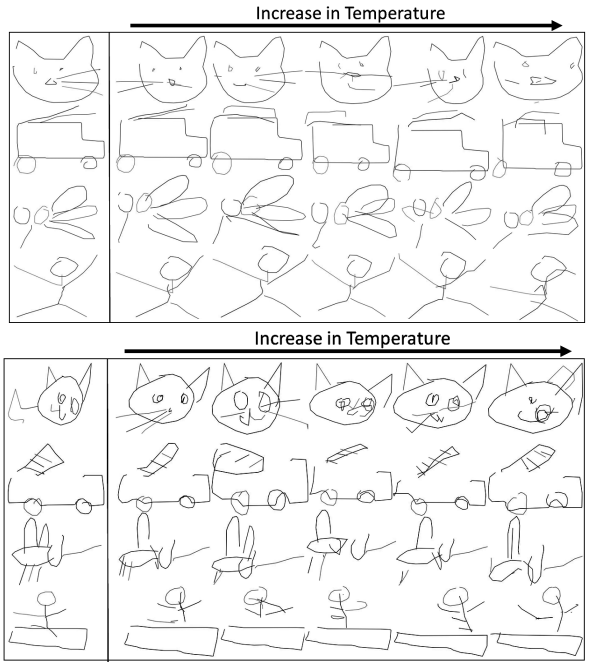

Figure 9: Effect of temperature on VASkeGAN with GRU discriminator (Top) and LSTM discriminator (Bottom).

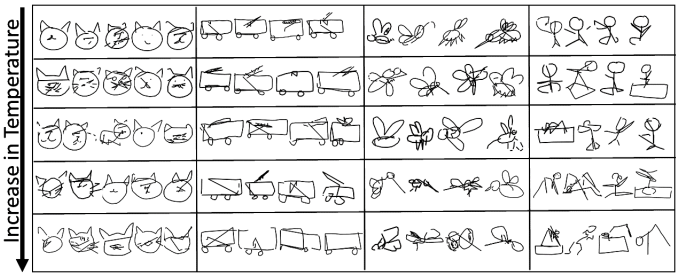

Figure 10: Effect of τ on sketches generated by SkeGAN.

sketches generated visually. Visually inspecting these sketches, we conclude that a value of 0.5 is ideal for w_{KL}. Unlike in [9], where a higher w_{KL} produces images closer to the data manifold, in our case, this behaviour is exactly opposite. Figure 12 shows the plot of L_{KL} for different values of w_{KL} while training the proposed model on cat sketches with GRU and LSTM respectively as the discriminators. The implication of this on the sketch generation is shown in Figure 13.

Training-time Comparison: Both VASkeGAN and SkeGAN are trained on *NVIDIA GeForce GTX 1080 Ti*. The average time per iteration for training VASkeGAN is 0.6s, and hence a total time of 33.33 hours to train the model (for 200000 iterations) for a given category. The average time per iteration (one iteration of generator + two iterations of discriminator) for SkeGAN is 16.95s. The total time to train SkeGAN for cat, mosquito, yoga and firetruck sketches are 13.18 hours, 9.88 hours, 19.77 hours and 13.18 hours respectively, which are much lesser than the training time of VASkeGAN

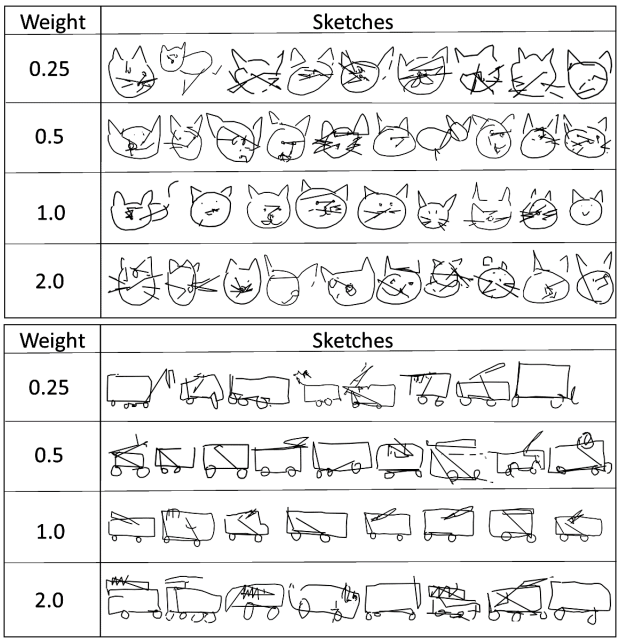

Figure 11: Effect of weighting policy gradient loss for cat sketches (Top) and firetruck sketches (Bottom).

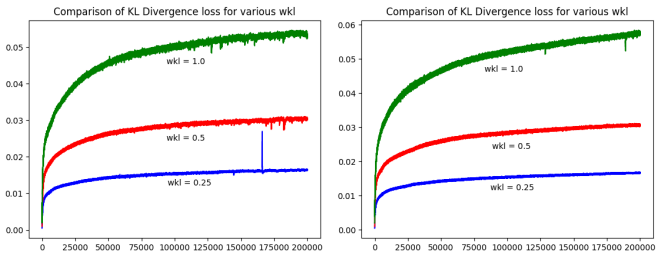

Figure 12: Plot of L_{KL} for various w_{KL} for GRU (left) and for LSTM (right) discriminators.

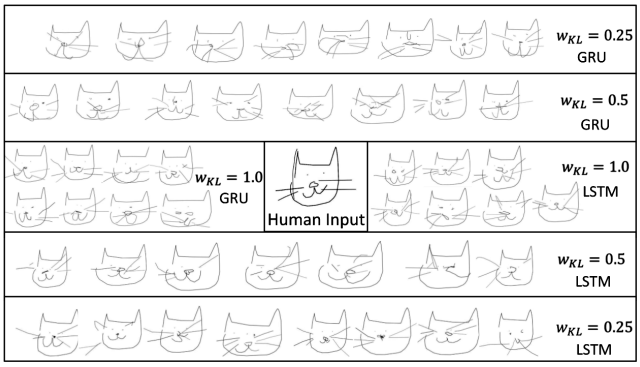

Figure 13: Effect of w_{KL} on the generated sketches at constant τ of 0.25.

(33.33 hours). SkeGAN evidently admits faster convergence than VASkeGAN. Both SkeGAN and VASkeGAN converges faster than Sketch-RNN which takes more than a million iterations to converge[2].

6 CONCLUSION

In this work, we propose two GAN-based approaches to address the problem of sketch generation in vector format. Until now, only a handful of approaches based on VAE [2, 7, 9, 33], address this problem. We propose two architectures viz. *SkeGAN* and *VASkeGAN*, the former a standalone GAN with policy gradients and adversarial loss, while the latter based on VAE-GAN. SkeGAN generates sketches that are better, both qualitatively and quantitatively, than those generated by VASkeGAN and has a faster convergence than VASkeGAN. This is attributed to the effective modelling of the sketching process and the novel combination of adversarial and policy gradient loss in SkeGAN. It also generates sketches that are at par with those from the dataset and those generated by Sketch-RNN. Most importantly, both VASkeGAN and SkeGAN minimize the scribble effect. They converge faster than Sketch-RNN. This highlights the effectiveness of GANs for this task. Future directions include generalizing this work to larger vector-art datasets, including cartoons.

REFERENCES

[1] Wengling Chen and James Hays. 2018. SketchyGAN: towards diverse and realistic sketch to image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9416–9425.

[2] Yajing Chen, Shikui Tu, Yuqi Yi, and Lei Xu. 2017. Sketch-pix2seq: a model to generate sketches of multiple categories. *arXiv preprint arXiv:1709.04121* (2017).

[3] Kenneth Forbus, Jeffrey Usher, Andrew Lovett, Kate Lockwood, and Jon Wetzel. 2011. CogSketch: Sketch understanding for cognitive science research and for education. *Topics in Cognitive Science* 3, 4 (2011), 648–666.

[4] Songwei Ge, Vedanuj Goswami, Larry Zitnick, and Devi Parikh. 2021. Creative Sketch Generation. In *International Conference on Learning Representations*. https://openreview.net/forum?id=gwnoVHIES05

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[6] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. 2015. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623* (2015).

[7] D Ha. 2015. Recurrent net dreams up fake chinese characters in vector format with tensorflow.

[8] David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106* (2016).

[9] David Ha and Douglas Eck. 2018. A Neural Representation of Sketch Drawings. In *International Conference on Learning Representations*.

[10] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. 2007. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Technical Report 07-49. University of Massachusetts, Amherst.

[11] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-Image Translation with Conditional Adversarial Networks. *CVPR* (2017).

[12] Jonas Jongejan, Henry Rowley, Takashi Kawashima, Jongmin Kim, and Nick Fox-Gieg. 2016. The quick, draw!-ai experiment.

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

[14] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. 2015. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300* (2015).

[15] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4681–4690.

[16] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

[17] Yongyi Lu, Shangzhe Wu, Yu-Wing Tai, and Chi-Keung Tang. 2018. Image Generation from Sketch Constraint Using Contextual GAN. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 205–220.

[18] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. 2016. Context Encoders: Feature Learning by Inpainting. In *Computer Vision and Pattern Recognition (CVPR)*.

[19] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[20] Ravi Kiran Sarvadevabhatla et al. 2015. Eye of the dragon: Exploring discriminatively minimalist sketch-based abstractions for object categories. In *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, 271–280.

[21] Ravi Kiran Sarvadevabhatla et al. 2016. Analyzing structural characteristics of object category representations from their semantic-part distributions. In *Proceedings of the 24th ACM international conference on Multimedia*. ACM, 97–101.

[22] Ravi Kiran Sarvadevabhatla, Isht Dwivedi, Abhijat Biswas, Sahil Manocha, et al. 2017. Sketchparse: Towards rich descriptions for poorly drawn sketches using multi-task hierarchical deep networks. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 10–18.

[23] Ravi Kiran Sarvadevabhatla, Jogendra Kundu, et al. 2016. Enabling my robot to play pictionary: Recurrent neural networks for sketch recognition. In *Proceedings of the 24th ACM international conference on Multimedia*. ACM, 247–251.

[24] Ravi Kiran Sarvadevabhatla, Sudharshan Suresh, and R Venkatesh Babu. 2017. Object category understanding via eye fixations on freehand sketches. *IEEE Transactions on Image Processing* 26, 5 (2017), 2508–2518.

[25] Ravi Kiran Sarvadevabhatla, Shiv Surya, Trisha Mittal, and R Venkatesh Babu. 2018. Game of Sketches: Deep Recurrent Models of Pictionary-Style Word Guessing. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[26] Saul Simhon and Gregory Dudek. 2004. Sketch Interpretation and Refinement Using Statistical Models.. In *Rendering Techniques*. 23–32.

[27] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

[28] Patrick Tresset and Frederic Fol Leymarie. 2013. Portrait drawing by Paul the robot. *Computers & Graphics* 37, 5 (2013), 348–363.

[29] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. 2018. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[30] Donggeun Yoo, Namil Kim, Sunggyun Park, Anthony S Paek, and In So Kweon. 2016. Pixel-level domain transfer. In *European Conference on Computer Vision*. Springer, 517–532.

[31] Jun Yu, Shengjie Shi, Fei Gao, Dacheng Tao, and Qingming Huang. 2017. Composition-Aided Face Photo-Sketch Synthesis. (2017).

[32] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[33] Kimberli Zhong. 2018. *Learning to draw vector graphics: applying generative modeling to font glyphs*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[34] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networkss. In *Computer Vision (ICCV), 2017 IEEE International Conference on*.

[2]Training details of Sketch-RNN are found here.