AI-Sketcher : A Deep Generative Model for Producing High-Quality Sketches

Nan Cao, Xin Yan, Yang Shi, Chaoran Chen
Intelligent Big Data Visualization Lab, Tongji University, Shanghai, China
{nancao, xinyan, yangshi, crchen}.idvx@gmail.com

Abstract

Sketch drawings play an important role in assisting humans in communication and creative design since ancient period. This situation has motivated the development of artificial intelligence (AI) techniques for automatically generating sketches based on user input. Sketch-RNN, a sequence-to-sequence variational autoencoder (VAE) model, was developed for this purpose and known as a state-of-the-art technique. However, it suffers from limitations, including the generation of low-quality results and its incapability to support multi-class generations. To address these issues, we introduced AI-Sketcher, a deep generative model for generating high-quality multi-class sketches. Our model improves drawing quality by employing a CNN-based autoencoder to capture the positional information of each stroke at the pixel level. It also introduces an influence layer to more precisely guide the generation of each stroke by directly referring to the training data. To support multi-class sketch generation, we provided a conditional vector that can help differentiate sketches under various classes. The proposed technique was evaluated based on two large-scale sketch datasets, and results demonstrated its power in generating high-quality sketches.

Introduction

Deep generative models are considered one of the greatest inventions in the field of AI. It has many applications, such as auto-programming (Mou et al. 2015), visual arts (Elgammal et al. 2017), and content development (Giacomello, Lanzi, and Loiacono 2018). In the past few years, various impressive results have been generated by two major types of deep generative models, i.e., the generative adversarial networks (GAN) (Goodfellow et al. 2014) and the variational autoencoder (VAE) (Kingma and Welling 2013). However, most existing studies have been designed to generate raster images (Goodfellow et al. 2014; Mirza and Osindero 2014; Radford, Metz, and Chintala 2015; Arjovsky, Chintala, and Bottou 2017) but seldom used to produce sketches in sequences of strokes. Sketch drawings play an important role in both communication and design. Drawing a sketch is considered one of human's natural behavior: in ancient times, our ancestors carved strokes on rocks to record events; at present, we draw blueprints and design drafts as sketches. These applications have motivated the need to generate sketches via AI techniques.

Previous studies in this topic have mostly focused on sketch extraction (Yu et al. 2015; Yesilbek and Sezgin 2017; Yu et al. 2017) and fine-grained sketch-based image retrieval (Yu et al. 2016; Sarvadevabhatla et al. 2017; Song et al. 2017; Chen et al. 2017). Sketch-RNN (Ha and Eck 2017) was introduced as the first AI technique that enables a computer to automatically generate simple and cursive sketch drawings based on human input. Although it is a remarkable concept, Sketch-RNN suffers from key limitations that considerably affect its application: (1) it only captures sequential orders of strokes, but fails to precisely preserve the relative position between strokes. Therefore, it typically generates low-quality results when a sketch consists of multiple parts. (2) Sketch-RNN is also incapable of dealing with multi-class situations and frequently generates incoherent sketches that integrate features from other sketch categories.

To address the aforementioned issues, in this paper, we introduce AI-Sketcher, a hybrid deep learning model that automatically generates high-quality sketch drawings by learning the sequences of strokes. It exhibits three significant improvements over the Sketch-RNN model that help overcome the aforementioned limitations: (1) To support multi-class generation, we imposed additional conditional information on both encoding and decoding processes to help differentiate sketches under various classes. (2) To capture the relative positions of strokes, we introduced an autoencoder based on a convolutional neural network (CNN) to extract spatial features from the training sketches. (3) To improve drawing quality, we provided an influence layer to enforce the effect of the encoded training data on the decoding process by considering all the previous hidden node values in the RNN encoder, thereby better guiding the generation of each stroke. We verified the performance of the proposed technique by comparing it with Sketch-RNN based on the QuickDraw dataset [1] and the FaceX dataset [2], a collection of high-quality sketches of cartoon facial expressions. Our evaluation showed that AI-Sketcher outperformed the baseline models in generating coherent sketches on both datasets.

[1] https://quickdraw.withgoogle.com/
[2] https://facex.idvxlab.com

Related Work

Among various deep generative models (Oussidi and Elhassouny 2018), variational autoencoder (Kingma and Welling 2013), i.e., VAE, is one of the most widely used techniques that is originally designed for reconstructing images. This model uses an encoder to capture the features of training data via a latent representation Z (e.g., the feature distribution of training data) and applies a decoder to reconstruct data via a sampled vector $\boldsymbol{z}$ from Z. Given that it has a simple structure and easy training process, VAE has been successfully applied to many domains, such as image reconstruction (Gulrajani et al. 2016), dialogue generation (Zhao, Zhao, and Eskenazi 2017), and molecular synthesis (Lim et al. 2018).

Sketch-RNN, a sequence-to-sequence VAE, has been recently introduced for generating sketch drawings (Ha and Eck 2017), which is highly relevant to our work. The model captures the drawing sequences of strokes within training sketches in a latent vector $\boldsymbol{z}$ via a bidirectional RNN (Schuster and Paliwal 1997) implemented based on long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) and reconstructs a stroke sequence using an autoregressive RNN (Reed et al. 2017). Since its invention, several models have been developed based on Sketch-RNN. Song et al. (Song et al. 2018) introduced a stroke-level photo-to-sketch synthesis model based on Sketch-RNN to extract sketches from images. Zhang et al. (Zhang et al. 2018) extended the technique to sketch classification.

The aforementioned work not only extends the application scope of Sketch-RNN, but also demonstrates its key limitations, such as insufficient quality of the generated sketches and incapability to generate sketches from multiple categories. AI-Sketcher addresses these limitations. It introduces an influence layer to enforce the effect of the encoded training data on the decoding process to better guide the generation of each stroke. It uses a CNN-based autoencoder to capture the relative positions of strokes within sketches, which considerably improves drawing quality. AI-Sketcher also conditions both the encoder and decoder of the model via a conditional vector to enable the model to support multi-class sketch generation. Notably, another model, called Sketch-pix2seq (Chen et al. 2017), was also designed to improve the quality of generated multi-class sketches by replacing the RNN-based encoder in Sketch-RNN with a CNN-based encoder. Compared with this model, AI-Sketcher is more controllable because the category of the generated results can be easily controlled by the conditional vector. Our experiments showed that AI-Sketcher can generate better results.

Proposed Method

In this section, we introduce the technical details of AI-Sketcher. We start with a brief review of Sketch-RNN, followed by an overview of the proposed AI-Sketcher model and its detailed design and implementation.

Sketch-RNN

Sketch-RNN is a sequence-to-sequence variational autoencoder (VAE) for generating sketches in a stroke-by-stroke manner. Similar to other VAE models, Sketch-RNN consists of two parts: the VAE encoder and the VAE decoder, as shown in Figure 1 (a).

During the training stage, the model first takes a set of sketches X_s in vector format as input. A bidirectional recurrent neural network (Schuster and Paliwal 1997) is used as the VAE encoder. It compresses X_s into a hidden node vector $\boldsymbol{h}$ (i.e., the value of the last hidden node in the RNN encoder), which can be further decomposed into two parameters, $\boldsymbol{\mu}_s$ and $\boldsymbol{\sigma}_s$, to formulate a normal distribution, $Z = N(\boldsymbol{\mu}_s, \boldsymbol{\sigma}_s)$. A latent vector $\boldsymbol{z}$ is randomly sampled from Z and used by the VAE decoder to generate the next stroke. The VAE decoder is an autoregressive RNN (Reed et al. 2017). It uses $\boldsymbol{z}$ and the last stroke $\boldsymbol{s}_i$ as inputs and produces $Y = [\boldsymbol{w}, \boldsymbol{\mu}_x, \boldsymbol{\mu}_y, \boldsymbol{\sigma}_x, \boldsymbol{\sigma}_y, \boldsymbol{\rho}_{xy}, \boldsymbol{p}]$, i.e., the parameters of a Gaussian mixture model with m normal distributions (denoted as G_M). Finally, $\boldsymbol{s}_{i+1}$ is sampled from G_M as the next generated stroke. The entire training process optimizes the following loss function:

$$L(\boldsymbol{\theta}, \boldsymbol{\phi}; X_s) = E_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|X_s)}[log p_{\boldsymbol{\theta}}(X'_s|\boldsymbol{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|X_s)||p_{\boldsymbol{\theta}}(\boldsymbol{z})) \quad (1)$$

where $q(\cdot)$ denotes the encoder, and $p(\cdot)$ denotes the decoder. $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ are the parameters to be trained in the encoder and decoder, respectively. The first term, $E_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|X_s)}(\cdot)$, is the reconstruction loss that ensures the similarity between the generated strokes and the strokes within the sketches in the training set. The second term, $D_{KL}(\cdot)$, is the KL loss that ensures the distribution of the generated strokes is similar to that of the training set.

AI-Sketcher

The design of the AI-Sketcher model extends Sketch-RNN, thereby improving drawing quality via three additional components, as shown in Figure 1(b).

First, in contrast with Sketch-RNN, which generates the next stroke $\boldsymbol{s}_{i+1}$ by considering only the stroke features captured by the RNN encoder's last hidden node value $\boldsymbol{h}$, AI-Sketcher estimates the features of all the previous strokes captured by all the hidden node values ($\boldsymbol{h}_0$, ..., $\boldsymbol{h}_t$) in the encoder, as shown in Figure 1(b-1). A fully-connected layer is introduced to estimate how the previous strokes will influence $\boldsymbol{s}_{i+1}$ in accordance with the VAE framework, based on which a latent influence vector $\boldsymbol{a}_d$ is randomly sampled and used to guide the generation of $\boldsymbol{s}_{i+1}$.

Second, motivated by conditional GAN (Mirza and Osindero 2014), a conditional vector is used in AI-Sketcher to ensure high-quality generated sketches from multiple categories. As shown in Figure. 1(b-2), we concatenated the last hidden node value $\boldsymbol{h}$, i.e., the output of the VAE encoder, with a k-dimensional one-hot conditional vector $\boldsymbol{c}$ (denoted as $\boldsymbol{h}_c$) to encode the categorical information of the input sketch data (Figure 1(b-1)). Here, k indicates the number of classes in the training set.

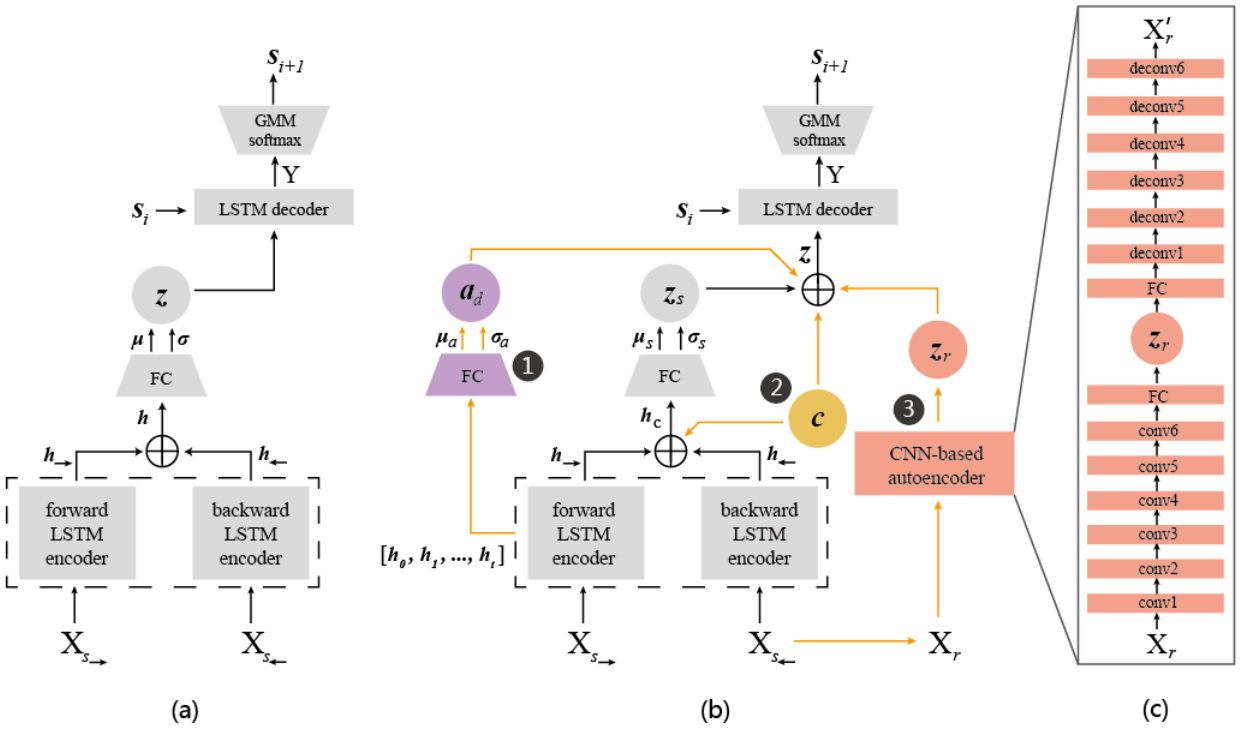


Figure 1: Schematic diagrams of (a) Sketch-RNN, (b) AI-Sketcher, and (c) the CNN-based autoencoder.

Third, a CNN-based autoencoder (Li, Qiao, and Zhang 2018) is also used in AI-Sketcher (Figure 1(b-3)). It produces a latent vector $\boldsymbol{z}_r$ that captures the spatial information of a training set X_r (the raster images transformed from X_s) at the pixel level. The latent vector $\boldsymbol{z}_r$ is particularly useful for generating sketches with multiple parts, such as a human face with nose, mouth, and eyes.

Finally, the aforementioned conditional vector $\boldsymbol{c}$, image feature vector $\boldsymbol{z}_r$, stroke feature vector $\boldsymbol{z}_s$, and influence vector $\boldsymbol{a}_d$ are concatenated into a single vector $\boldsymbol{z}$ and used as the input of the VAE decoder for the subsequent calculation.

Conditional Sequence-to-sequence VAE As shown in Figure 1(b), we introduced a conditional sequence-to-sequence VAE to AI-Sketcher, through which the other components of the proposed model are aligned. This VAE adopts sequences of strokes, i.e., X_s as input. Similar to Sketch-RNN, a bidirectional RNN is used as the VAE encoder, which encodes X_s as a hidden vector $\boldsymbol{h}^{\boldsymbol{enc}}$ (i.e., output of the RNN's last hidden node value):

$$\boldsymbol{h}^{enc} = encode(X_s) \tag{2}$$

To support multi-class generation, we introduced $\boldsymbol{h}_c = [\boldsymbol{h}^{enc}; \boldsymbol{c}]$, where $\boldsymbol{c}$ is a k-dimensional one-hot conditional vector. k indicates the number of conditions (Figure 1(b-2)). $\boldsymbol{h}_c$ is further transformed into two vectors, $\boldsymbol{\mu}_s$ and $\boldsymbol{\sigma}_s$, which are the parameters (i.e., mean values and standard deviations) of a set of normal distributions used to capture the distributions of training strokes:

$$\boldsymbol{\mu}_s = W_\mu \boldsymbol{h}_c + \boldsymbol{b}_\mu$$

$$\boldsymbol{\sigma}_s = exp(\frac{W_\sigma \boldsymbol{h}_c + \boldsymbol{b}_\sigma}{2}) \tag{3}$$

A latent vector $\boldsymbol{z}_s$ is randomly sampled from the distributions to generate the next stroke:

$$\boldsymbol{z}_s = \boldsymbol{\mu}_s + \boldsymbol{\sigma}_s \cdot \boldsymbol{\lambda} \tag{4}$$

where $\boldsymbol{\lambda}$ is a random vector sampled from the distribution $N(0, I)$ that ensures that $\boldsymbol{z}_s$ is nondeterministic.

In the next step, $\boldsymbol{z}_s$ is concatenated with the image feature vector $\boldsymbol{z}_r$, the latent influence vector $\boldsymbol{a}_d$, the conditional vector $\boldsymbol{c}$, and the last stroke vector s_i into $\boldsymbol{z} = [\boldsymbol{z}_s; \boldsymbol{z}_r; \boldsymbol{a}_d; \boldsymbol{c}; \boldsymbol{s_i}]$ for decoding:

$$\boldsymbol{h}^{dec} = decode(\boldsymbol{z}) \tag{5}$$

where $\boldsymbol{h}^{dec}$ captures the features of the previous strokes, which is further transformed into $\boldsymbol{Y}$, the parameters of a Gaussian mixture model (GMM) used to predict the next stroke. Formally, $\boldsymbol{Y}$ is calculated as follows:

$$Y = W_y \boldsymbol{h}^{dec} + \boldsymbol{b}_y \tag{6}$$

which can be decomposed in form of

$$Y = [(w_1, \boldsymbol{q}_1), ..., (w_m, \boldsymbol{q}_m), \boldsymbol{p}] \tag{7}$$

where w_i is the weight of each normal distribution in the Gaussian mixture model, and $\boldsymbol{q}_i = [\mu_{x,i}, \mu_{y,i}, \tilde{\sigma}_{x,i}, \tilde{\sigma}_{y,i}, \tilde{\rho}_{xy,i}]$ are the parameters of the 2-dimensional normal distribution of the potential x,y positions of the next point for drawing a stroke on canvas; $\boldsymbol{p} = [p_1, p_2, p_3]$ is a one-hot state vector with three fields

that indicate the status of (1) continuous drawing from the last point, (2) end of drawing a stroke, and (3) end of drawing a sketch. w_i and $p_i \in \boldsymbol{p}$ are calculated based on a softmax layer in the model. Finally, on the basis of the preceding information, we predict the probability of the relative position $p(\Delta x_{i+1}, \Delta y_{i+1})$ of the next drawing point with regard to the last drawing with the status $\boldsymbol{q}_i$ as follows:

$$p(\Delta x_{i+1}, \Delta y_{i+1}) = \sum_{i=1}^{m} w_i N(\Delta x_{i+1}, \Delta y_{i+1} | \boldsymbol{q}_i) \tag{8}$$

where N is the GMM determined by Y_i.

In our implementation, LSTM (Hochreiter and Schmidhuber 1997) with layer normalization (Ba, Kiros, and Hinton 2016) is used as both the encoder and decoder, which respectively consist of 512 and 2048 hidden nodes. The amount of GMM, which is denoted as m, is equal to 20. Our model is trained by the Adam optimizer (Kingma and Ba 2014). The learning rate of the optimizer is 0.001 and the gradient clipping is 1.0, which is used to avoid the exploding gradient problem. The batch size of the input data for each training step is set as 100.

Influence Layer We introduce a fully-connected layer, namely, the influence layer, to better guide the generation of each stroke. This layer is similar to the attention mechanism that is frequently used in RNN, but it generates the output from a latent distribution instead of directly calculating the weighted average of all the previous hidden node values. The influence layer is applied to enhance the influence of the input training data on the decoding process by considering all the previous hidden node values ($\boldsymbol{h}_0, ..., \boldsymbol{h}_t$) until the latest drawing step in the RNN encoder. As shown in Figure 1(b-1), in accordance with the VAE framework, the information of each previous stroke is captured in $\boldsymbol{h}_i$ and transformed into a normal distribution parameterized by the mean $\mu_i \in \boldsymbol{\mu}_a$ and standard deviation $\sigma_i \in \boldsymbol{\sigma}_a$. The influence vector $\boldsymbol{a}_d$ is a latent vector whose fields are sampled from the aforementioned normal distributions:

$$\boldsymbol{a}_d = \boldsymbol{\mu}_a + \boldsymbol{\sigma}_a \cdot \boldsymbol{\lambda}_a \tag{9}$$

where $\boldsymbol{\lambda}_a$ is a random vector sampled from the distribution $N(0, I)$, which ensures that the sampled vector is nondeterministic. $\boldsymbol{\mu}_a$ and $\boldsymbol{\sigma_a}$ are computed as follows:

$$\begin{aligned} \boldsymbol{\mu}_a &= \sum_{i=1}^{j} \alpha_{ij} \boldsymbol{h}_i^{enc} \\ \boldsymbol{\sigma}_a &= exp(\frac{tanh(\boldsymbol{\mu}_a)}{2}) \end{aligned} \tag{10}$$

where $\boldsymbol{h}_i^{enc}$ denotes the i_{th} hidden node values in the RNN encoder. In each decoding step j, α_{ij} indicates the weight of the i_{th} hidden node values, which is learned during the training procedure and calculated as follows:

$$\alpha_{ij} = \frac{exp(\hat{\alpha}_{ij})}{\sum_{i=1}^{j} exp(\hat{\alpha}_{ij})}, \hat{\alpha}_{ij} = \boldsymbol{h}_j^{dec} W \boldsymbol{h}_i^{enc} \tag{11}$$

where $\boldsymbol{h}_i^{enc}$ and $\boldsymbol{h}_j^{dec}$ respectively indicate the hidden node values in the encoder and decoder, and W is the weight matrix trained on the basis of a fully-connected layer.

CNN-based Autoencoder A CNN-based autoencoder, as shown in Figure 1 (b-3,c), is used to extract the latent vector $\boldsymbol{z}_r$ from the input raster image matrix X_r to capture the pixel arrangements (i.e., positional information) of the input sketches. In particular, a series of convolutional neural networks (encoder) first project an image matrix X_r onto a latent vector $\boldsymbol{z}_r$, which is later used to reconstruct X_r via a series of deconvolutional neural networks (decoder). The model minimizes the total Euclidean distance between the input image X_r and the reconstructed image $X_r^{'}$ to obtain the best $\boldsymbol{z}_r$.

In our implementation, the encoder includes three convolutional layers with the stride size as 2 and three other layers with the stride size as 1. The width and height of all the convolutional and deconvolutional kernels are set as 2. The depth of the kernels in each convolutional layers is (4, 4, 8, 8, 8, 8). The last layer in the encoder is a fully-connected neural network to produce the latent feature vector $\boldsymbol{z}_r$ with 128 dimensions, which captures the spatial information of the input data. Meanwhile, the decoder consists of three deconvolutional layers with the stride size as 2 and three other layers with the stride size as 1. The depth of the kernels in the deconvolutional layers is (8, 8, 8, 8, 4, 4). $ReLU$ (Krizhevsky, Sutskever, and Hinton 2012) is used as the activation function in the convolutional and deconvolutional layers, and $tanh$ is used as the activation function of the fully-connected neural network.

Loss Function AI-Sketcher is trained by minimizing the following loss function based on a set of sketches in vector format (i.e., X_s):

$$Loss = l_r + \alpha \cdot max(l_{kl}, \epsilon) \tag{12}$$

The first term, l_r, in the above equation is the reconstruction loss that estimates the differences between the generated strokes and the training samples. The second term estimates the distribution differences between the generated strokes and the strokes in the training set. In particular, l_r is formally defined as follows:

$$\begin{aligned} l_c &= -\frac{1}{n_{max}} \sum_{i=1}^{n_s} log(\sum_{j=1}^{m} w_{ij} N(\Delta x_i, \Delta y_i | \boldsymbol{q}_i)) \\ l_s &= -\frac{1}{n_{max}} \sum_{i=1}^{n_{max}} \sum_{k=1}^{3} p_{ki}^{enc} log(p_{ki}^{dec}) \\ l_r &= l_c + l_s \end{aligned} \tag{13}$$

where n_s is the total number of generated strokes, and n_{max} is the longest stroke length in our training set. l_c estimates the likelihood of the predicted startcoordinates of each generated stroke (i.e., $(\Delta x_i, \Delta y_i)$) under a normal distribution $N(\cdot)$ parameterized by $\boldsymbol{q}_i = [\mu_{x_i}, \mu_{y_i}, \tilde{\sigma}_{x_i}, \tilde{\sigma}_{y_i}, \tilde{\rho}_{xy_i}]$ based on GMM. $(\Delta x_i, \Delta y_i)$ is the position of the next predicted

drawing point in a stroke that is under generation. This position is relevant relative to the previous drawing point in the same stroke. l_s calculates the cross entropy between the stroke state $\boldsymbol{p}^{enc}$ of an input stroke from the training set and the stroke state $\boldsymbol{p}^{dec}$ of a generated stroke, which estimates the difference between $\boldsymbol{p}^{enc}$ and $\boldsymbol{p}^{dec}$.

The second term, l_{kl}, in Equation (12) estimates the divergence between the distributions of the generated strokes and the training data modeled by $N(0, I)$ based on the Kullback-Leibler divergence (Kullback and Leibler 1951). In particular, l_{kl} is defined as:

$$\begin{aligned} l_z &= -\frac{1}{2n_z}\sum_{i=1}^{n_z}(1+\boldsymbol{\sigma}_{s_i} - exp(\boldsymbol{\sigma}_{s_i}) - \boldsymbol{\mu}_{s_i}^2) \\ l_a &= -\frac{1}{2n_a}\sum_{j=1}^{n_a}(1+\boldsymbol{\sigma}_{a_j} - exp(\boldsymbol{\sigma}_{a_j}) - \boldsymbol{\mu}_{a_j}^2) \\ l_{kl} &= l_z + \beta l_a \end{aligned} \tag{14}$$

where l_z / l_a is the KL divergence between the distributions of the latent vector $\boldsymbol{z}_s$ / $\boldsymbol{a}_d$ (i.e., $N(\boldsymbol{\mu}_s, \boldsymbol{\sigma}_s)$ / $N(\boldsymbol{\mu}_a, \boldsymbol{\sigma}_a)$) and the strokes in the training data (i.e., $N(0, I)$). n_z and n_a indicate the dimensions of $\boldsymbol{z}_s$ and $\boldsymbol{a}_d$, respectively. β is a hyperparameter that balances the two terms (see Figure 1(b) for the notations). In our implementation, we set $n_z = 256, n_a = 512$, and $\beta = 0.1$.

To simultaneously minimize l_r and l_{kl}, additional parameters and settings are added and made in Equation (12). In particular, to avoid gradient vanishing, an annealing weight α is introduced to balance between l_r and l_{kl}, which will be gradually increased during the training process to ensure that l_r will be trained at the first moment. In addition, a lower bound of the l_{kl} loss, denoted as ϵ, is also introduced to guarantee that training will provide sufficient attention to optimizing l_r when l_{kl} is adequately small. In our implementation, we set $\epsilon = 0.20$ and the upper bound of α as 1.00.

AI-Sketcher was trained based on a Nvidia Tesla K80 graphic card. Each training step takes approximately 7.8 seconds on average. Once trained, the model supports sketch generation in real time. It takes approximately 0.013 seconds on average on an iMac machine (3.3 GHz Intel Core i5, 8 GB RAM) to produce each stroke.

Evaluation

We compared AI Sketcher with other relevant models based on two datasets: QuickDraw and FaceX. The QuickDraw dataset contains over 50 million sketches under 75 object categories, such as birds, dogs, and cars, and was originally used to train Sketch-RNN. The FaceX dataset consists of 5 million sketches of male and female facial expressions that show seven types of emotions, including anger, disgust, fear, happiness, sadness, surprise, and neutral. Compared with QuickDraw data, FaceX sketches were drawn by a group of professional designers, following strict drawing guidelines to ensure data quality.

Both datasets are in SVG format and converted into raster images saved as PNG files. Each stroke in a sketch is further transformed into a quintuple $(\Delta x, \Delta y, p_1, p_2, p_3)$, which was first introduced in (Graves 2013). Here, Δx and Δy indicate the deviation from the last drawing point, whereas $p_{1,2,3}$ are three binary status flags that respectively indicate the continuous drawing from the last point, the end of drawing a stroke, and the end of drawing a sketch. The raster images were reshaped into $128px \times 128px$ and each pixel was binarized to facilitate training.

Experiments

We performed three experiments based on the above datasets to validate AI-Sketcher's drawing quality, its capability to generate multi-class sketches, and generation diversity.

Drawing Quality In this experiment, we trained three baseline models, including a Conditional Sketch-RNN, and two alternative models by respectively removing the influence layer and the CNN-based autoencoder from the standard AI-Sketcher. The experiment was performed on the FaceX dataset to eliminate the potential negative influence of low-quality training data. During the experiment, an initial face (either female or male) was used as the input, based on which seven different facial expressions were respectively generated by AI-Sketcher and three baseline models. Figure 2 illustrates the experiment results, which suggest that AI-Sketcher produced sketches with the best quality, whereas, Sketch-RNN produced the worst, i.e., the most distorted, facial sketches. The influence layer and autoencoder helped overcome the distortion.

A within-subject user study with 20 participants (10 females) was also performed to allow users to rate the quality of 140 sketches generated with the AI-Sketcher and the baseline models using the same set of inputs. The 5-Likert scale was used, with 1 indicating "very poor" quality and 5 indicating "very good" quality. The repeated measures one way ANOVA analysis of the rating results showed that the generation quality of AI-Sketcher had an average rating of 3.9 and was significantly better than those of the baseline models (with all $p < .01$).

We also applied the t-SNE (Maaten and Hinton 2008) to verify the coherence of the latent vectors (Figure 3). For each model, 30 latent vectors of each category (shown by color) were randomly sampled and visualized. The results indicated that the latent vectors of different expressions sampled from the AI-Sketcher were more coherent, i.e., better clustered in t-SNE, compared with those of the other models. In addition, AI-Sketcher also obtained minimum overall loss and reconstruction loss. Its KL loss was also smaller than that of Sketch-RNN.

Multi-Class Generation The second experiments evaluated AI-Sketcher's performance in terms of generating different types of sketches. We trained Sketch-RNN, Sketch-pix2seq (Chen et al. 2017), and AI-Sketcher on the subsets of the QuickDraw dataset with different numbers of data categories for comparison. As shown in Figure 4 (a), AI-Sketcher generated higher-quality results, particularly when the number of classes was large. The comparison of total loss also showed that AI-Sketcher exhibited the best performance (Figure 4 (b)).

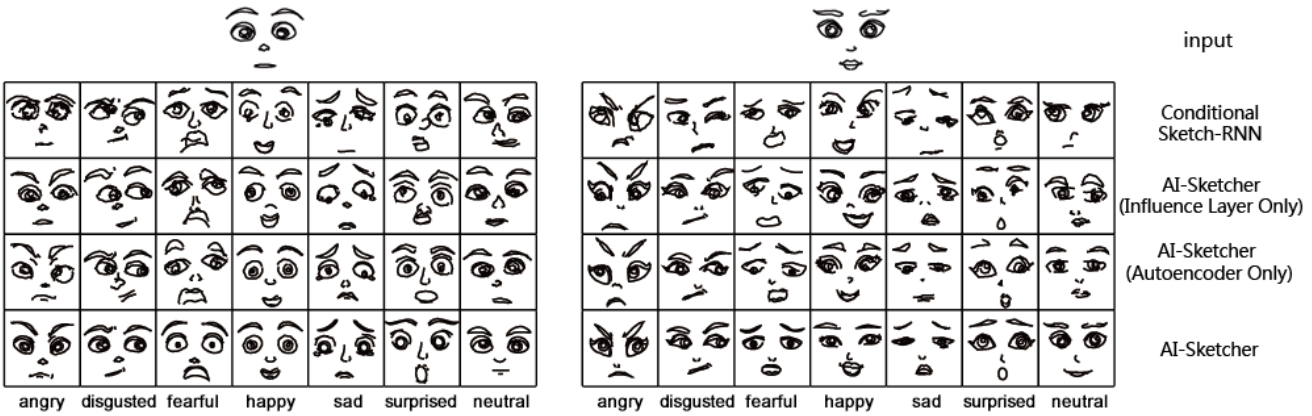


Figure 2: Generating the emotional facial expressions based on the Conditional Sketch-RNN , AI-Sketcher (Influence Layer Only), AI-Sketcher (Autoencoder Only), and AI-Sketcher (the complete version).

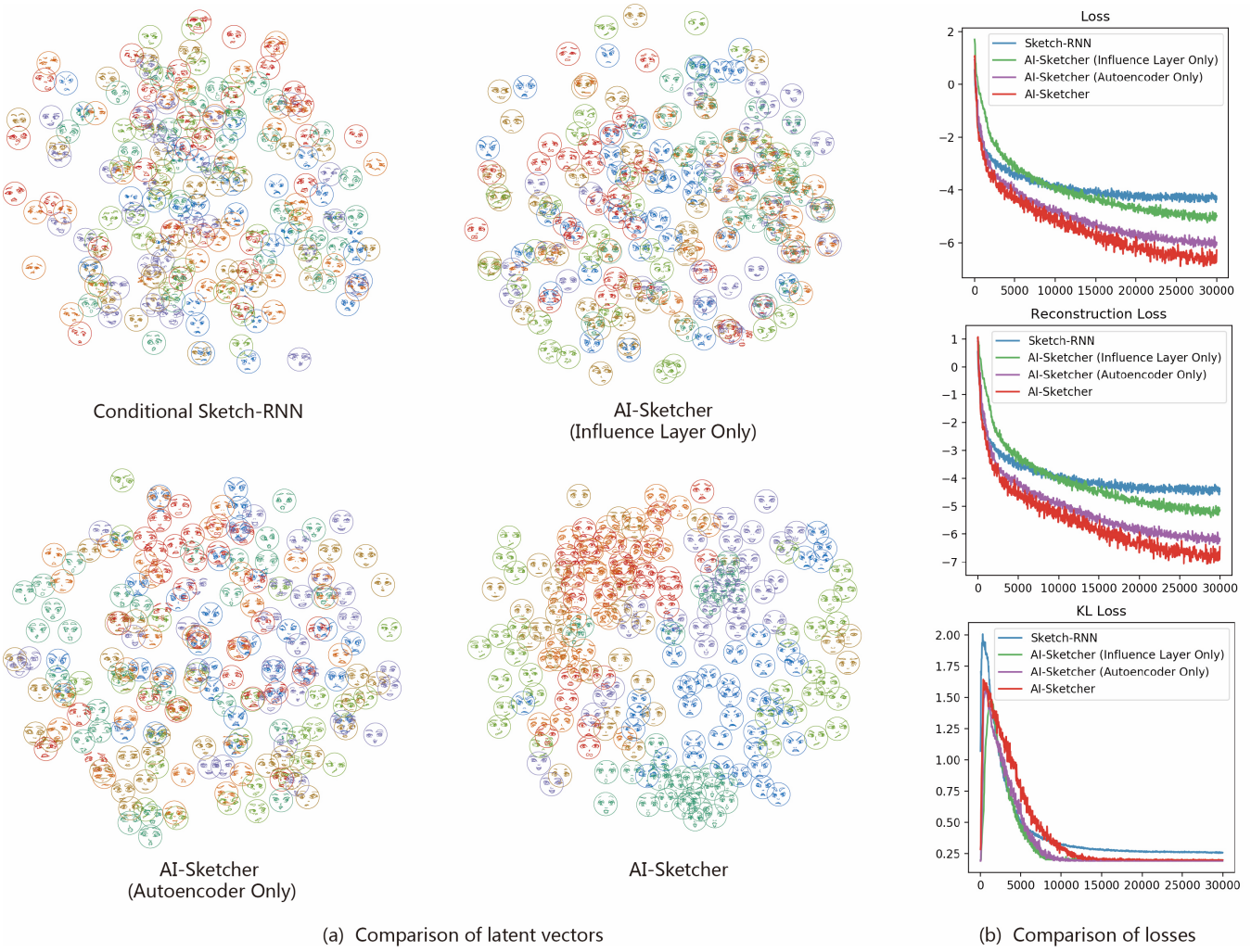
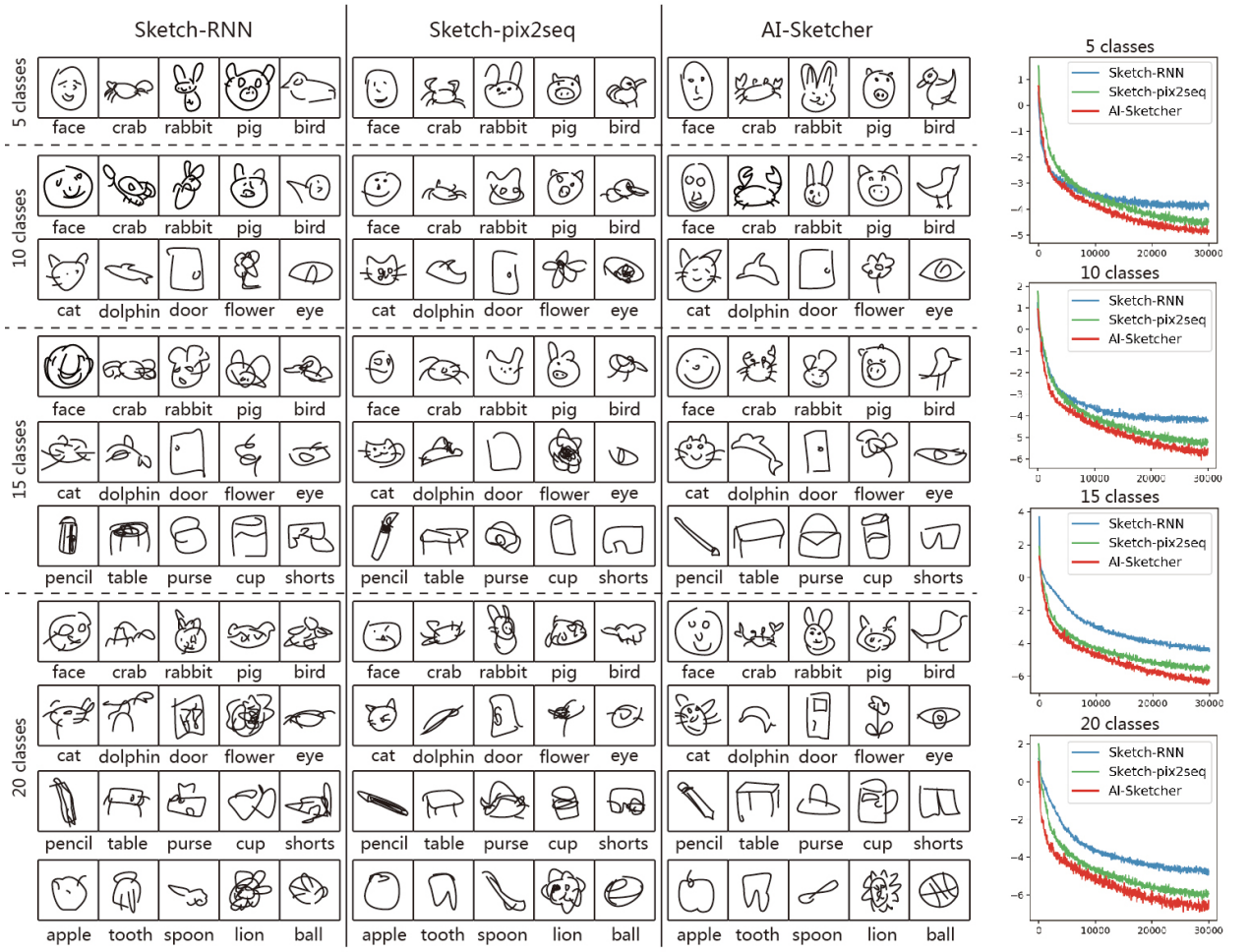


Figure 3: Comparison of (a) the distribution of latent vectors $\boldsymbol{Z}$ and (b) the overall loss (top), reconstruction loss (middle), and KL loss (bottom) among Sketch-RNN, AI-Sketcher (Influence Layer Only), AI-Sketcher (Autoencoder Only), and AI-Sketcher (full version) trained based on the facial expression dataset.



(a) Comparison of sketch generation results in different classes. (b) Comparison of losses

Figure 4: Multi-Class Sketch Generation. Comparison of Sketch-RNN, Sketch-pix2seq, and AI-Sketcher trained on a subset of the QuickDraw data respectively with 5, 10, 15, and 20 classes.

Input										
Model	●	●	●	●	●	●	●	●	●	●
Mean	30.66	30.97	29.76	29.87	28.98	28.82	29.23	29.45	29.66	30.00
SD	5.18	5.54	5.79	5.84	5.93	6.20	6.08	5.87	5.37	5.83
t(198)	-1.42		-0.53		0.65		-0.92		-1.53	
p	0.16 >.05		0.56 > .05		0.51 > .05		0.35 > .05		0.13 > .05	

● AI-Sketcher ● Sketch-RNN

Figure 5: Comparison of generation diversity.

Generation Diversity We also compared the generation diversity of AI-Sketcher and Sketch-RNN based on the QuickDraw dataset. In particular, we generated a set of 50 sketches in each of the five preselected categories respectively based on AI-Sketcher and Sketch-RNN. In each set, the pairwise distances between sketches were calculated based on the perceptual hash (Zauner 2010). A larger average distance indicates higher generation diversity. The unpaired t-test showed that AI-Sketcher and Sketch-RNN exhibited no significant difference, as shown in Figure 5.

Conclusion

This paper presents AI-Sketcher, a deep generative model for generating high quality multi-class sketches. The proposed model learns sequential and spatial information from a set of training sketches to automatically produce multi-class sketch drawings with higher quality. We evaluated our technique by comparing it with state-of-the-art models, including Sketch-RNN and Sketch-pix2seq, on two large-scale sketch datasets. The results showed that AI-Sketcher produced better results, particularly for complex sketches with multiple parts. Further work includes conducting more experiments and using the model in various applications.

Acknowledgments

We would like to thank all the users who participated in our study, all the designers who created the FaceX dataset for the project, and all the reviewers for their valuable comments.

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Chen, Y.; Tu, S.; Yi, Y.; and Xu, L. 2017. Sketch-pix2seq: a model to generate sketches of multiple categories. *arXiv preprint arXiv:1709.04121*.

Elgammal, A.; Liu, B.; Elhoseiny, M.; and Mazzone, M. 2017. Can: Creative adversarial networks, generating" art" by learning about styles and deviating from style norms. *arXiv preprint arXiv:1706.07068*.

Giacomello, E.; Lanzi, P. L.; and Loiacono, D. 2018. Doom level generation using generative adversarial networks. *arXiv preprint arXiv:1804.09154*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, 2672–2680.

Graves, A. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Gulrajani, I.; Kumar, K.; Ahmed, F.; Taiga, A. A.; Visin, F.; Vazquez, D.; and Courville, A. 2016. Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*.

Ha, D., and Eck, D. 2017. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., and Welling, M. 2013. Auto-encoding variational bayes. *CoRR* abs/1312.6114.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Kullback, S., and Leibler, R. A. 1951. On information and sufficiency. *The annals of mathematical statistics* 22(1):79–86.

Li, F.; Qiao, H.; and Zhang, B. 2018. Discriminatively boosted image clustering with fully convolutional auto-encoders. *Pattern Recognition* 83:161–173.

Lim, J.; Ryu, S.; Kim, J. W.; and Kim, W. Y. 2018. Molecular generative model based on conditional variational autoencoder for de novo molecular design. *arXiv preprint arXiv:1806.05805*.

Maaten, L. v. d., and Hinton, G. 2008. Visualizing data using t-sne. *Journal of machine learning research* 9(Nov):2579–2605.

Mirza, M., and Osindero, S. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Mou, L.; Men, R.; Li, G.; Zhang, L.; and Jin, Z. 2015. On end-to-end program generation from user intention by deep neural networks. *arXiv preprint arXiv:1510.07211*.

Oussidi, A., and Elhassouny, A. 2018. Deep generative models: Survey. In *International Conference on Intelligent Systems and Computer Vision (ISCV)*, 1–8.

Radford, A.; Metz, L.; and Chintala, S. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Reed, S.; Oord, A. v. d.; Kalchbrenner, N.; Colmenarejo, S. G.; Wang, Z.; Belov, D.; and de Freitas, N. 2017. Parallel multiscale autoregressive density estimation. *arXiv preprint arXiv:1703.03664*.

Sarvadevabhatla, R. K.; Dwivedi, I.; Biswas, A.; Manocha, S.; et al. 2017. Sketchparse: Towards rich descriptions for poorly drawn sketches using multi-task hierarchical deep networks. In *Proceedings of the 2017 ACM on Multimedia Conference*, 10–18.

Schuster, M., and Paliwal, K. K. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45(11):2673–2681.

Song, J.; Yu, Q.; Song, Y.-Z.; Xiang, T.; and Hospedales, T. M. 2017. Deep spatial-semantic attention for fine-grained sketch-based image retrieval. In *IEEE International Conference on Computer Vision*, 5552–5561.

Song, J.; Pang, K.; Song, Y.-Z.; Xiang, T.; and Hospedales, T. M. 2018. Learning to sketch with shortcut cycle consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 801–810.

Yesilbek, K. T., and Sezgin, T. M. 2017. Sketch recognition with few examples. *Computers & Graphics* 69:80–91.

Yu, Q.; Yang, Y.; Song, Y.-Z.; Xiang, T.; and Hospedales, T. 2015. Sketch-a-net that beats humans. *arXiv preprint arXiv:1501.07873*.

Yu, Q.; Liu, F.; Song, Y.-Z.; Xiang, T.; Hospedales, T. M.; and Loy, C.-C. 2016. Sketch me that shoe. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 799–807.

Yu, Q.; Yang, Y.; Liu, F.; Song, Y.-Z.; Xiang, T.; and Hospedales, T. M. 2017. Sketch-a-net: A deep neural network that beats humans. *International journal of computer vision* 122(3):411–425.

Zauner, C. 2010. Implementation and benchmarking of perceptual image hash functions.

Zhang, J.; Chen, Y.; Li, L.; Fu, H.; and Tai, C.-L. 2018. Context-based sketch classification. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch-Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*, 3.

Zhao, T.; Zhao, R.; and Eskenazi, M. 2017. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. *arXiv preprint arXiv:1703.10960*.