DOI: 10.1111/cgf.13861
Pacific Graphics 2019
C. Theobalt, J. Lee, and G. Wetzstein
(Guest Editors)

Volume 38 (2019), Number 7

FontRNN: Generating Large-scale Chinese Fonts via Recurrent Neural Network

Shusen Tang[†], Zeqing Xia[†], Zhouhui Lian[‡], Yingmin Tang, Jianguo Xiao

Institute of Computer Science and Technology, Peking University, Beijing, P.R. China
Center For Chinese Font Design and Research, Peking University, Beijing, P.R. China
{tangshusen, zeqing.xia, lianzhouhui, tangyingmin, xiaojianguo}@pku.edu.cn

Abstract

Despite the recent impressive development of deep neural networks, using deep learning based methods to generate large-scale Chinese fonts is still a rather challenging task due to the huge number of intricate Chinese glyphs, e.g., the official standard Chinese charset GB18030-2000 consists of 27,533 Chinese characters. Until now, most existing models for this task adopt Convolutional Neural Networks (CNNs) to generate bitmap images of Chinese characters due to CNN based models' remarkable success in various applications. However, CNN based models focus more on image-level features while usually ignore stroke order information when writing characters. Instead, we treat Chinese characters as sequences of points (i.e., writing trajectories) and propose to handle this task via an effective Recurrent Neural Network (RNN) model with monotonic attention mechanism, which can learn from as few as hundreds of training samples and then synthesize glyphs for remaining thousands of characters in the same style. Experimental results show that our proposed FontRNN can be used for synthesizing large-scale Chinese fonts as well as generating realistic Chinese handwritings efficiently.

CCS Concepts

• ***Computing methodologies*** → *Shape representations; Point-based models;*

1. Introduction

Techniques that focus on handling characters have attracted intensive attentions from researchers in the communities of Computer Graphics and Computer Vision, among which the following two topics are most popular: character recognition and glyph synthesis, corresponding to the two basic skills of humans: reading and writing. Thanks to the recent rapid development of Deep Neural Networks (DNNs), especially Convolutional Neural Networks (CNNs), the field of character recognition has witnessed great success [SBM80, ZZYL16, LYWW13, ZJX15], even on some recognition tasks, the reading skill of artificial intelligence (AI) has surpassed human beings [CWF*15]. Meanwhile, many researchers have made lots of attempts [Tia17, SRL*17, ZYZ*18, CZPM18] on glyph synthesis. However, endowing AI with writing capability like humans still needs to be explored further due to its complexity and diversity, especially for the generation of large-scale Chinese fonts.

As we know, characters are typically presented in the form of (bitmap) images and CNN based models have been proven to be extremely effective to process images. Thus, many CNN based methods [Tia17, SRL*17, CZPM18] have been proposed to handle the glyph synthesis task. However, existing methods that simply treat characters as images could inevitably obtain unsatisfactory synthesis results containing artifacts such as incorrect stroke connections, wrong topology, blurs, and so on. Obviously, approaches capable of processing and synthesizing characters in the sequential data format as shown in Figure 1 are potential solutions to address the above-mentioned problems.

In areas of Computer Vision and Computer Graphics, for recognition relevant tasks such as image classification [SLJ*15], face recognition [PVZ*15], object detection [RHGS15], etc, and generation relevant tasks like image style transfer [IZZE17], etc, CNN is usually the winner mainly due to its powerful capability of spatial information extraction. However, sequential data that implies abundant temporal information is not suited to be processed by CNN. Recurrent Neural Network (RNN) is more suitable to handle this form of data widely used in areas such as natural language processing [BCB14], speech processing [MGM15], etc. Here, we propose to apply RNN to learn the font style from a few writing trajectory samples for Chinese characters.

As shown in Figure 1, a Chinese character in the sequential data format typically contains a number of strokes and each stroke is made up of ordered points (data representation will be discussed in detail in Section 3.1). And our proposed FontRNN aims at generating large numbers of Chinese characters in this format auto-

† Equal contribution.
‡ Corresponding author.

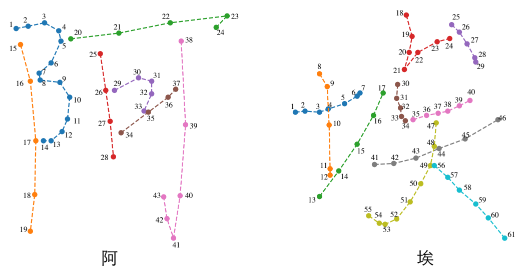


Figure 1: *Illustration of two Chinese characters in the sequential data format. A Chinese character usually consists of several ordered strokes and each stroke can be simply represented as multiple points. As shown above, each color represents a stroke and numbers represent the order of writing.*

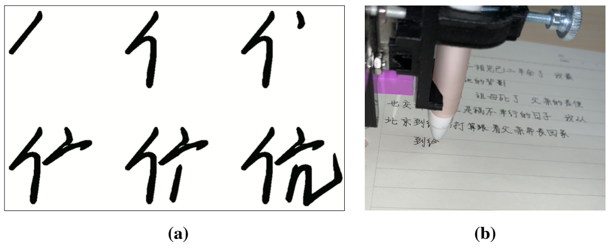


Figure 2: *Examples of the writing process of Chinese character. (a) Human beings write a Chinese character by drawing strokes one by one in correct order. (b) Synthesized sequential data by our algorithm can be utilized in a robot to write characters on paper like humans, and the video can be found in supplementary materials.*

matically. At the first step towards our goal, we use only hundreds of training samples to train the FontRNN, and each sample contains two Chinese characters: reference and target. During training, our model learns how to transform from reference characters into corresponding target characters. Afterwards, in test stage, we only feed the trained model with reference characters that have never been seen before and then predict the writing trajectories of corresponding target characters. In fact, our proposed FontRNN is a sequence-to-sequence (seq2seq) [SVL14] model for sequence style transfer task. Recently, attention mechanism has been proven to be helpful for many tasks, e.g., machine translation [BCB14, LPM15], speech recognition [BCS*16]. Therefore, we adopt an attention layer, monotonic attention [RLL*17] layer specifically, which can improve our model's performance markedly and enable the generated characters to be more readable and realistic.

Compared with the image representations, Chinese characters in the sequential data format contain more dynamic sequential information, e.g., temporal order. As shown in Figure 2, such sequential information is natural and valuable because we typically write a Chinese character by drawing strokes one by one in the predefined order instead of "rendering" an image at once and a writing robot must receive sequential character signals instead of image-like characters as input. Compared with existing methods in the literature, our FontRNN can model the temporal information of Chinese characters and write like humans. Generally speaking, our main contributions are threefold:

- We propose a RNN based model called FontRNN to generate Chinese characters which are represented as sequence-like format instead of the image format used in most existing models. Thus, our FontRNN can be more capable of reflecting the human writing process. The code of FontRNN is available at `https://github.com/ShusenTang/FontRNN`.
- Experimental results indicate that our FontRNN can be conveniently adopted for generating large-scale Chinese fonts which used to be a time-consuming and labor-intensive task. To the best of our knowledge, FontRNN is the first RNN based model which can synthesize large-scale Chinese fonts through learning from only hundreds of training samples.
- In addition, FontRNN can learn to synthesize cursive but readable handwritten Chinese characters automatically. Just like human beings, even for the same character, FontRNN generates different but reasonable handwriting at each time.

2. Related Work

In this section, we discuss the related work briefly. Recently, a number of algorithms have been proposed for the task of western (e.g., English) glyph generation, using RNNs [Gra13, APH18] as well as other approaches [CK14, BBB*18, HAU19]. In contrast to alphabetic languages such as English that contain very limited numbers of letters, Chinese has a much larger dictionary, e.g., even the most commonly used Chinese charset GB2312 consists of 6763 characters. Thereby, for example, we can't simply use one-hot encoding vectors to represent Chinese characters as we represent English characters. Basically, these two kinds of tasks have many differences and we only focus on the relevant works regarding Chinese font generation in this section.

2.1. Component Assembling based Method

There exist some previous methods (e.g., [XJJL09, ZWC11, ZZ14, PLS*14, LZX16, MTS*17]) that rely on the hierarchical representation of Chinese characters and synthesize a Chinese glyph by assembling components of radicals or strokes. For example, StrokeBank [ZZ14] decomposes every Chinese character into components, i.e., strokes or radicals, and synthesizes characters through mapping the standard font components to their handwritten counterparts. FlexiFont [PLS*14] scans and processes the camera-captured handwritten character images, and then vectorizes and normalizes these characters as personalized font libraries.

However, these component based models require a lot of prior knowledge such as elaborate preceding parsing, and thus fail to satisfactorily handle characters with complicated shapes and/or in cursive handwriting styles.

2.2. GAN based Model

Recently, generative adversarial network (GAN) [GPAM*14] which contains two nets, i.e., generator and discriminator, has been

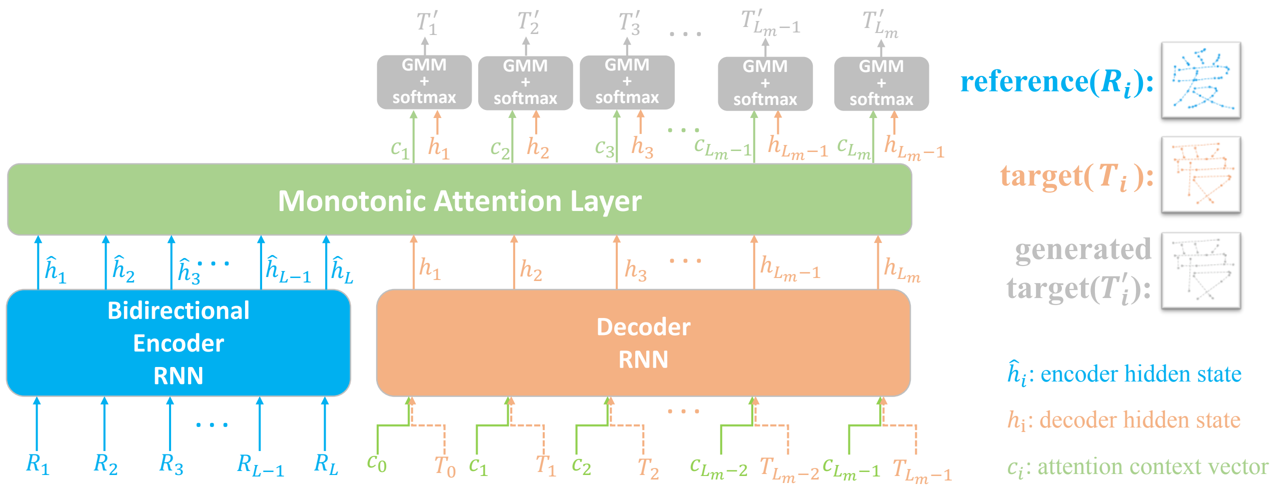

GMM + softmax
Monotonic Attention Layer
Bidirectional Encoder RNN
Decoder RNN
reference(R_i):
target(T_i):
generated target(T'_i):
$\hat{h}_i$: encoder hidden state
h_i: decoder hidden state
c_i: attention context vector

Figure 3: *An overview of our proposed FontRNN. (1) Encoder: A bidirectional RNN that takes a reference character* $[R_1, R_2, ..., R_L]$ *(L is the actual points number of this character) as input and outputs all hidden states* $[\hat{h}_1, \hat{h}_2, ..., \hat{h}_L]$*, each contains forward and backward states. (2) Monotonic Attention Layer: This layer takes encoder's output and current decoder hidden state* h_i *as input and then calculates the context vector* c_i *according monotonic attention mechanism. (3) Decoder: By passing the previous attention context vector* c_{i-1} *and target input* T_{i-1} *through the decoder, FontRNN can obtain current decoder hidden state* h_i *during the training phase. And the generated target* T'_i *can be produced by sampling from* c_i *and* h_i*. After training, unlike the training process, we feed decoder the previous generated* T'_{i-1} *as input.*

actively studied in the field of deep learning, the generator learns from real data distribution and generates realistic fake samples while the discriminator distinguishes real from generated fake samples. Thanks to the GAN's high-quality image synthesizing ability, recent GAN based models can synthesize readable Chinese characters images. [Kog15] made a preliminary attempt to generate Chinese character images using DCGAN [RMC15]. The zi2zi [Tia17] model, borrowed from pix2pix [IZZE17], adopts a style transfer method using condition GAN to achieve the goal of Chinese font generation. In zi2zi, a mapping was learned from source style characters to target style characters by training with thousands of character image pairs. After that, many GAN based methods have emerged, such as DCFont [JLTX17], HAN [CGZ17], CycleGAN based [CZPM18], etc. Unfortunately, these GAN based methods inevitably suffer from time-consuming and untamed training stage, and synthesize unreasonable or blurred results for characters with complicated structures.

Besides, researchers also exploited other methods for Chinese glyph generation. Sun et al. [SRL*17] proposed a Style-Aware Variational Auto-Encoder (SA-VAE) model to disentangle the latent features into content-related and style-related components. There exist two previous works on Chinese character generation using RNN. [Ha15] was the first attempt using RNN for generating Chinese characters but the quality of its synthesis results is poor (unreasonable and unreadable). After that, Zhang et al. [ZYZ*18] adopted RNN to recognize Chinese handwriting characters, and in order to strengthen their recognition network, they trained a generative RNN model for data augmentation. The input data of their generative model includes a char embedding c trained jointly with the model so its test character classes must have been seen in training stage and the generated results are mainly used as data augmentation for its recognition model. In addition, their generative model inevitably needs millions training samples to train the char embedding c. In contrast, our work aims at new character generation that only requires a small number of characters to generate the whole charset.

3. Method

In this section, our proposed method is described. First, we briefly introduce the representation of sequential data format. Then, the proposed FontRNN framework is depicted in detail. Finally, we describe the simple additional CNN network that is used for recovering the contour shape details on the generated character trajectories.

Generally speaking, our method implements step by step as follows. First, sequence based skeleton data (i.e., trajectory, see Section 3.1) is directly taken as input for FontRNN (see Section 3.2). Then, sequence based skeleton results synthesized by FontRNN are rendered into bitmaps. Finally, the contour shape details of skeleton bitmaps are rendered using a CNN model (see Section 3.3). The FontRNN and CNN networks are trained separately. For different target fonts, each network needs to be trained from scratch.

3.1. Data Representation

As shown in Figure 1, Chinese characters can be simply regarded as sequences of points. As suggested by [Gra13] and sketch-rnn [HE17], we use a data format that represents a character as a list of points and each point is a vector which contains 5 elements: $(\Delta x, \Delta y, p_1, p_2, p_3)$. Namely, a Chinese character with L points can be represented as

$$[P_1,...,P_L], \text{where } P_i = (\Delta x_i, \Delta y_i, p_{1i}, p_{2i}, p_{3i}). \tag{1}$$

We add some padding elements to ensure that each character contains the same number of points. Consequently, every character in the font library can be denoted as

$$[P_1,...,P_{Lm}], \tag{2}$$

where Lm is the points' number of the character that contains maximum points. $(\Delta x, \Delta y) \in \mathbf{R}^2$ is the offset distance in the x and y directions of the pen (i.e., current point) from the previous point, particularly, $(\Delta x_1, \Delta y_1)$ is the absolute coordinate of the first point P_1 and $(\Delta x_i, \Delta y_i)$ is $(0,0)$ if $i > L$. The (p_1, p_2, p_3) represents a binary one-hot vector of three possible point categories. The first point category, p_1, indicates that the pen is currently touching the paper, so a line will be drawn to connect this point with the next point. On the contrary, the second point category, p_2, denotes the end of one stroke and no line will be drawn next. The final point category, p_3, indicates that the writing has ended, so this point and subsequent points will not be rendered, thus $p_{3i} = 0$ if and only if $i > L$.

In the next part of this paper, we use R_i and T_i to represent reference and target character points respectively, and T_i' is the generated target character point.

3.2. Generating Chinese Characters via RNN

As illustrated in Figure 3, our proposed FontRNN contains three parts: an encoder, a decoder and a monotonic attention layer. Next we'll describe these three parts in detail.

3.2.1. Encoder

The encoder is a bidirectional RNN built by combining forward and backward RNN which contains powerful capability of feature extraction. Because of only using hundreds of training samples, we do not use the multi-layer bidirectional RNN but a single-layer one to avoid over fitting. During both training and testing processes, our encoder takes a reference character which is represented by $[R_1,...,R_L]$ as input, where L is the actual number of points of the input character, and then produces L hidden states $[\hat{h}_1, \hat{h}_2, ..., \hat{h}_L]$. We can treat these hidden states as this Chinese character's content information which guides the decoder to generate correct character.

3.2.2. Monotonic Attention Layer

According to a national standard of Chinese character, GF3002-1999, with regard to the same character in different font styles, both the number of strokes and the order of strokes should be exactly identical (It should be noted that the number of points in the same stroke of a character written in different styles can be different.). That is to say, although we can write a Chinese character in different writing styles, the order of writing strokes is typically fixed. Namely, the alignment between reference and target characters is monotonic. For example, there is no doubt that the appearance of the target stroke corresponding to the first stroke of reference character should precede the appearance corresponding to the second stroke of reference character. Accordingly, FontRNN utilizes a monotonic attention [RLL*17] layer to make decoding at each timestep more focused. Experiments in section 4.3.2 demonstrate the importance of this module.

As usual, we refer to the output $[\hat{h}_1, \hat{h}_2, ..., \hat{h}_L]$ as "memory", the monotonic attention layer processes the memory in a left-to-right way: for decode timestep i the attention mechanism begins processing memory entries starting at index t_{i-1}, where t_i is the index of the memory entry chosen at decode timestep i. It then computes an energy scalar $e_{i,j}$ for $j = t_{i-1}, t_{i-1}+1, ...$ and uses a logistic sigmoid function $\sigma(\cdot)$ to transform these energy scalars into probabilities $p_{i,j}$. A Bernoulli distribution parameterized by $p_{i,j}$ then "chooses" a memory entry as the context vector c_i. The above process can be formulated as

$$e_{i,j} = \text{Energy}(h_{i-1}, \hat{h}_j) \tag{3}$$

$$p_{i,j} = \sigma(e_{i,j}) \tag{4}$$

$$z_{i,j} \sim \text{Bernoulli}(p_{i,j}), \tag{5}$$

where $\text{Energy}(h_{i-1}, \hat{h}_j)$ is a function that measures how well h_{i-1} and $\hat{h}_j$ match, we can define $\text{Energy}(h_{i-1}, \hat{h}_j)$ as (6) or (7) referring to [BCB14](Bahdanau Attention) and [LPM15](Luong Attention) respectively.

$$\text{Energy}(h_{i-1}, \hat{h}_j) = v_a^T \tanh(\mathbf{W}_a h_{i-1} + \mathbf{U}_a \hat{h}_j) \tag{6}$$

$$\text{Energy}(h_{i-1}, \hat{h}_j) = h_{i-1} \mathbf{W}_a \hat{h}_j. \tag{7}$$

And each $z_{i,j}$ in (5) can be seen as a discrete choice of whether picking $\hat{h}_j$ or not. As soon as $z_{i,j} = 1$ for some j, the attention layer stops and sets $t_i = j$ and $c_i = \hat{h}_j$. Note that the above computation involves sampling, thus the model can not be trained via backpropagation. As suggested in [RLL*17], we adopt the soft monotonic attention and train model with respect to the expected value of c_i.

3.2.3. Decoder

We use a RNN decoder which takes the previous attention output c_{i-1} and target input T_{i-1} (particularly, T_0 is defined as (0, 0, 1, 0, 0)) as input, and produces a hidden state h_i at each decode timestep.

Motivated by [Gra13], we model the offset $(\Delta x, \Delta y)$ as a Gaussian mixture model (GMM) with M bivariate normal distributions and the point category (p_1, p_2, p_3) as a 3-category classification task. Therefore, the final output of decoder contains parameters of GMM and categorical distribution, i.e.,

$$\{\{\hat{\pi}^m, \mu_x^m, \mu_y^m, \hat{\delta}_x^m, \hat{\delta}_y^m, \hat{\rho}_{xy}^m\}_{m=1}^M, \hat{q}_1, \hat{q}_2, \hat{q}_3\}, \tag{8}$$

where the superscript m above stands for the mth bivariate normal distribution in GMM. In order to ensure the validity of above parameters, as suggested in [Gra13], we apply the following mathematical operations

$$\pi^m = \frac{\exp(\hat{\pi}^m)}{\sum_{k=1}^M \exp(\hat{\pi}^k)} \Rightarrow \pi^m \in (0,1), \sum_{m=1}^M \pi^m = 1 \tag{9}$$

$$\delta_x^m = \exp(\hat{\delta}_x^m), \delta_y^m = \exp(\hat{\delta}_y^m) \Rightarrow \delta_x^m, \delta_y^m > 0 \tag{10}$$

$$\rho_{xy}^m = \tanh(\hat{\rho}_{xy}^m) \Rightarrow \rho_{xy}^m \in (-1,1) \tag{11}$$

$$q_j = \frac{\exp(\hat{q}_j)}{\sum_{k=1}^{3}\exp(\hat{q}_k)} \Rightarrow q_1, q_2, q_3 \in (0,1), \sum_{k=1}^{3} q_k = 1. \tag{12}$$

With regard to the m^{th} bivariate normal distribution in GMM, π^m denotes the distribution weight, μ_x^m and μ_y^m denote the means, δ_x^m and δ_y^m are the standard deviations, and ρ_{xy}^m is the correlation parameter. Therefore, the probability density of $(\Delta x, \Delta y)$ is

$$p(\Delta x, \Delta y) = \sum_{m=1}^{M} \pi^m \mathcal{N}(\Delta x, \Delta y | \mu_x^m, \mu_y^m, \delta_x^m, \delta_y^m, \rho_{xy}^m), \tag{13}$$

where $\mathcal{N}$ is the probability distribution function for a bivariate normal distribution

$$\mathcal{N}(x, y | \mu_x, \mu_y, \delta_x, \delta_y, \rho) = \frac{\exp[\frac{-Z}{2(1-\rho^2)}]}{2\pi\delta x \delta y \sqrt{1-\rho^2}} \tag{14}$$

$$Z = \frac{(x-\mu_x)^2}{\delta_x^2} + \frac{(y-\mu_y)^2}{\delta_y^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\delta_x \delta_y}. \tag{15}$$

To train our model, we should define the loss function. Intuitively, our training loss consists of two parts: negative log-likehood of $(\Delta x, \Delta y)$ and the cross entropy of (p_1, p_2, p_3), they can be formulated respectively as

$$\text{Loss}_d = -\frac{1}{L}\sum_{i=1}^{L} \log(\sum_{m=1}^{M} \mathcal{P}^m(\Delta x_i, \Delta y_i)) \tag{16}$$

$$\text{Loss}_c = -\frac{1}{L_m}\sum_{i=1}^{L_m}\sum_{k=1}^{3} p_{k,i} log(q_{k,i}), \tag{17}$$

where

$$\mathcal{P}^m(\Delta x_i, \Delta y_i) = \pi^m \mathcal{N}(\Delta x_i, \Delta y_i | \mu_{x_i}^m, \mu_{y_i}^m, \delta_{x_i}^m, \delta_{y_i}^m, \rho_{xy_i}^m)), \tag{18}$$

L is the actual points' number of current character, while L_m is the maximum of all Ls in the font. As suggested in [HE17], we discard the points beyond L when calculating Loss_d, while Loss_p is calculated using all points until L_m. It should be noted that we normalize Loss_d by dividing it by L instead of L_m used in [HE17], because for some characters that contain a few points the Loss_d will always be extremely close to zero if we divide it by L_m.

The total loss function is defined as the weighted summation of Loss_d and Loss_c

$$\text{Loss} = \text{Loss}_d + \lambda_c \text{Loss}_c, \tag{19}$$

where the hyper parameter λ_c is the weight of Loss_c. Then we can optimize Loss for training our model.

After training, the decoder generates the parameters of GMM and categorical distributions at each timestep and samples an output T_i' for that timestep, and then we feed decoder T_i' as the input of next timestep. This process is repeated until the sampled $p_{3,i}' = 1$ or $i = L_m$. When sampling from m^{th} bivariate normal distribution, we can use the means (μ_x^m, μ_y^m) greedily as the determinate output (e.g., fonts generation) as well as sampling randomly for the indeterminate output (e.g., handwriting generation).

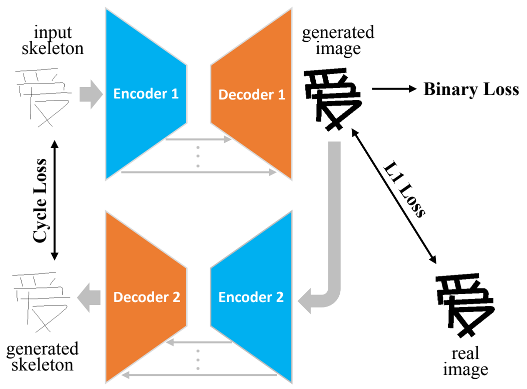


Figure 4: *Illustration of our model for shape synthesis. All of the images have pixel values from 0 to 1.*

3.3. Synthesizing Shape Details

In order to produce complete practicable font libraries, we need to recover the contour shape of characters (i.e., skeletons/trajectories) generated by FontRNN. Consequently, as shown in Figure 4, we utilize a simple additional CNN based network that contains two modified U-Net [RFB15] structures, one for shape synthesizing and the other for extracting skeleton images from the former step to refine the shape generated above in a circular way. Specifically, each encoder (decoder) contains 9 layers and each one consists of convolution (or de-convolution for decoder) layer, batch normalization [IS15] and leaky relu (or relu for decoder) layer.

Besides the L1 loss, to avoid blurred results that often occur in other models (e.g., pix2pix [IZZE17]), we adopt a pixel-wise quadratic loss to force model to generate images with clear details

$$\text{Loss}_{\text{binary}} = \frac{1}{N}\sum_{i=1}^{N} x_i^2(1-x_i^2), \tag{20}$$

where N is the number of pixels in one image. $\text{Loss}_{\text{binary}}$ can make the output pixel values close to 0 or 1.

In addition, the cycle loss between input skeleton and generated skeleton is also computed using the L1 norm, thus the total loss is combined with three parts

$$\text{Loss} = \text{Loss}_{\text{L1}} + \alpha\text{Loss}_{\text{binary}} + \beta\text{Loss}_{\text{cycle}}, \tag{21}$$

where we simply set β to 1, but α grows every epoch from 0.01 based on the following equation

$$\alpha = 0.01 + 0.99\alpha. \tag{22}$$

Finally, the error backpropagation algorithm is utilized to optimize the model.

4. Experiments

In this section, we first describe the details of our experiments including datasets and hyper-parameters settings. After that, we show experimental results in three parts: Chinese character generation via RNN, shape synthesis and handwriting generation.

4.1. Datasets

We conduct our experiments on Chinese font libraries in section 4.3 and 4.4, as well as Chinese handwriting dataset in section 4.5. Specifically, in section 4.3 and 4.4, we choose various font libraries (each contains 6763 Chinese characters) in different styles and extract the skeletons of strokes from character images in font libraries. We use an automatic method proposed in [LZCX18] to extract strokes, followed with a few manual adjustments when it fails (if necessary). The stroke order is defined exactly as the national standard GF3002-1999. Then we use the character set called optimal set proposed in [LZCX18] as training set, which contains only 775 characters and covers nearly all kinds of components in the font with 6763 Chinese characters, and we render all characters to 300×300 bitmap images for section 4.4.

In section 4.5, handwriting trajectories we used are from CASIA [LYWW11] which were written cursively by humans. We increase the size of the training set to 2000 because styles in handwritings are more difficult to learn compared to those in fonts.

In all our experiments, we collect a set of Chinese fonts and calculate the average writing trajectory for each character in these fonts, which is used as our input reference trajectory. In addition, to make FontRNN focus more on learning from key points, we adopt the Ramer-Douglas-Peucker algorithm [DP73] on every sequential character to remove redundant points.

4.2. Implementation Details

In this section, we introduce the implementation details of our models which include FontRNN and shape synthesis model described in section 3.2 and 3.3, respectively.

4.2.1. FontRNN

We implement RNN encoder and decoder of our FontRNN using Long Short-Term Memory (LSTM) [HS97] with 256 neurons on account of its great capability of learning long-term dependencies. When training FontRNN with batch size of 128, we use an Adam optimizer with gradient clipping of 1.0. To alleviate over fitting, we apply recurrent dropout [SSB16] with a keep probability of 0.6, and, in addition, we perform data augmentation by multiplying the $(\Delta x, \Delta y)$ by a random scale factor chosen uniformly between 0.90 and 1.10 and dropping some points randomly with a chance of 0.10. We set the weight of Loss$_c$ λ_c to 2.0 and M, the normal distribution number of GMM, to 20.

In order to speed up the FontRNN training at the beginning of training while ensuring final convergence, we utilize a learning rate decay strategy, the learning rate at training step i is computed as

$$\mathrm{Lr}_i = (\mathrm{Lr}_{\max} - \mathrm{Lr}_{\min}) * \mu^i + \mathrm{Lr}_{\min}, \tag{23}$$

where $\mu = 0.9999$ is the decay rate, $\mathrm{Lr}_{\max} = 0.001$ and $\mathrm{Lr}_{\min} = 0.00001$ are the maximum and minimum learning rates, respectively.

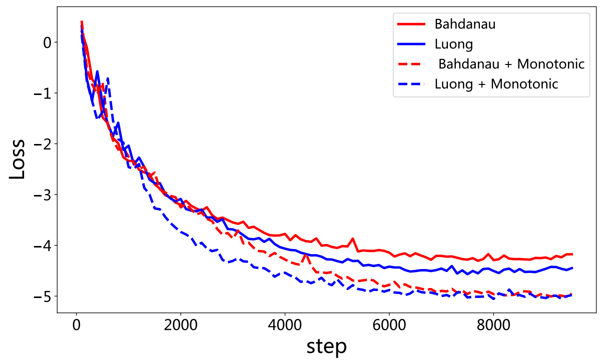


Figure 5: *Comparison of different attention methods.*

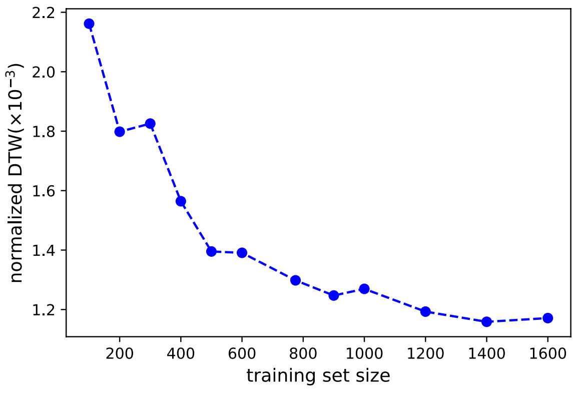


Figure 6: *Comparison of different training set sizes.*

4.2.2. Model for Shape Synthesis

The convolution and deconvlution layers in Figure 4 all use 4-by-4 filters with 2-by-2 strides. The α value of leaky relu layer is 0.2. The last layer of decoder actually use $0.5(\tanh(x)+1)$ for activation function to limit output value between $[0,1]$. We also utilize dropout with keeping probability 0.5 in decoder, and we still use an Adam Optimizer with learning rate 0.001 and batch size 10 to train our model for about 130 epochs.

4.3. Chinese Character Generation via RNN

4.3.1. Evaluation Metric

In order to quantitatively evaluate our FontRNN, an evaluation metric needs to be determined. The most straightforward similarity measure for sequences is Euclidean Distance and its variants, e.g.,

Table 1: *Comparison of performance of models using different attention methods. Each column represents one font while each row represents one attention mechanism. Each number* ($\times 10^{-3}$) *represents the average normalized DTW over testing set.*

	FZJHSXJW	FZSSBJW	FZTLJW	FZYNJW	FZZJ-LPYBJW	FZZJ-GBWKJW	Average
No attention	2.7870	2.6619	4.0960	3.2481	2.7409	2.6711	3.0342
Bahdanau	1.5140	1.5460	3.4492	1.7024	1.7456	1.3617	1.8865
Luong	**1.4393**	1.8813	2.3113	1.8310	1.9469	1.7388	1.8581
monotonic Bahdanau	1.4429	1.4451	2.0546	1.3814	1.7475	1.4158	1.5812
monotonic Luong	**1.4393**	**1.2990**	**2.0128**	**1.2982**	**1.7244**	**1.3762**	**1.5250**

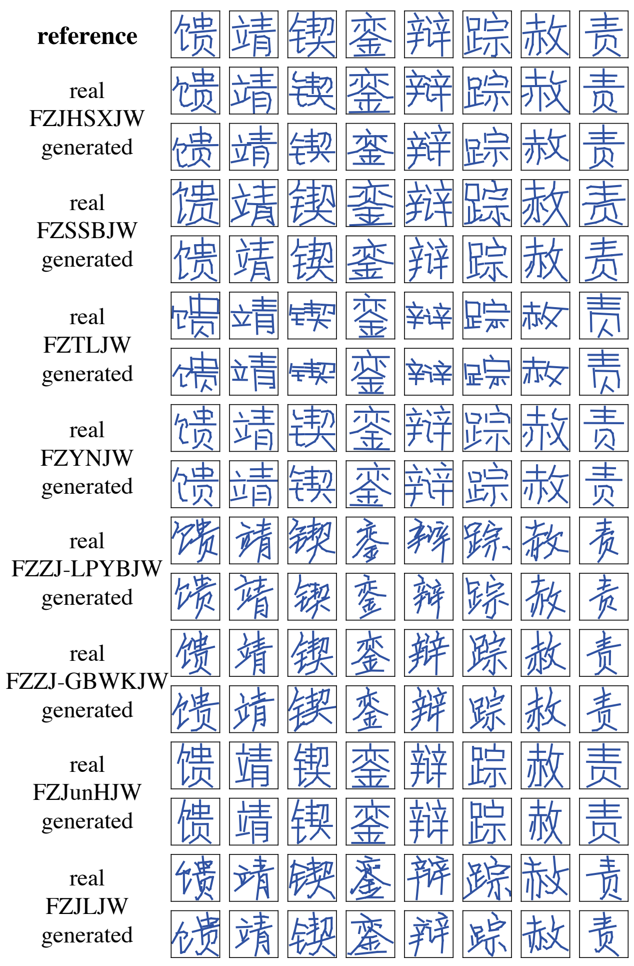


Figure 7: *Illustration of generated skeletons of different fonts by FontRNN. More results can be found in supplementary materials.*

L_1 (Manhattan) and L_2 (Euclidean) distance. However, since the mapping between the points of generated and target sequences is fixed, these distance measures are very sensitive to misalignments and noise. Dynamic time warping (DTW) was proposed [BC94] to solve the above problem of misalignments by ignoring local shifting of sequence. In general, DTW calculates an optimal match between two given sequences with certain restrictions and the optimal match has the minimal cost, where the cost is computed as the sum of distance (e.g., we use L_2 distance) between each matched pair of points. More details about DTW can be found in [BC94].

In our case, we use the DTW distance to evaluate the accuracy of coordinate prediction. We first convert the target and the predicted relative coordinates into corresponding absolute coordinates C and C'

$$\begin{aligned} C &= [(x_1,y_1),(x_2,y_2),...,(x_{|C|},y_{|C|})] \\ C' &= [(x'_1,y'_1),(x'_2,y'_2),...,(x'_{|C'|},y'_{|C'|})], \end{aligned} \tag{24}$$

where $|C|$ and $|C'|$ are the lengths of C and C', respectively. Since the sequence length $|C|$ is different for various characters and DTW cost is computed as the sum of distance between each matched pair of points, we divide the DTW result by $|C|$ to eliminate the effects of different lengths. Meanwhile, we normalized the DTW result by character spatial size for more robust evaluation metric. Namely, the normalized DTW we used for evaluation is computed as

$$\mathrm{nDTW}(C,C') = \frac{\mathrm{DTW}(C,C')}{|C|\sqrt{(x_{max}-x_{min})^2+(y_{max}-y_{min})^2}}, \tag{25}$$

where

$$\begin{aligned} x_{max} &= \max_{i=1}^{|C|} x_i, \quad x_{min} = \min_{i=1}^{|C|} x_i \\ y_{max} &= \max_{i=1}^{|C|} y_i, \quad y_{min} = \min_{i=1}^{|C|} y_i. \end{aligned} \tag{26}$$

4.3.2. Comparison of Attention Methods

As described in section 3.2.2, the monotonic attention layer is critical to our FontRNN. Here we conduct a series of experiments on different kinds of Chinese fonts (e.g., FZJHSXJW, FZSSBJW, FZTLJW) to prove this. For each font, we conduct five experiments on our FontRNN with no attention, Bahdanau attention [BCB14], Luong attention [LPM15], monotonic [RLL*17] Bahdanau attention and monotonic Luong attention, respectively. The average $\mathrm{nDTW}(C,C')$ over testing set of each experiment is shown in Table 1. As we can see from Table 1, the results of methods using attention are much better than those without attention. As we expected and analyzed in section 3.2.2, the monotonic mechanism can further improve the performance of FontRNN.

Furthermore, we also compare the training processes of FontRNN with different attention methods (Figure 5). According to

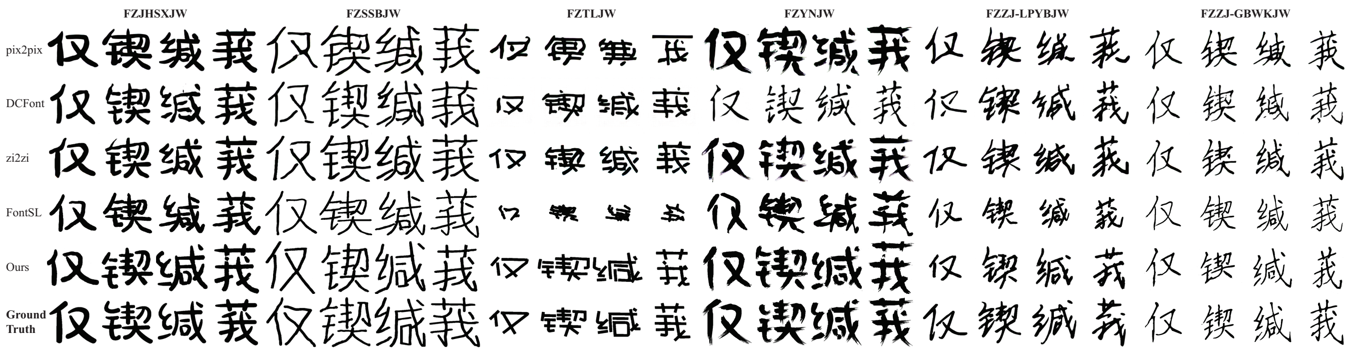


Figure 8: *Illustration of six Chinese fonts with different styles synthesized by our method and other related methods. More results can be found in supplementary materials.*

Table 2: *The characters recognition accuracy on six fonts generated by different methods.*

	FZJHSXJW	FZSSBJW	FZTLJW	FZYNJW	FZZJ-LPYBJW	FZZJ-GBWKJW	Average
pix2pix	0.9029	0.8039	0.1521	0.8583	0.6695	0.7553	0.6903
DCFont	0.9431	0.6590	0.3215	0.4683	0.8118	0.6289	0.6388
zi2zi	0.9435	0.9620	0.8005	0.9486	0.9042	0.8690	0.9046
FontSL	0.9556	**0.9833**	0.2164	0.9290	0.8849	0.9250	0.8157
Ours	**0.9669**	0.9801	**0.8639**	**0.9765**	**0.9016**	**0.9374**	**0.9377**
GroudTruth	0.9968	0.9846	0.9231	0.9950	0.9252	0.9820	0.9678

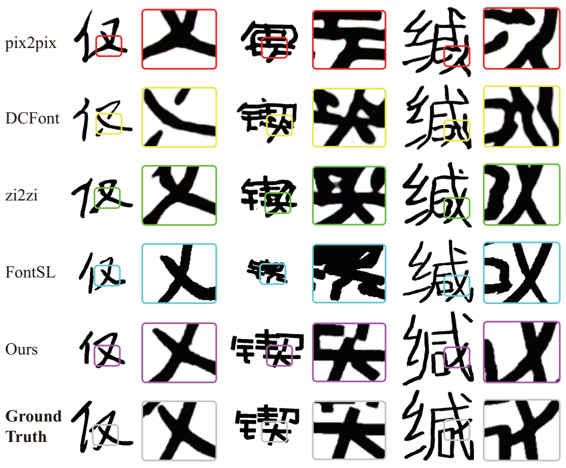


Figure 9: *Illustration of details at the intersections of strokes.*

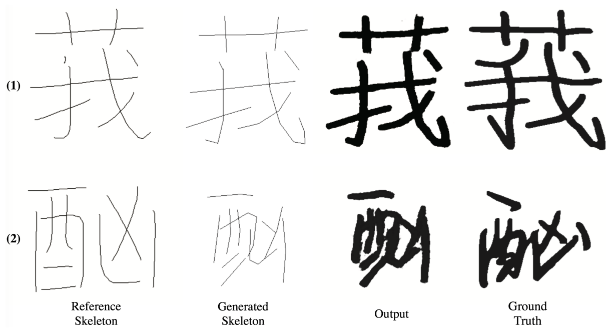
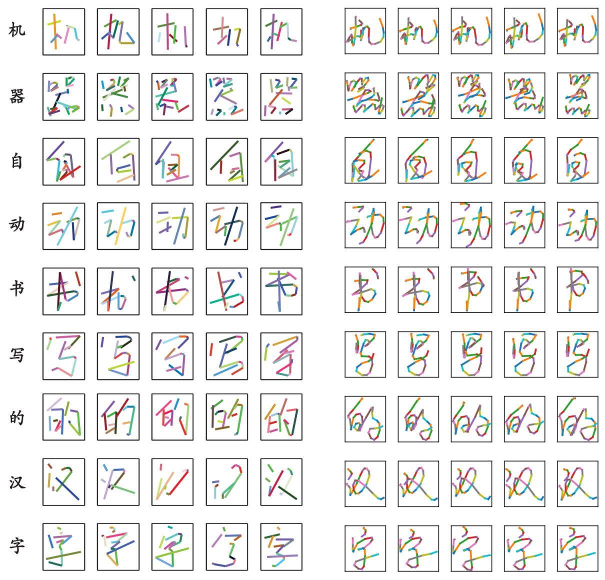


Figure 10: *Illustration of two types of failure cases.*

Table 1 and Figure 5, we can conclude that monotonic attention always outperforms normal attention and FontRNN with monotonic Luong attention converges faster during training and often gets better results than that with monotonic Bahdanau attention. Thus we adopt monotonic Luong attention in all following experiments.

4.3.3. Comparison of Training Set Sizes

In order to analyze the robustness of FontRNN, we evaluate the performance with a series of training sets with different sizes that contain 100, 200, 300, 400, 500, 600, 775, 900, 1000, 1200, 1400 and 1600 samples, respectively. The set with 775 samples is the so-called optimal set [LZCX18] and elements in the sets with less than 775 samples are randomly selected from the optimal set, while the sets with more than 775 samples consist of the optimal set and other randomly selected samples. As shown in Figure 6, the performance of FontRNN improves overall as the training set size increases but the improvement is not noticeable when the training set size exceeds about one thousand. That is, FontRNN only needs less



(a) Zhang et al. **(b) FontRNN**

Figure 11: *Comparison of the Chinese handwriting generation using our FontRNN and the method in [ZYZ*18]. Borrowed from [ZYZ*18], each color (randomly selected) denotes one straight line.*

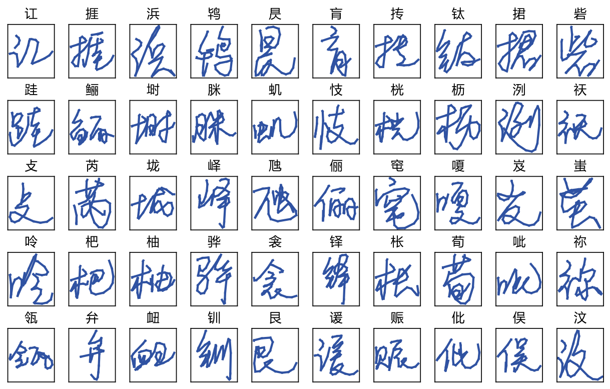

讧 捱 浜 鸨 昃 肓 抟 钛 捃 呰
跬 鳊 埘 脒 虮 忮 桄 枥 洌 祆
攴 芮 垅 峄 飑 俪 窀 嗄 岌 蚩
呤 杷 柚 骅 衾 铎 枨 荀 呲 祢
瓴 弁 岫 钏 艮 谖 赈 仳 俣 汶

Figure 12: *Illustration of generated handwriting results in test set. These character categories have not been seen during the training process. More results can be found in supplementary materials.*

than one thousand training samples to achieve good enough performance. Thus we use the optimal set in all other experiments.

4.3.4. Capability of FontRNN

Next, we conduct a series of experiments with the configurations described above on different fonts to qualitatively verify the effectiveness and efficiency of our FontRNN. For each experiment, we only use 775 training samples and it converges after around 7000 steps (spending less than 3 hours with an NVIDIA Titan-X 12G GPU). Some experimental results are shown in Figure 7, the first row represents the input reference characters and the next every two rows represent a kind of font: the upper row denotes the ground truth while the lower one shows the generated results by FontRNN. As we can be seen from Figure 7, most ground-truth glyphs are very similar to the generated ones, namely our proposed FontRNN can handle different styles (e.g., neat and cursive) of fonts effectively. These generated writing trajectories will be used as character skeletons in the following shape synthesis experiment.

Table 3: *The style classification accuracy.*

	Accuracy
pix2pix	0.8999
DCFont	0.7717
zi2zi	0.9195
FontSL	0.8341
Ours	**0.9846**
GroundTruth	0.9911

Table 4: *Results of user study.*

Model	Frequency
pix2pix	0.0989
DCFont	0.1222
zi2zi	0.1897
FontSL	0.1677
Ours	**0.4215**

4.4. Shape Synthesis

After generating character skeletons using FontRNN, we adopt an additional simple networks described in section 3.3 to recover the contour shapes of these characters.

4.4.1. Qualitative Comparison

As shown in Figure 8, we test our algorithm on six kinds of fonts that vary from thin to thickness, tough to smooth, and many other attributes, and compare the generated results with those generated using other four existing methods: pix2pix [IZZE17], DCFont [JLTX17], zi2zi [Tia17] and FontSL [LZX16]. From Figure 8 we can see that performances of other four methods are relatively poor (especially on the unusual font FZTLJW) while our method can synthesize high-quality results that are difficult to be distinguished from the ground truth. Moreover, our method always outperforms other algorithms at the intersections of strokes, as depicted in Figure 9, the intersections of strokes in our results are always reasonable and sharp. The reason is that each of our skeleton is generated by drawing strokes one by one instead of "rendering" an image at once and thus the stroke intersections of our synthesis results are always realistic. Furthermore, the binary loss (see Eq (20)) also makes the intersections clearer.

4.4.2. Quantitative Comparison

We quantitatively evaluate our method and other methods from two aspects: content and style. With regard to content evaluation, we utilize the VGG19 [SZ14] as a Chinese character recognizer and train it on 273 fonts then test it on six fonts that are not used for training. As we can see from Table 2, our method achieves the highest accuracy, which obviously demonstrates that our method is more capable to maintain the content information of characters.

Similarly, in term of style evaluation, we train a VGG11 model on the ground truth of above six fonts as a 6-class style classifier. The testing results are shown in Table 3, we can see that the accuracy of our method is closest to the ground truth, that is to say the styles of fonts synthesized by our method are most similar to the ground truth. In conclusion, quantitative analysis proves that our method is superior in terms of both content and style to other existing methods compared here.

4.4.3. User Study

For further comparison of the synthesis results generated by different methods, we conduct an experiment of user study. Specifically, we make an online questionnaire that consists of 100 5-choice questions and each contains one real (i.e., ground truth) glyph image and five corresponding fake glyph images generated by five different methods, participants need to pick the one that is most similar to the ground truth from the five fake images. Finally, a total of 79 people took part in this test. Selecting frequency of the characters generated by each method is shown in Table 4. We can see that the glyph images generated by our method are more likely to be selected than other methods. In other words, more people think that the glyphs generated by our method are more similar to the ground truth compared to other approaches.

4.4.4. Failure Cases

In fact, performance of our method is still far from perfect. As shown in Figure 10, there are roughly two types of failure cases in our method. The first one is mainly caused by the imperfect reference skeletons, and the reason of the stroke missing of the output in Figure 10(1) is that the missing stroke of the character from the reference dataset is too short, thus the generated result is also very short and even lost or covered by other strokes. The another one is because of the error accumulation during the FontRNN decoding phase. Namely, FontRNN predicts the current timestep output based on the previous output, the error will accumulate to a large extent if the sequence is long. For example, as shown in Figure 10(2), the prediction at the end of the generated skeleton becomes untrustworthy thus the quality of the final synthesis result is poor.

4.5. Handwriting Generation

In addition to generating large-scale Chinese fonts with different styles, our proposed FontRNN can generate cursive but readable handwritten Chinese characters automatically (see Figure 12). Moreover, even for the same character, FontRNN will write variously at every time like humans (see Figure 11).

As mentioned in section 2, [ZYZ*18] also propose to use RNN to generate readable handwritten Chinese characters and thus it is the most relevant work to our FontRNN. However, because of the joint training of char embedding c, it needs more than two millions training samples. On the contrary, we adopt the idea of style transfer and use a much smaller training set (with only 2000 samples).

As shown in Figure 11, the generated characters in each row are not exactly the same on account of the random sampling output process, and handwritten characters generated by our FontRNN contain richer details (namely, with more colors) than those produced by [ZYZ*18]. Furthermore, the method in [ZYZ*18] does not always produce correct results, e.g., as indicated by the last two rows in Figure 11(a), there are some missing strokes which make these characters poor in readability.

Furthermore, although the method proposed in [ZYZ*18] can be used to synthesize handwritten characters, these character classes must have been seen during training. Contrastively, as depicted in Figure 12, our FontRNN can generate character classes that have not been seen during training because of the utilization of transfer learning strategy.

5. Conclusion

This paper proposed a novel network called FontRNN using RNN with monotonic attention mechanism and transfer learning strategy for Chinese characters generation in the sequential data format, which can be conveniently used for synthesizing large-scale Chinese fonts. Moreover, FontRNN can learn to write cursive but readable handwritten Chinese characters like humans. Compared with most existing CNN base approaches which use image-like representations, FontRNN can reflect the way of humans writing, which is drawing strokes one by one in right order. Both qualitative and quantitative experiments demonstrated the superiority of our FontRNN on large-scale fonts synthesis as well as handwritten character generation.

Acknowledgements

This work was supported by National Natural Science Foundation of China (Grant No.: 61672056 and 61672043) and Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology).

References

[APH18] AKSAN E., PECE F., HILLIGES O.: Deepwriting: Making digital ink editable via deep generative modeling. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018), ACM, p. 205. 2

[BBB*18] BHUNIA A. K., BHUNIA A. K., BANERJEE P., KONWER A., BHOWMICK A., ROY P. P., PAL U.: Word level font-to-font image translation using convolutional recurrent generative adversarial networks. In *2018 24th International Conference on Pattern Recognition (ICPR)* (2018), IEEE, pp. 3645–3650. 2

[BC94] BERNDT D. J., CLIFFORD J.: Using dynamic time warping to find patterns in time series. In *KDD workshop* (1994), vol. 10, Seattle, WA, pp. 359–370. 7

[BCB14] BAHDANAU D., CHO K., BENGIO Y.: Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014). 1, 2, 4, 7

[BCS*16] BAHDANAU D., CHOROWSKI J., SERDYUK D., BRAKEL P., BENGIO Y.: End-to-end attention-based large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on* (2016), IEEE, pp. 4945–4949. 2

[CGZ17] CHANG J., GU Y., ZHANG Y.: Chinese typeface transformation with hierarchical adversarial network. *arXiv preprint arXiv:1711.06448* (2017). 3

[CK14] CAMPBELL N. D. F., KAUTZ J.: Learning a manifold of fonts. *ACM Trans. Graph. 33*, 4 (July 2014), 91:1–91:11. URL: http://doi.acm.org/10.1145/2601097.2601212, doi:10.1145/2601097.2601212. 2

[CWF*15] Chen L., Wang S., Fan W., Sun J., Naoi S.: Beyond human recognition: A cnn-based framework for handwritten character recognition. In *Pattern Recognition (ACPR), 2015 3rd IAPR Asian Conference on* (2015), IEEE, pp. 695–699. 1

[CZPM18] Chang B., Zhang Q., Pan S., Meng L.: Generating handwritten chinese characters using cyclegan. *arXiv preprint arXiv:1801.08624* (2018). 1, 3

[DP73] Douglas D. H., Peucker T. K.: 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 (1973), 112–122. 6

[GPAM*14] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y.: Generative adversarial nets. In *Advances in neural information processing systems* (2014), pp. 2672–2680. 2

[Gra13] Graves A.: Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013). 2, 3, 4

[Ha15] Ha D.: Recurrent net dreams up fake chinese characters in vector format with tensorflow. *blog.otoro.net* (2015). URL: http://blog.otoro.net/2015/12/28/recurrent-net-dreams-up-fake-chinese-characters-in-vector-format-with-tensorflow/. 3

[HAU19] Hayashi H., Abe K., Uchida S.: Glyphgan: Style-consistent font generation based on generative adversarial networks. *arXiv preprint arXiv:1905.12502* (2019). 2

[HE17] Ha D., Eck D.: A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477* (2017). 3, 5

[HS97] Hochreiter S., Schmidhuber J.: Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780. 6

[IS15] Ioffe S., Szegedy C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015). 5

[IZZE17] Isola P., Zhu J.-Y., Zhou T., Efros A. A.: Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), IEEE, pp. 5967–5976. 1, 3, 5, 9

[JLTX17] Jiang Y., Lian Z., Tang Y., Xiao J.: Dcfont: an end-to-end deep chinese font generation system. In *SIGGRAPH Asia 2017 Technical Briefs* (2017), ACM, p. 22. 3, 9

[Kog15] Kogan G.: A book from the sky: Exploring the latent space of chinese handwriting. http://genekogan.com/works/a-book-from-the-sky/, 2015. 3

[LPM15] Luong M.-T., Pham H., Manning C. D.: Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015). 2, 4, 7

[LYWW11] Liu C.-L., Yin F., Wang D.-H., Wang Q.-F.: Casia online and offline chinese handwriting databases. In *2011 International Conference on Document Analysis and Recognition* (2011), IEEE, pp. 37–41. 6

[LYWW13] Liu C.-L., Yin F., Wang D.-H., Wang Q.-F.: Online and offline handwritten chinese character recognition: benchmarking on new databases. *Pattern Recognition 46*, 1 (2013), 155–162. 1

[LZCX18] Lian Z., Zhao B., Chen X., Xiao J.: Easyfont: A style learning-based system to easily build your large-scale handwriting fonts. *ACM Trans. Graph. 38* (2018), 6:1–6:18. 6, 8

[LZX16] Lian Z., Zhao B., Xiao J.: Automatic generation of large-scale handwriting fonts via style learning. In *SIGGRAPH ASIA 2016 Technical Briefs* (2016), ACM, p. 12. 2, 9

[MGM15] Miao Y., Gowayyed M., Metze F.: Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on* (2015), IEEE, pp. 167–174. 1

[MTS*17] Miyazaki T., Tsuchiya T., Sugaya Y., Omachi S., Iwamura M., Uchida S., Kise K.: Automatic generation of typographic font from a small font subset. *arXiv preprint arXiv:1701.05703* (2017). 2

[PLS*14] Pan W., Lian Z., Sun R., Tang Y., Xiao J.: Flexifont: a flexible system to generate personal font libraries. In *ACM Symposium on Document Engineering* (2014). 2

[PVZ*15] Parkhi O. M., Vedaldi A., Zisserman A., et al.: Deep face recognition. In *BMVC* (2015), vol. 1, p. 6. 1

[RFB15] Ronneberger O., Fischer P., Brox T.: U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (2015), Springer, pp. 234–241. 5

[RHGS15] Ren S., He K., Girshick R., Sun J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (2015), pp. 91–99. 1

[RLL*17] Raffel C., Luong M.-T., Liu P. J., Weiss R. J., Eck D.: Online and linear-time attention by enforcing monotonic alignments. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 2837–2846. 2, 4, 7

[RMC15] Radford A., Metz L., Chintala S.: Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015). 3

[SBM80] Suen C. Y., Berthod M., Mori S.: Automatic recognition of handprinted characters—the state of the art. *Proceedings of the IEEE 68*, 4 (1980), 469–487. 1

[SLJ*15] Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A.: Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 1–9. 1

[SRL*17] Sun D., Ren T., Li C., Su H., Zhu J.: Learning to write stylized chinese characters by reading a handful of examples. *arXiv preprint arXiv:1712.06424* (2017). 1, 3

[SSB16] Semeniuta S., Severyn A., Barth E.: Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118* (2016). 6

[SVL14] Sutskever I., Vinyals O., Le Q. V.: Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (2014), pp. 3104–3112. 2

[SZ14] Simonyan K., Zisserman A.: Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014). 9

[Tia17] Tian Y.: zi2zi: Master chinese calligraphy with conditional adversarial networks. https://kaonashi-tyc.github.io/2017/04/06/zi2zi.html, 2017. 1, 3, 9

[XJJL09] Xu S., Jin T., Jiang H., Lau F. C.: Automatic generation of personal chinese handwriting by capturing the characteristics of personal handwriting. In *IAAI* (2009). 2

[ZJX15] Zhong Z., Jin L., Xie Z.: High performance offline handwritten chinese character recognition using googlenet and directional feature maps. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on* (2015), IEEE, pp. 846–850. 1

[ZWC11] Zhou B., Wang W., Chen Z.: Easy generation of personal chinese handwritten fonts. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on* (2011), IEEE, pp. 1–6. 2

[ZYZ*18] Zhang X.-Y., Yin F., Zhang Y.-M., Liu C.-L., Bengio Y.: Drawing and recognizing chinese characters with recurrent neural network. *IEEE transactions on pattern analysis and machine intelligence 40*, 4 (2018), 849–862. 1, 3, 9, 10

[ZZ14] Zong A., Zhu Y.: Strokebank: Automating personalized chinese handwriting generation. In *AAAI* (2014), pp. 3024–3030. 2

[ZZYL16] Zhong Z., Zhang X.-Y., Yin F., Liu C.-L.: Handwritten chinese character recognition with spatial transformer and deep residual networks. In *Pattern Recognition (ICPR), 2016 23rd International Conference on* (2016), IEEE, pp. 3440–3445. 1