Creating New Chinese Fonts based on Manifold Learning and Adversarial Networks

Yuan Guo, Zhouhui Lian[†], Yingmin Tang, Jianguo Xiao

Institute of Computer Science and Technology, Peking University, PR China

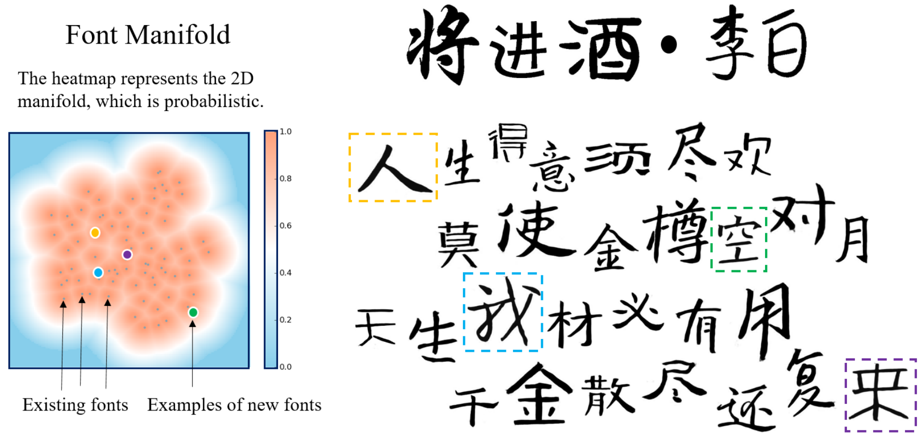

Figure 1: *Illustration of our Chinese font manifold. On the left, we visualize a manifold learned from 72 existing Chinese fonts which are denoted as dark points. Every point sampled from the manifold represents a unique font. On the right, we show a text rendering example in which each character is rendered in a different font style derived from the manifold. Characters in the first row are rendered in existing fonts while the others are in new font styles generated by randomly selected points in the manifold. Here, four character examples of new fonts are marked in the colors of their corresponding points in the manifold.*

Abstract

The design of fonts, especially Chinese fonts, is known as a tough task that requires considerable time and professional skills. In this paper, we propose a method to easily generate Chinese font libraries in new styles based on manifold learning and adversarial networks. Starting from a number of existing fonts that cover various styles, we firstly use convolutional neural networks to obtain the representation features of these fonts, and then build a font manifold via non-linear mapping. Using the font manifold, we can interpolate and move between those existing fonts to get new font features, which are then fed into a generative network learned via adversarial training to generate the whole new font libraries. Experimental results demonstrate that high-quality Chinese fonts in various new styles against existing ones can be efficiently generated using our method.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation; I.2.4 [Artificial Intelligence]: Learning—Connectionism and neural nets

1. Introduction

With the worldwide adoption of digital messaging, we have access to various types of fonts. However, designing new fonts, especially Chinese fonts, is a time-consuming task and requires expertise. Currently, convolutional neural networks (CNNs) have shown impressive capacity in many generative tasks, such as image synthesis [DTSB15] and texture transfer [ZZP*17]. Motivated by these, we aim to develop a new method based on CNNs and manifold learning to efficiently generate high-quality new Chinese fonts.

[†] Corresponding author. E-mail: lianzhouhui@pku.edu.cn

DOI: 10.2312/egs.20181045

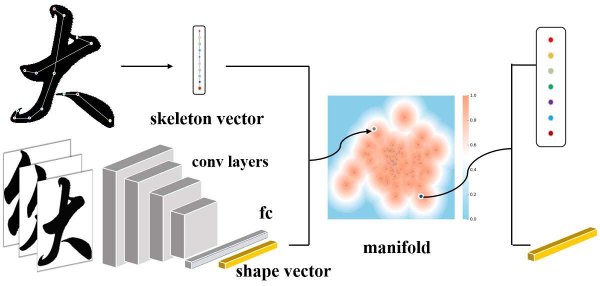

Figure 2: *An overview of our manifold building procedure.*

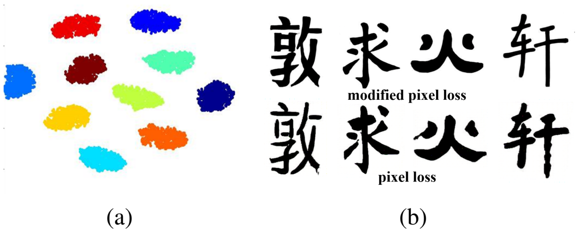

Figure 3: *(a) Visualization for the shape vectors of 10 fonts. (b) Comparison of results synthesized by the networks with pixel loss and modified pixel loss, respectively.*

Up to now, several methods regarding font generation have been presented. [CK14] proposed a method to build an English font manifold, which is essentially an outline based manifold. But failures always occur if the outlines of a character in different styles are quite dissimilar from each other. [LZX16] presented a font generation method in which users need to input a set of sample characters.

Deep learning methods have been widely used in image synthesis. Generative Adversarial Networks (GAN) [GPAM*14] excel at synthesizing realistic images. “pix2pix”, proposed by [IZZE16], is an efficient image to image style-transfer framework, which combines *U-net* architecture and adversarial training. However, those methods fail to generate high-quality images with complex shapes.

In this paper, we propose a method to automatically generate new Chinese fonts. Due to the complicated structures of Chinese characters, we decompose a glyph into two parts: the skeleton and outline shape. Given a set of training fonts, for each character of each font style, we extract feature vectors to represent its skeleton and outline shape, respectively. Then, we map these vectors into a low dimensional manifold by using a dimension reduction technique. By smoothly interpolating or extrapolating in the manifold, we can get some new feature vectors with reasonable properties. Feeding a new feature vector into the trained deep neural network, we can get a character in new font style. In this manner, we finally come up with a strategy that makes the automatic generation of large-scale Chinese fonts in brand new styles possible.

2. Method

Our method contains three steps. First, we need to extract the font feature vectors for all training data. Then, we learn font manifolds using the vectors. Finally, we train a generative network via adversarial training to synthesize character images.

2.1. Feature Extraction

As mentioned above, the information of a given glyph can be described by its skeleton and outline shape. Thus, the feature vector also consists of two parts: skeleton vector and shape vector.

Skeleton Vector: Since existing methods are not able to accurately extract the skeleton of every stroke for all Chinese characters due to their complexity, as shown in Figure 2, we manually label the key points of each stroke in the same stroke order of a standard reference character. Then, those key points are utilized to build the skeleton vector. It should be noted that the same character in different font styles may have different numbers of manually-labeled key points, which makes the length of skeleton vector unequal. So distance-based uniform up-sampling is applied after labeling to ensure the numbers of sampled points for each character in different font libraries are identical. Since a character consists of some strokes, the information that each point belongs to which stroke is also saved as an extra file, which will be used latter.

Shape Vector: As shown in Figure 2, we adopt a CNN model to compute the shape vector. Specifically, the VGG19 model [SZ14] is utilized with some modifications to train a font classifier. We change the neuron number of FC7 layer from 4096 to 50, and the last softmax layer is also modified to match the number of font classes. Finally, the output of FC7 layer of the trained VGG19 model is selected as the shape vector of a given character.

We visualize the shape vectors of 10 fonts in Figure 3, where characters in the same font are gathered together. In order to build a font manifold, we have to describe the style of all characters in a font library as a same shape vector. Thereby, we utilize the average shape vector of these characters as their shape vector.

2.2. Building Font Manifold

The Gaussian Process Latent Variable Model (GP-LVM) [Law04] is a non-linear probabilistic dimensionality reduction technique, which maps a high dimensional dataset Y to a low dimensional ‘latent’ dataset X. After feature extraction, we obtain several high dimensional feature vectors and attempt to find a latent space to represent them. Therefore, GP-LVM is well suited for our purpose. As shown in Figure 2, by using GP-LVM, we can build a manifold for each character in various fonts. Details are presented below.

Suppose there are M font libraries available, we have M high dimensional feature vectors, and each vector is composed by concatenating the skeleton vector and shape vector as follows

$$Y = \left[\begin{array}{cccccc} \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_i & \cdots & \mathbf{y}_M \end{array} \right]^{\mathrm{T}}, \tag{1}$$

$$\mathbf{y}_i = \left[\begin{array}{c} \mathbf{v}_{sp,i} \\ \mathbf{v}_{sk,i} \end{array} \right]^{\mathrm{T}}, \tag{2}$$

where $\mathbf{y}_i$ denotes the i-th feature vector, $\mathbf{v}_{sp,i}$ and $\mathbf{v}_{sk,i}$ are the shape vector and skeleton vector, respectively. Elements of the feature

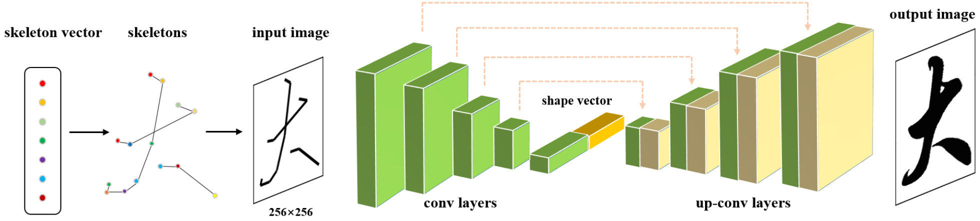

Figure 4: *The Character Rendering Network.*

vector should be normalized into [0,1], and it is also subtracted by its mean vector to ensure the zero mean.

GP-LVM considers that every vector is independent of others, and the likelihood of Y is defined by

$$P(Y|X,\theta)=\prod_{i=1}^{M}N(\mathbf{y}_i|\mathbf{0},C(X,X|\theta)+\sigma^2 I), \tag{3}$$

$$k(\mathbf{x}_i,\mathbf{x}_j)=\alpha\exp(-\beta\|\mathbf{x}_i-\mathbf{x}_j\|_2^2), \tag{4}$$

where N means Gaussian distribution, I is the identity matrix, and σ is a noise variance that accounts for miss matches. $C(X,X|\theta)$ denotes the covariance matrix calculated by covariance function $k(\mathbf{x}_i,\mathbf{x}_j)$. We get the latent variables by maximizing the likelihood jointly over the latent vectors $X=[\cdots\mathbf{x}_i\cdots]^{\mathrm{T}}$ as well as the hyper-parameters θ as follows

$$X^*,\theta^*=\arg\max_{X,\theta}[\log(P(Y|X,\theta))]. \tag{5}$$

The latent vectors are initialized with the PCA reduction values of Y, and hyperparameters θ are initialized with a uniformative prior. Gradients can be calculated by using Conjugate Gradient [Law05].

With the learned font manifold, new feature vectors can be reconstructed. Let $\hat{\mathbf{x}}$ be a point in the manifold, the corresponding high dimensional vector $\hat{\mathbf{y}}$ can be calculated by

$$\hat{\mathbf{y}}=C(\hat{\mathbf{x}},X^*|\theta^*)[C(X^*,X^*|\theta^*)]^{-1}Y, \tag{6}$$

where $[C(X^*,X^*|\theta^*)]^{-1}$ is precomputed. Afterwards, we add the mean feature vector to $\hat{\mathbf{y}}$ and get the data that contains the skeleton vector and shape vector.

After generating a new feature vector from the manifold, we have its shape vector $\mathbf{v}_{sp}$ and skeleton vector $\mathbf{v}_{sk}$. Assume that we want to create a font library with large numbers of Chinese characters in the same style, we still need to compute the skeleton vectors of all other characters that share the same shape vector. In other words, given another character, we need to find a point $\hat{\mathbf{x}}$ whose corresponding shape vector's value is also $\mathbf{v}_{sp}$ in its manifold. The point can be found by solving the following optimization task

$$\begin{aligned}&\arg\min_{\hat{\mathbf{x}}}\ \|\hat{\mathbf{v}}_{sp}-\mathbf{v}_{sp}\|_1\\ \text{s.t.}\ &\begin{bmatrix}\mathbf{v}_{sp}\\ \mathbf{v}_{sk}\end{bmatrix}^{\mathrm{T}}=C(\hat{\mathbf{x}},X^*|\theta^*)[C(X^*,X^*|\theta^*)]^{-1}Y,\end{aligned} \tag{7}$$

which can be solved by applying the simulated annealing algorithm [KGV*83].

Eventually, by searching over all the other characters' manifolds to find their most matched corresponding points, we get all the skeleton vectors we need and feed them into a generative neural network, which is called Character Rendering Network, to synthesize the complete shapes of all characters.

2.3. Character Rendering Network

Architecture: As shown in Figure 4, since the input of the network is an image, we need to plot the points in the skeleton vector and link points sequentially in each stroke. Here, we use the extra file saved in skeleton vector extraction to link the points which belong to the same stroke.

Our network consists of two parts. The first part contains several convolution layers and outputs an encoded vector. We concatenate the encoded vector with our shape vector as the input of the second part of our network, which has several up-convolution layers and outputs a rendered image. Similar to [IZZE16], we connect the layers of the first part to the corresponding layers in the second part. Furthermore, for adversarial training, we use a discriminative network, which is the same as the one used in [IZZE16].

Loss Function: In our network, we combine the adversarial loss with a modified pixel-wise loss mentioned below. Here, we denote the input image and desired output image as I_i and $\hat{I}_i$, respectively.

For adversarial training, the GAN framework learns two networks with competing losses. Our network G acts as a image generator trying to fool the discriminator, and the discriminator D tries to distinguish between generated and real images. We use the adversarial losses $\mathcal{L}_{(G)}$ and $\mathcal{L}_{(D)}$ defined in GAN [GPAM*14] as follows

$$\mathcal{L}_{(G)}=\sum_i log(1-D(G(I_i))), \tag{8}$$

$$\mathcal{L}_{(D)}=-\sum_i log(D(\hat{I}_i))-\sum_i log(1-D(G(I_i))). \tag{9}$$

To ensure the quality of synthesis results, we use pixel-wise loss (L1 distance) in our loss function. The key idea is that since we focus more on the outlines, we need to assign different weights on different pixels. Namely, pixels near the outlines have a higher weight than others. Specifically, the pixel-wise loss is computed by

$$\mathcal{L}_{(pixel)}=\sum_i \mathbf{W}_i\left\|\hat{I}_i-G(I_i)\right\|_1, \tag{10}$$

where $\mathbf{W}_i$ is the weight matrix. Finally, the loss function of our

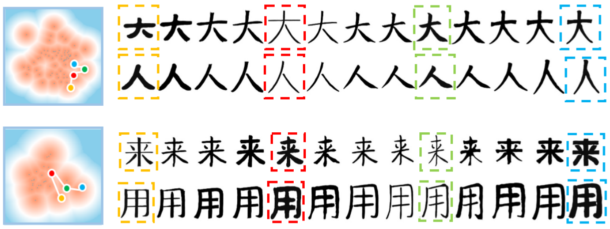

Figure 5: *Interpolation and jointly changing results.*

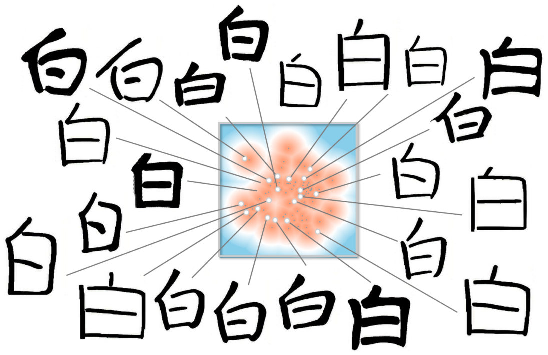

Figure 6: *Demonstration of an example font manifold for Chinese character "bai" and synthesis results obtained from the manifold.*

Character Rendering Network is defined as:

$$\mathcal{L}_{loss} = \mathcal{L}_{(G)} + \mathcal{L}_{(pixel)}. \tag{11}$$

3. Experiments

Training details: We collect 72 font libraries for training. For each font, we manually label the skeletons for 2,000 characters. As for the Character Rendering Network, the input image has a resolution of 256×256, and there are 7 convolution layers and 7 upconvolution layers. They both have 256 4×4 kernels and 2×2 stride with ReLU [NH10] as the activation function. We train the network for 40 epochs with batch size 32 and learning rate 0.001. Here, 40 epochs is maximum. During training, we calculate the mean error of each iteration. If the error doesn't decrease in the previous 1k iterations, we regard the network as well-trained and stop training.

Loss function comparison: We compare different results obtained via the modified pixel-wise loss and normal pixel-wise loss. As shown in Figure 3(b), the image quality is better with the modified pixel-wise loss.

Manifold interpolation and exploration: We perform interpolation and exploration in the manifold. As shown in Figure 5, we choose some points which are colored, and interpolate among them. As shown in the first and third rows, we observe that our method can produce a smooth transformation and interpolation between two fonts. Moreover, the exploration results, shown in Figure 6, demonstrate the diversity of results synthesized by our manifold.

Font generation via jointly changing: Our method is capable of generating jointly changing results. We use the strategy mentioned above to find the most matched skeletons in another manifold and then feed them into the Character Rendering Network. As shown in the second and forth rows of Figure 5, every character corresponds to the one in the first and third rows, respectively. We can find that the points coordinate well between two characters and the glyphs in the second and forth rows also have smooth changes between different font styles. Thus, the goal of generating large-scale Chinese fonts in brand new styles is achieved.

4. Conclusion

Our paper proposed a novel method to generate new Chinese fonts by building font manifolds learned from existing font libraries. The method requires no input from users, and there is no limitation on the size of character set. Experimental results demonstrated that our method is capable of generating new and high-quality Chinese fonts. Yet, there also exist limitations in our method. For example, the jointly changing process sometimes fails due to the inevitable reconstruction loss from the low dimensional space to high dimensional space. In the future, we plan to apply some advanced techniques and strategies to overcome those drawbacks.

Acknowledgements

This work was supported by National Natural Science Foundation of China (61672056, 61472015 and 61672043) and Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology).

References

[CK14] Campbell N. D., Kautz J.: Learning a manifold of fonts. *TOG 33*, 4 (2014), 91. 2

[DTSB15] Dosovitskiy A., Tobias Springenberg J., Brox T.: Learning to generate chairs with convolutional neural networks. In *CVPR* (2015), pp. 1538–1546. 1

[GPAM*14] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y.: Generative adversarial nets. In *NIPS* (2014), pp. 2672–2680. 2, 3

[IZZE16] Isola P., Zhu J.-Y., Zhou T., Efros A. A.: Image-to-image translation with conditional adversarial networks. *arXiv* (2016). 2, 3

[KGV*83] Kirkpatrick S., Gelatt C. D., Vecchi M. P., et al.: Optimization by simulated annealing. *science 220*, 4598 (1983), 671–680. 3

[Law04] Lawrence N. D.: Gaussian process latent variable models for visualisation of high dimensional data. In *NIPS* (2004), pp. 329–336. 2

[Law05] Lawrence N.: Probabilistic non-linear principal component analysis with gaussian process latent variable models. *JMLR 6*, Nov (2005), 1783–1816. 3

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

[NH10] Nair V., Hinton G. E.: Rectified linear units improve restricted boltzmann machines. In *ICML-10* (2010), pp. 807–814. 4

[SZ14] Simonyan K., Zisserman A.: Very deep convolutional networks for large-scale image recognition. *arXiv* (2014). 2

[ZZP*17] Zhu J.-Y., Zhang R., Pathak D., Darrell T., Efros A. A., Wang O., Shechtman E.: Toward multimodal image-to-image translation. In *NIPS* (2017), pp. 465–476. 1