EasyFont: A Style Learning-Based System to Easily Build Your Large-Scale Handwriting Fonts

ZHOUHUI LIAN, BO ZHAO, XUDONG CHEN, and JIANGUO XIAO, Institute of Computer Science and Technology, Peking University, China

Generating personal handwriting fonts with large amounts of characters is a boring and time-consuming task. For example, the official standard GB18030-2000 for commercial font products consists of 27,533 Chinese characters. Consistently and correctly writing out such huge amounts of characters is usually an impossible mission for ordinary people. To solve this problem, we propose a system, *EasyFont*, to automatically synthesize personal handwriting for all (e.g., Chinese) characters in the font library by learning style from a small number (as few as 1%) of carefully-selected samples written by an ordinary person. Major technical contributions of our system are twofold. First, we design an effective stroke extraction algorithm that constructs best-suited reference data from a trained font skeleton manifold and then establishes correspondence between target and reference characters via a non-rigid point set registration approach. Second, we develop a set of novel techniques to learn and recover users' overall handwriting styles and detailed handwriting behaviors. Experiments including Turing tests with 97 participants demonstrate that the proposed system generates high-quality synthesis results, which are indistinguishable from original handwritings. Using our system, for the first time, the practical handwriting font library in a user's personal style with arbitrarily large numbers of Chinese characters can be generated automatically. It can also be observed from our experiments that recently-popularized deep learning based end-to-end methods are not able to properly handle this task, which implies the necessity of expert knowledge and handcrafted rules for many applications.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Shape modeling**;

Additional Key Words and Phrases: Handwriting, Chinese, style learning, fonts

ACM Reference format:
Zhouhui Lian, Bo Zhao, Xudong Chen, and Jianguo Xiao. 2018. EasyFont: A Style Learning-Based System to Easily Build Your Large-Scale Handwriting Fonts. *ACM Trans. Graph.* 38, 1, Article 6 (December 2018), 18 pages.
https://doi.org/10.1145/3213767

This work was supported by the National Natural Science Foundation of China (Grant No.: 61472015, 61672056 and 61672043), National Language Committee of China (Grant No.: ZDI135-9), National Key Research and Development Program of China (2017YFB1002601) and Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology).
Authors' addresses: Z. Lian, B. Zhao, X. Chen, and J. Xiao, No. 128 Zhongguancun North Street, Haidian District, Beijing 100080, China; emails: {lianzhouhui, bozhao, chenxudong, xiaojianguo}@pku.edu.cn.

0730-0301/2018/12-ART6 $15.00
https://doi.org/10.1145/3213767

1 INTRODUCTION

Computer fonts are widely used in our daily lives. Nowadays, texts displayed in almost all books, posters, computers, mobile devices, and the like, are rendered using various fonts mainly created by professional companies. Although the number of font products has increased rapidly in the last two decades, existing resources still cannot satisfy the demand of every individual since more and more people want to render texts in their own handwriting styles, which are unique and full of personal information. Communicating with others by posting texts in personal handwriting styles instead of using uniform printing font styles could make those experiences more conformable and interesting.

However, building a handwriting font library with a large number of different characters is not easy. As we know, it is not a problem for writing systems (e.g., English) that only contain a small alphabet. For example, to create your own English handwriting font, only 26 letters and corresponding capitals need to be written. The whole font generation process can be accomplished in a few minutes by using some existing tools (e.g., FontCreator (2017) and FontLab (2017)). Yet, the task becomes tougher when the number of characters included in the font library increases. Let us take Chinese fonts as an example, the official character set GB18030-2000 consists of 27,533 Chinese characters. What's more, shapes and structures of many Chinese characters are very complicated. Figure 1(a) shows an example of the Chinese character pronounced as "biang," which has 57 strokes. As we know, to be a qualified font library, not only the glyph of each character should represent the correct meaning, but also the style of all glyphs must be consistent. According to a report made by FounderType (Founder 2017), a leading Chinese font producing company, it takes more than 12 months for a group with three to five experienced font designers to generate a GB18030-2000 Chinese font library. Therefore, building a complete Chinese font library in his/her own personal handwriting style is usually an impossible task for an ordinary person.

One possible way to accelerate the efficiency of producing large-scale font libraries is to exploit the redundancy of components (i.e., radicals, stroke sets, and strokes) for characters in a given character set. In other words, typically, components of characters in a selected subset are adequate to cover all the characters' components. Following this intuitive idea, several methods have been reported (Lin et al. 2014; Zhou et al. 2011) to generate a given user's handwriting font library from a number of characters written by the user. However, there exist the following two intrinsic drawbacks that hinder the application of this kind of method in practical use: (1) It is not guaranteed that all components to be reused can be correctly extracted, which prevents those methods from being fully automatic; (2) Not only is the percentage of characters that need to be written too large (more than 20%), but also the

Fig. 1. Examples of some writing systems that contain large numbers of different characters.

quality of the auto-generated font library is not satisfactory for real applications.

The other possible solution attracting many researchers is the utilization of deep learning techniques, which have become extremely popular in the last few years. Methods adopting deep neural networks not only obtain state-of-the-art performance in many classical Computer Vision tasks (He et al. 2016; Krizhevsky et al. 2012; Long et al. 2015; Russakovsky et al. 2015; Simonyan and Zisserman 2014) including image classification, object detection, sematic segmentation, and so on, but also become more and more competitive in solving generative problems (Gatys et al. 2017; Isola et al. 2017) thanks to the introduction of Generative Adversarial Networks (GAN) (Goodfellow et al. 2014) and its variants (Arjovsky et al. 2017). Deep learning based synthesis approaches are good at transferring color/texture styles for images, but less capable of constructing new shapes with complex structures. Scripts, such as Chinese characters, are shapes with high-level information and complicated structures in which even tiny variations of location and geometry for their elements may greatly change their meanings and/or styles. As we can see from our experimental results, these kinds of end-to-end methods work poorly for the task of synthesizing Chinese characters in handwriting styles. This is mainly due to the lack of understanding of the high-level information about the characters such as the constitution and layout of their basic elements (e.g., strokes).

Therefore, in order to establish such a fully automatic handwriting font generation system, we believe that the following two important problems need to be resolved: *"How can we automatically decompose the input glyphs and thus convert them into uniform and learnable data?"* and *"How can we describe and reconstruct handwriting style for a given user?"*

This article aims to handle the challenging task of automatic generation of large-scale font libraries. We develop a system that only requires the user to spend a short time (less than 30 minutes) writing out a small number of commonly-seen Chinese characters on blank papers. Then, the system can automatically synthesize all the other characters in the same style and generate the user's personal handwriting font library with an arbitrarily large number of Chinese characters. It should be pointed out that although our current system only supports the generation of Chinese fonts, the proposed method can be easily extended to any other writing systems (see some examples in Figure 1 and experimental results in Figure 21) that have the following two properties: (1) the total number of different characters is relatively large; (2) all characters can be decomposed into given limited numbers and types of strokes. Unless otherwise specified, *font library* and *character* mentioned in this article mean *Chinese font library* and *Chinese character*, respectively.

As shown in Figure 2, using our system to build a large-scale handwriting font library is convenient, as the user only needs to write a small amount (as few as 1%) of carefully-selected Chinese characters on blank papers, take pictures of them, and upload those photos to our system. After receiving these text images, a GB18030-2000 font library in the user's personal handwriting style can be automatically generated by the system in about two hours. More specifically, we first extract the writing trajectory of each stroke for every individual character image segmented from input text pictures based on a non-rigid point set registration approach and several heuristic rules. Then, artificial neural networks (ANNs) are utilized to learn and reconstruct the user's overall handwriting style, which can be decomposed into stroke shape style and stroke layout style. Meanwhile, handwriting details including stroke connectivity and shapes of contours are also properly described and recovered. Finally, a complete personal font library can be generated by vectorizing both images of human-written samples and machine-generated handwritings for all other characters. Experiments including Turing tests with 97 participants verify that the proposed system is able to accurately learn personal handwriting style, automatically synthesize indistinguishable handwritings, and quickly generate high-quality large-scale font libraries for ordinary people. Our experiments also demonstrate that domain knowledge is still critical for many machine learning tasks (e.g., handwriting synthesis and font design), in which only a small number of training samples are available. Recently-popularized deep learning based end-to-end approaches cannot handle those kinds of problems well without hand-crafted features and rules. To the best of our knowledge, our work (Lian et al. 2016) is the first to be able to automatically generate a practical handwriting font library in a user's personal style with arbitrarily large numbers of Chinese characters. This article, which upgrades the original system by adopting several new techniques and presents more implementation details and experimental results, is the extended version of our conference papers (Chen et al. 2017; Lian et al. 2016).

Major contributions of this article are threefold:

(1) We design an effective stroke extraction algorithm that constructs the best-suited reference data from a trained font skeleton manifold and then establishes correspondences between target and reference characters via a non-rigid point set registration approach. In this way, we can know precisely how the user writes those characters.
(2) We propose a novel system that makes the easy generation of large-scale handwriting fonts possible for ordinary people. Moreover, by utilizing a carefully-designed input character selection scheme, automatic stroke extraction, and handwriting detail recovery techniques, high-quality text rendering results of font libraries generated by the system can be guaranteed in practical use for almost any kind of handwriting styles (even strange and cursive styles). Extensive experiments have been conducted to verify the effectiveness of our method.
(3) We have manually specified the writing trajectory of each stroke for all 27,533 Chinese characters in the standard "Kaiti" style (reference) and a smaller character set in other two handwriting styles (target). This ground truth data together with a set of other handwriting image data in various

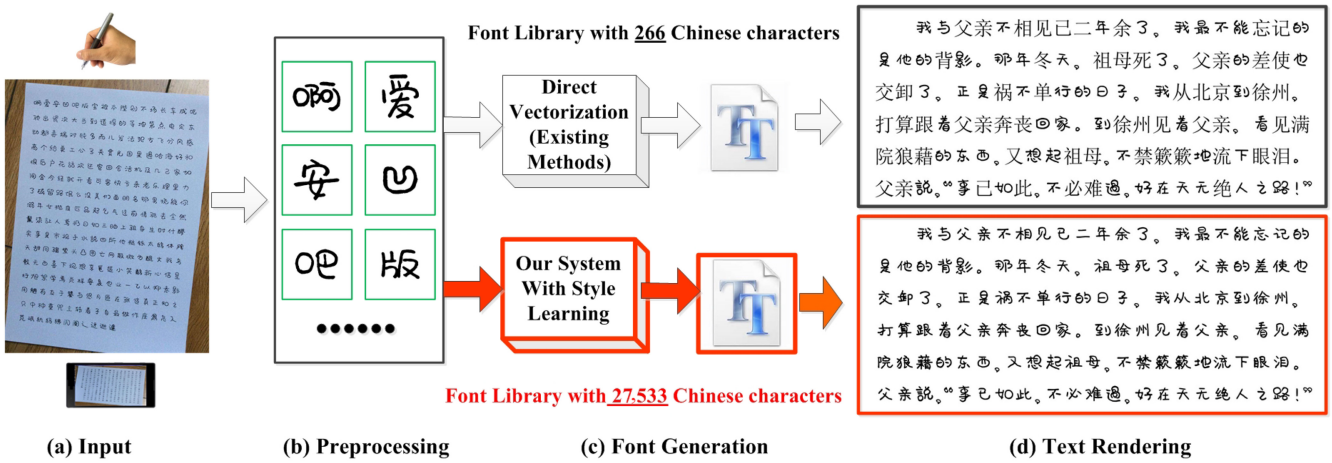


Fig. 2. Overview of the proposed system and comparison of our approach with an existing font generation method. Using our system, the user only needs to write a small number (e.g., 266) of Chinese characters on a blank paper and uploads the photo to our system. The system can then automatically generate a handwriting font library in the user's personal style with huge amounts (e.g., 27,533) of Chinese characters. The text rendered using the font library generated by our system looks similar to a text written by the user. In comparison, the one rendered by the other font library only containing 266 human-written characters is unreadable, due to the mixture of human-written samples and characters rendered in a word processor's default font style (e.g., "Songti").

styles used in our experiments can serve as a benchmark for handwriting synthesis, which are publicly available on our website.[1]

2 RELATED WORK

Virtual Brush: Traditionally, font designers need to use pens, brushes, and/or rulers to write or draw characters on paper, and then scan and vectorize them to construct computer fonts. In the last few years, many researchers have tried to develop "virtual brushes" that could generate high-fidelity paintings or writings similar to results obtained using real brushes or pens. With such kinds of virtual brushes, designers are able to conveniently carry out their font design works on computers, and thus, the efficiency of font design and production can be markedly enhanced. Wang and Pang (1991) simulated the writing speeds of calligraphers and the variation of ink amounts on brushes via some computer graphic techniques to achieve different writing styles. Then, the contours of characters were described using cubic Bezier splines. In this way, users can interactively create Chinese calligraphy on the computer. However, simulation results of their system are not satisfactory since the method they employed is too simple to model the complicated physical properties of real brushes. In Strassmann (1986), a 2D virtual brush model was first proposed by taking above-mentioned physical properties into account. Strassmann (1986) described the brush as a collection of bristles that evolve with writing trajectories. From then on, researchers have developed a large number of algorithms for virtual brushes, such as the multi-parameter controlled realistic virtual brush (Xu et al. 2003), the reverse modeling based virtual brush (Wong et al. 2008), the data-driven 3D virtual brush (Baxter and Govindaraju 2010), and so on.

Using devices with multiple degrees of freedom (DOFs), skilled users are able to generate expressive results based on above-mentioned virtual brush techniques. However, it is hard to control such devices to create high-quality drawings and handwritings for ordinary people, and most of them use common input devices with lower DOFs. To address this problem, Lu et al. (2012) proposed a data-driven method for synthesizing the 6D hand gesture data for users of low-quality input devices, and the 6D trajectories generated by their method can be utilized as inputs for any virtual brush engine to get expressive handwriting results. Then, they successively reported two data-driven painting systems including RealBrush (Lu et al. 2013), which uses scanned images of physical media to synthesize drawings, and DecoBrush (Lu et al. 2014), which can synthesize structured decorative patterns along with trajectories written by users. Meanwhile, Zitnick (2013) presented a handwriting beautification algorithm based on the insight that the appearance of the average of multiple instances in the same handwriting style is better than most of the individual instances. Recently, several virtual brushes that were specifically designed for Chinese characters have been proposed (Xia and Jin 2009; Yi et al. 2014). For instance, the approach presented in Yi et al. (2014) can synthesize high-quality Chinese handwritings in any font style by assembling the best-suited stroke segments, which are selected from the corresponding style database trained offline, according to the trajectories written by users.

Computer Calligraphy: The above-mentioned methods can be applied to greatly facilitate the design and production of computer fonts. However, what these methods simulate are tools people use but not the artistic creation processes, and thus, large amounts of manual operations are still required to generate calligraphy or paintings by using virtual brushes on computers. Dong et al. (2008) proposed to automatically generate new stroke shapes for Chinese characters based on a statistical model that learns how to

[1] http://www.icst.pku.edu.cn/zlian/EasyFont/.

create strokes in brand new styles from a training set with lots of calligraphy samples. Xu et al. (2005) presented an analogous reasoning based system to automatically generate Chinese calligraphic artworks. The system is able to create Chinese calligraphy in various new styles by learning parametric representations of training calligraphic images. Then, Xu et al. (2007) designed a numeric grading approach to evaluate the beauty of calligraphy based on a back-propagation neural network and applied it in their previous system (Xu et al. 2005) to generate visually pleasing calligraphic artworks in completely new styles. Xu et al. (2009) also proposed an augmented shape grammar system to describe the personal handwriting styles of Chinese characters and employed a trained ANN to measure the similarity of writing styles between two character images. By integrating this method into the system proposed in Xu et al. (2005), Chinese characters represented in the same style of input calligraphic images can be automatically generated. Wang et al. (2008) and Yu et al. (2009) introduced automatic generation methods for Chinese calligraphy by reusing strokes and/or radicals of a set of input character images, while Xu et al. (2008) not only selected existing components in the input data set but also used computer-generated components with a certain probability to synthesize output character shapes. A different but promising approach to generate handwritten characters was reported in Dolinsky and Takagi (2009), where the authors found that recurrent neural networks (RNNs) could be used to approximately model the naturalness of handwritten characters, which was defined in their paper as the difference between handwritings and archetypal font characters. Based on the naturalness learning approach, they successfully generated 34 hiragana characters in a user's handwriting style via the naturalness model learned from 27 hiragana characters written by the user. Li et al. (2014) proposed to synthesize Chinese calligraphy in a similar topological style by learning information extracted from characters written by the user. They adopted a new feature called the WF-histogram to measure the topological similarity between two characters and then established a evaluation model to guide the synthesizing process. More recently, Haines et al. (2016) developed a system that is able to render new texts in a given user's handwriting style. By learning parameters for spacing, line thickness, pressure, color, paper texture, and so on, high-quality synthesized handwritten texts in the similar style as the input data can be obtained by the system. However, the methodology presented in Haines et al. (2016) cannot be extended to solve the problem of automatic generation of large-scale handwriting fonts due to the following two reasons. First, manual interactions on the semiautomatic user-interface are still required to precisely segment and label the training data. If we want to provide the service of our system to thousands of millions of ordinary people, fully automatic processing is always necessary. Second, as mentioned by the authors, *"Languages with high character counts, e.g., Chinese (*>3000*), would be prohibitively difficult to synthesize, as it is unreasonable to capture sufficient data,"* and thus, it is intrinsically unsuitable to solve the problem mentioned in this article.

Font Generation: Although some expressive and creative calligraphic artworks can be generated by existing computer calligraphy approaches, they are still not applicable for creating large-scale font libraries (e.g., Chinese fonts) that contain huge amounts of different characters. On the one hand, a practical Chinese font library typically consists of at least thousands of characters. However, those existing methods, which were originally tested on small datasets specified by corresponding researchers, can only deal with a small number (mostly several hundreds) of Chinese characters with relatively simple topological structures and geometric shapes. On the other hand, many results obtained by those existing methods are not able to satisfy the high requirements on visual quality, readability, and style consistency for the shapes of characters in practical font libraries. Therefore, instead of using automatic font generation approaches, currently font designers still rely heavily on personal experiences and manual operations to generate commercial font products with the help of some typeface editing software systems (e.g., FontCreator (2017) and FontLab (2017)). Due to the complexity and particularity of Chinese characters, some companies have developed their own font designing systems to create Chinese fonts other than directly using universal typeface editing software. For instance, HAND, a typeface editing system developed by the world's largest Chinese font producer (i.e., Founder Group (Founder 2017)), was specially designed for Chinese fonts by taking the unique characteristics (e.g., hierarchical representations) of Chinese characters into account.

Indeed, these CAD systems could markedly improve the efficiency of font design, but lots of times and manual operations (spending about 12 months by three to five skilled font designers) are still required to create a commercial Chinese font library. During the last two decades, several works that tried to reduce the heavy manual operations by introducing more heuristic rules and automatic processing into the font producing procedure have been reported (Fan 1990a, 1990b; Lai et al. 1996; Lian and Xiao 2012; Lin et al. 2014), but these methods are still far from practical. Recently, Suveeranont and Igarashi (2010) presented a system to automatically generate all characters in the font library based on a glyph designed by the user. Their key idea is to manipulate in a natural manner the contours and skeletons of characters in a template font library, which is selected to have the most similar font style as the input character, to construct shapes for all characters in the required font style. Their experiments showed that the system could significantly enhance the efficiency of font design for English characters. However, their system needs to manually create accurate and complex shape models for all characters, which is unsuitable for handling large-scale font libraries that contain huge amounts of characters with complicated shapes. Campbell and Kautz (2014) proposed to build a generative manifold of standard fonts to smoothly interpolate and move between existing fonts. In this way, large numbers of new high-quality fonts can be easily generated from the so-called font manifold. Nevertheless, their method is also unsuitable for Chinese characters due to the requirement of establishing accurate correspondence between the outlines of two glyphs. Phan et al. (2015) developed a system called "FlexyFont," which is able to construct a complete English font library by applying the learned style, transferring rules to assemble segmented parts of given glyphs to generate other glyphs. More recently, Lake et al. (2015) presented a concept learning method using probabilistic program induction, which is able to produce new exemplars in novel writing styles given a single character image. However, their method

cannot handle the tough problem of generating handwritings in the same style as input samples for all other unseen characters.

In the last decade, deep neural networks (Hinton and Salakhutdinov 2006; Silver et al. 2016) have been widely adopted to handle many challenging tasks in areas of Computer Vision (Russakovsky et al. 2015), Computer Graphics (Soltani et al. 2017), and the like, and obtain state-of-the-art results in tasks including image classification, object detection, sematic segmentation, and so on. Recently, the surprising performance of AlphaGo (Silver et al. 2016) against professional Go players including large numbers of world champions shows more potential of deep learning approaches in addressing many other tough problems previously thought to be impossible. Until now, several deep learning based architectures, which are either specifically designed or suitable for font generation have been proposed. The Autoencoder (Hinton and Salakhutdinov 2006) is well known to be able to effectively reconstruct images. By implementing neural style transfer (Gatys et al. 2017), a given image can be converted into a new perceptually appealing image that possesses the similar style as the other target image. Learning texture/color styles is much easier than shape geometric styles, which typically contain high-level intelligent knowledge that is even hard to learn for ordinary human beings. Generative tasks that aim to create high-quality 2D images (Isola et al. 2017) or 3D objects (Soltani et al. 2017) have recently attracted more and more attention from researchers. There also exist some works specifically focusing on synthesizing fonts in the same style as training samples (Baluja 2016; Bernhardsson 2016; Tian 2016), but according to our experiments, most of them are not able to generate reasonable results for complex Chinese handwritten characters. To the best of our knowledge, up to now, "Rewrite" (Tian 2016) and "pix2pix" (Isola et al. 2017) are the two best-performing existing frameworks that can be adopted to automatically generate Chinese fonts from a small set of handwritten samples. Specifically, "Rewrite" (Tian 2016) employs a traditional top-down CNN structure while "pix2pix" (Isola et al. 2017) has a powerful capability to learn a mapping from input images to output images mainly due to the utilization of a GAN. Although those "end-to-end" deep learning based architectures work well for some printing fonts given large amounts of training samples, without high-level domain knowledge, they still fail to achieve satisfactory performance when trying to synthesize glyphs with complicated structures for many handwriting styles.

The work that is most relevant to this article was reported in Zhou et al. (2011), where Zhou et al. developed a system to construct the glyphs of 2,500 relatively simple Chinese characters by reusing radicals of 522 characters written by a user, and thus built a small font library in the user's handwriting style. However, there exist three major drawbacks that hinder the practical use of their system. First, radicals instead of strokes are reused in their system, which results in the requirement of large numbers of input characters (more than 20%). Moreover, if characters with more complicated shapes are considered, more input characters will inevitably be needed by the system. Second, characters with overlapped radicals cannot be properly segmented out by their method, while these kinds of situations happen frequently in handwritten Chinese characters. Third, in their method selected radicals are reused and assembled directly based on the layout information extracted from a standard font library. Unless the required font style is almost the same as the standard one, it is not guaranteed that high-quality results can be obtained by their method. Also, at least 6,763 Chinese characters (GB2312 official standard) are required for a commercial font product, but their system can only deal with a much smaller character set. To solve those problems, we propose a system to automatically generate the practical handwriting font library (e.g., GB18030-2000) with arbitrarily large amounts of Chinese characters by learning style from a small number (as few as 1%) of human-written characters.

3 METHOD DESCRIPTION

As mentioned above, we intend to learn handwriting style from a small amount of characters written by an ordinary person, then automatically generate the whole handwriting font library, which can have arbitrarily large numbers of characters in the user's personal style. More specifically, during the offline processing period, we first manually specify the writing trajectory of each stroke for all (e.g., 27,533) characters in the standard "Kaiti" font library, which is employed as reference data for style learning. Then, a series of input character sets are properly chosen to satisfy different requirements in real applications (Section 3.1). Finally, a font skeleton manifold is built to generate the best-suited reference data for stroke extraction (Section 3.2). Online, we first automatically extract stroke trajectories for individual character images segmented from input text photos (Section 3.3 and 3.4). Then, we utilize ANNs to learn and reconstruct the user's overall handwriting style, which can be decomposed into stroke shape style and stroke layout style (Section 3.5). Meanwhile, handwriting details including stroke connectivity and shapes of contours are also properly described and recovered (Section 3.6). Finally, a complete personal font library can be generated by vectorizing both images of human-written samples and machine-generated handwritings for all other characters (Section 3.7). The pipeline of our system is shown in Algorithm 1 and more details are presented below.

3.1 Selecting Input Character Set

In order to imitate a user's handwritings, our system needs to learn the handwriting style by analyzing some samples written by the user. Here comes a critical question: "which characters should be written?" Obviously, without "seeing" enough handwritten samples, neither our system nor even professional calligraphers would be able to precisely mimic the user's handwritings. As we know, there exist 32 different types of strokes that constitute the basic elements of Chinese characters (see Figure 3(b)). So, one fundamental requirement of our system is that all types of strokes should appear at least once in the input character set. However, this is far from enough. Therefore, during the offline processing period, we manually specified the writing trajectory of each stroke for all 27,533 characters in the standard "Kaiti" font library, and defined 1,032 categories of components (see Figure 3(a)), which consist of a set of strokes. Since shapes of strokes in the same basic type may vary greatly, we further classify them into 339 fine-grained categories. With these reference data, a suitable character set can be determined by choosing characters that cover all 339 kinds of strokes. If better synthesis performance is required, we can select more

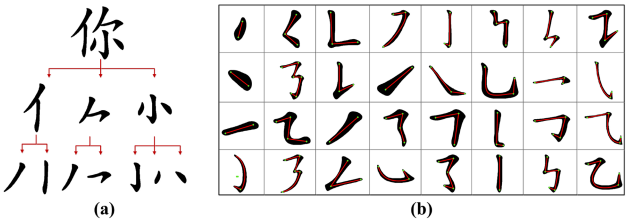


Fig. 3. (a) Demonstration of decomposing a Chinese character sample into components and finally strokes. (b) Examples of strokes that constitute the basic elements of Chinese characters. They can be classified into 32 basic categories.

ALGORITHM 1: The pipeline of our system.

Input:

Photos of papers with some characters written by a user;

1: **Text Segmentation**: Obtain individual character images by segmenting rectified text pictures;
2: **Stroke Extraction**: Extract writing trajectory of each stroke for every character image and select correct extraction results;
3: **Overall Style Learning**: Employ ANNs to learn the user's overall handwriting style;
4: **Details Modeling**: Analyze and describe the connectivity of all sequential stroke pairs and details on the contour for each type of strokes;
5: **Handwriting Synthesis**: Create trajectory for each character by adding the learned style on reference data and then recover handwriting details;
6: **Font Generation**: Vectorize images of human-written samples and synthesis results for other characters, and then generate a TrueType font library.

Output:

The user's personal handwriting font library.

characters for the input set so that all 1,032 types of components can also be completely covered.

To make the font product generated by our system perform better in real text rendering applications (see Figure 2), we choose to vectorize images of human-written samples and machine-generated handwritings for other characters together to build the practical font library. This is due to the fact that characters written by a person typically have random and unpredictable variations while machine learning systems can only capture the person's average and stable handwriting style. Through our experiments, we found that text rendering results consisting of evenly-distributed human-written samples and machine-generated characters look more natural and more similar as real handwritten texts compared to those containing only synthesized glyphs. Therefore, we would like to have an input character set that is able to cover about 50% or more characters that appear in all normal Chinese articles. To achieve this goal, we utilized automated crawls to acquire huge amounts of Chinese articles, comments on blogs, chatting notes, and the like, from the World Wide Web, and thus obtained a data set with more than 87 billion characters. By calculating the frequency of occurrence of each character in this data set, we obtain its average rate of coverage in a normal Chinese article. It can be observed, after sorting the coverage rates of all characters in descending order, that theoretically, the combination of the first 190 characters is able to cover about 50% content of any normal Chinese article. In this manner, the average rate of human-written characters appearing in an article rendered using the font library generated by our system can be estimated.

Table 1. Three Input Character Sets Adopted in this Article

Purposes	For test only	MinSet for real uses	OptSet for real uses
Character Number	639	266	775
Coverage Rate	21.9%	50.7%	59.3%

Based on the aforementioned criteria, as shown in Table 1, three input character sets are chosen for different purposes. The first one, which consists of 639 characters, is used in the experiments of this article to evaluate the performance of overall style learning algorithms. In this case, without considering coverage rate, characters are selected to simply ensure that all types of components and strokes can be covered by the character set. The second one, with 266 characters, serves as the minimum input character set (MinSet) of our system in practical use. This input character set not only includes the above-mentioned 190 characters that have the highest coverage rates, but also contains another 76 characters to ensure that all 339 categories of strokes can be written at least once. Adding the requirement of covering all kinds of components to the selection criterion adopted in the second case, we obtain the last character set that is composed of 775 commonly-seen characters. We call it the optimal input character set (OptSet) of our system due to the above-analyzed advantages and its high and robust performance in our experiments.

3.2 Learning Font Skeleton Manifold

To construct a font skeleton manifold, the thinning algorithm using mathematical morphology (Jang and Chin 1990) is first applied to get the writing trajectory (i.e., skeleton) of each character in the training set. Using the skeleton instead of the contour of each character simplifies the complexity of building a font manifold. We match the skeleton points of each individual character across all selected fonts via a non-rigid point set registration procedure with predefined stroke trajectory models. Then we use the dense skeleton point correspondences for each character as a basis to fit a non-linear manifold that ties the character in different font styles together into a single space (see Figure 4). Here, the non-linear manifold is learned by using the Gaussian Process Latent Variable Model (GP-LVM) (Lawrence 2005).

3.2.1 Character Matching. Specifically, we construct a high dimensional vector that contains the stroke skeleton points to represent each glyph. To learn the font skeleton manifold, we need to first establish point correspondences among glyphs in different font styles. Campbell and Kautz (2014) developed an effective approach to handle the character matching task. However, since the energy model proposed by them (Campbell and Kautz 2014) is only suitable to build correspondences among glyphs with the equal number of closed outlines, it fails to match the skeletons of Chinese characters that often possess different and complicated

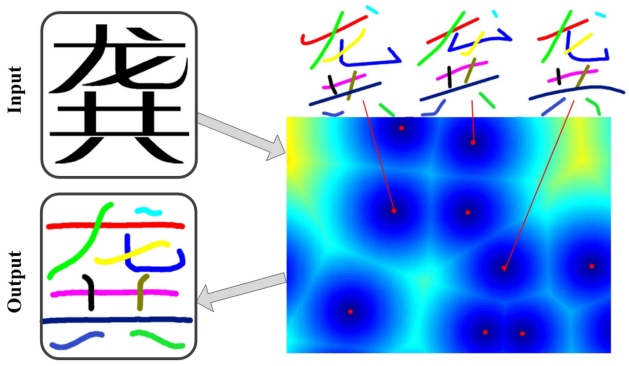


Fig. 4. Using the trained font skeleton manifold to construct the best reference for an input character image during the stroke extraction procedure.

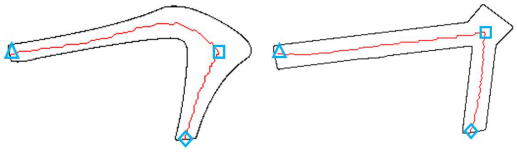


Fig. 5. Key point alignment between a standard stroke model in our database (left) and the corresponding stroke in another font style (right). The triangles, diamonds, and squares denote the corresponding start points, end points, and corner points of the two stroke trajectories.

topologies. Thereby, we choose to solve this problem by designing a new method that is based on the point set registration algorithm. To figure out the exact correspondence, we build a stroke model database, which contains strokes in 339 different categories. We manually select the key points on the writing trajectory of each stroke model and then get its revised skeleton with no fork points. There are mainly three kinds of skeleton key points: start point, end point, and corner point, around which rich information of writing styles exist. The skeleton segment between each key point pair is then uniformly sampled into a certain number of points. We develop a tool to manually label the stroke skeleton points of each character across selected fonts. Afterwards, we directly use the Coherent Point Drift (CPD) (Myronenko and Song 2010) algorithm to register the skeleton points of the input stroke to its corresponding stroke model. As shown in Figure 5, the key points are aligned to the stroke model and the skeleton segment between each key point pair is sampled into the same number of points as the stroke model.

Since the points on every stroke skeleton have been properly aligned with each other, correspondences of each character among a number of (e.g., 28) selected fonts can be established.

3.2.2 Training the GP-LVM. The Gaussian Process Latent Variable Model (GP-LVM) (Lawrence 2005) is an effective non-linear dimensionality reduction technique. It produces a probabilistic model of a high-dimensional dataset Y with a low-dimensional dataset X which is "latent". We are working in a very high-dimensional space at the beginning since there are about 600 skeleton points in each glyph on average. Compared to PCA, MDS, IsoMap, and other linear dimensionality reduction methods, GP-LVM performs much better in reconstructing high-dimensional data from corresponding low-dimensional latent vectors. Therefore, the GP-LVM is well suited for the task of learning such a font skeleton manifold.

Suppose there are M fonts, we need to generate M high-dimensional vectors for each character. Each vector is composed by sequentially putting all skeleton point samples together in stroke order as

$$\mathbf{v}_m = \begin{bmatrix} \begin{bmatrix} \mathbf{v}^{\mathbf{m}}_{\mathbf{1,1}} \\ \mathbf{v}^{m}_{1,2} \\ \vdots \end{bmatrix}^{\mathrm{T}} & \begin{bmatrix} \mathbf{v}^{\mathbf{m}}_{\mathbf{2,1}} \\ \mathbf{v}^{m}_{2,2} \\ \vdots \end{bmatrix}^{\mathrm{T}} & \cdots & \begin{bmatrix} \mathbf{v}^{\mathbf{m}}_{\mathbf{n,1}} \\ \mathbf{v}^{m}_{n,2} \\ \vdots \end{bmatrix}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}, \tag{1}$$

where $\mathbf{v}^m_{i,j}$ denotes the coordinate of the j-th point on the i-th stroke skeleton in font style m and n is the stroke number of the character. The value of the point coordinate should be normalized to $[0, 1]$. To apply GP-LVM, we subtract the mean vector $\bar{\mathbf{v}}$ off to get the vector $\mathbf{y}_m = \mathbf{v}_m - \bar{\mathbf{v}}$. This allows us to use a zero mean function, and the mean vector $\bar{\mathbf{v}}$ is actually the average character skeleton of the training data. Then, the high-dimensional dataset Y can be represented as

$$Y = [\, \mathbf{y_1} \quad \mathbf{y}_2 \quad \cdots \quad \mathbf{y}_i \quad \cdots \quad \mathbf{y}_M \,]^{\mathrm{T}}. \tag{2}$$

Using these skeleton vectors for Y, the training process of the GP-LVM considers the likelihood of Y as

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

where I is the identity matrix, M denotes the number of fonts, $C(X, X|\theta)$ denotes the covariance between vectors, and σ denotes the noise variance accounting for difference between every original high-dimensional vector and its reconstructed version. In our experiments, the manifold works well only when the parameter σ is set to be small (e.g., 0.1), which suggests that the font skeletons actually lie on a low-dimensional manifold. We get the latent variables by jointly maximizing the likelihood below over the latent vectors $X = [\cdots \mathbf{x}_j \cdots]^{\mathrm{T}}$ as well as the hyperparameters θ so that

$$X^*, \theta^* = \arg\max_{X,\theta} [\log(P(Y|X, \theta))]. \tag{4}$$

Generating the skeleton of a given character in a new font style from the manifold is straight-forward with the latent variables X^* and hyperparameters θ^* (Lawrence 2005). Suppose $\hat{\mathbf{x}}$ is the target location on the manifold, then the corresponding high-dimensional vector $\hat{\mathbf{y}}$ can be calculated by the following equation

$$\hat{\mathbf{y}} = C(\hat{\mathbf{x}}, X^*|\theta^*)[C(X^*, X^*|\theta^*)]^{-1} Y, \tag{5}$$

where $C(\hat{\mathbf{x}}, X^*|\theta^*)$ denotes the covariance between vectors and $[C(X^*, X^*|\theta^*)]^{-1}$ is precomputed. We add the above-mentioned mean vector $\bar{\mathbf{v}}$ to $\hat{\mathbf{y}}$, to get $\hat{\mathbf{v}}$ that consists of the coordinates of skeleton points of the character in a new font style. In this way, the font skeleton manifold is constructed offline to provide the most similar reference data for each input character image during the stroke extraction step (Section 3.4) described below.

There are two major reasons for building such a font skeleton manifold for our system. First, the manifold is able to generate infinite numbers of meaningful character trajectories in different styles while the number of manually labeled reference data is

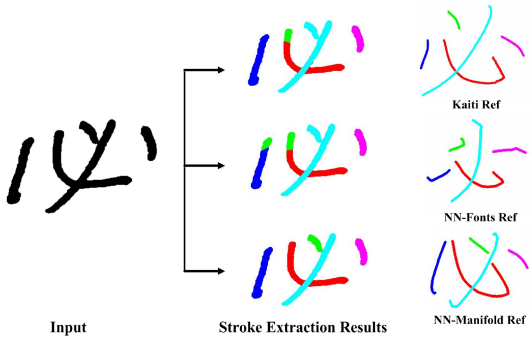


Fig. 6. Comparison of stroke extraction results for a handwritten Chinese character using the following three different reference data: Kaiti (a standard "Kaiti" font), NN-Fonts (nearest-neighbor retrieval on a given database with 28 fonts), NN-Manifold (nearest-neighbor retrieval on our font skeleton manifold).

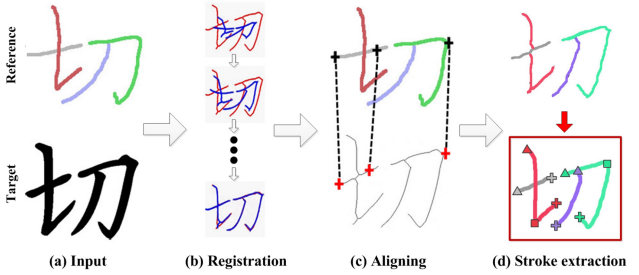


Fig. 7. Illustration of our stroke extraction algorithm. Given a target handwritten character and its best-suited corresponding reference (a), non-rigid point set registration between two trajectories is first implemented (b). Then, correspondences between them can be established (c). Finally, the trajectory of each stroke for the target character is extracted and the registration process is carried out again to locate key points (triangles, crosses, and squares denote start, end, and corner points, respectively) on each stroke's trajectory (d).

always limited, and thus, better stroke extraction performance can be achieved compared to other approaches (see Figure 6). Second, searching on the trained low-dimensional (e.g., two in this article) manifold instead of the high-dimensional (e.g., about 1,200) interpolation space makes the task of finding similar reference data for most input character images possible and more efficient.

3.3 Text Segmentation

As shown in Figure 2, anyone can use our system to generate their handwriting fonts quickly and conveniently. The user should first follow the instruction to write out all (266 or 775, depending on the quality required) characters in a selected input character set. In order to make a qualified font library, characters should be written separately without touching each other in a given order and consistently in size and style. Then, the user should take pictures for those papers in correct directions and upload the text photos to our system.

After receiving the user's text images, the system will automatically segment individual character images from those pictures in the following five steps: (1) apply Gaussian smoothing and adaptive image binarization on the original text images; (2) find regions with connected pixels, calculate their bounding boxes, and consider them as candidates of characters; (3) discard unsuitable candidates by applying several heuristic filters (e.g., size, ratio of length to width, ratio of black and white pixels). Then, go to the next step if the number of valid candidates is equal to the number of characters required to be written. Otherwise, dilate the image and go back to the second step; (4) compute the mass centers of the detected candidates in the bottom row of the picture and thus the rotation angle of the text image can be estimated by fitting a line for these center points; (5) sequentially segment individual character images from the rectified text images and label them with the unicode values of corresponding characters.

The text localization algorithm adopted here is quite simple and straightforward, but already sufficient enough for the proposed system. In practical use, the template-based text segmentation scheme we presented in Pan et al. (2014) has also been adopted to collect handwriting data from thousands of users via our website.[2] Other state-of-the-art text detection and recognition methods (e.g., Sun et al. (2016) and Zhang et al. (2016)) can also be integrated into our system to improve its robustness.

3.4 Stroke Extraction

Given a number of character images, in order to know how the user wrote these characters, we must precisely locate the writing trajectory of each stroke on the characters. The key idea of our method is to utilize the Coherent Point Drift (CPD) (Myronenko and Song 2010) algorithm to implement non-rigid registration between the skeletons of a given target character image and its best-suited reference (see Figure 7).

During the offline period mentioned above, a font skeleton manifold is learned and thus numerous font skeletons can be generated, from which we can find the most similar one for the input image as its reference data for stroke extraction. It is impractical to traverse everywhere in the manifold since the manifold space is continuous and infinite. However, it can be observed that locations nearby tend to generate similar character skeletons and the skeleton changes continuously from one location to another. Let $\mathbf{z}$ denote the location that generates the most similar skeleton to the target character image among the existing selected fonts in the manifold; we speculate that the best-suited reference character skeleton is likely to be located in the neighbourhood of $\mathbf{z}$.

Afterwards, we use the thinning directional feature (Jin and Gao 2004) to measure the similarity of character skeletons in different fonts. Specifically, given the skeleton of a character, we divide the image into $8 \times 8 = 64$ grids and then count the number of points in four directions (the horizontal direction, the vertical direction, the left falling diagonal direction and the right falling diagonal direction) in each grid. Thus, we get a feature vector of $8 \times 8 \times 4 = 256$ dimensions for each character. By calculating the minimum Euclidean distance between the feature vectors of the target character and the corresponding characters in the styles of existing selected fonts, we get the value of $\mathbf{z}$. From Equation (5), we can generate the corresponding character skeleton when traversing around the

[2] http://www.flexifont.com/.

location of $\mathbf{z}$. Here, we choose to traverse in circles centered at $\mathbf{z}$ and increase their radii gradually until reaching the threshold τ. For each character skeleton we generate, the thinning directional feature is calculated in the same way as mentioned above and we choose the one that is most similar to the target character image as its reference data (See Figure 4).

Since stroke labels are already known for all points on the best-suited reference data, after establishing correspondence between reference and target data, the writing trajectory of each stroke on the target character image can be obtained automatically (see Figure 7(d)). More specifically, let $\mathbf{X}_{N\times 2} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^T$ be the target point set with N points evenly sampled on the skeleton of a user-written character, and $\mathbf{Y}_{M\times 2} = (\mathbf{y}_1, \ldots, \mathbf{y}_M)^T$ be the reference point set (i.e., centroids of Gaussian Mixture Models (GMM)) with M points sampled on the corresponding reference character's writing trajectory. Registering the reference point set with target point set is equivalent to determining the locations of GMM centroids (i.e., θ) and the equal isotropic covariances of GMM distributions (i.e., σ^2) by minimizing the following objective function

$$E(\theta, \sigma^2) = -\sum_{n=1}^{N} \log\left(\sum_{m=1}^{M} \frac{1}{M}\frac{1}{2\pi\sigma^2} \exp^{-\frac{\|\mathbf{x}_n - \mathbf{y}_m\|^2}{2\sigma^2}} + \frac{1}{N}\right). \quad (6)$$

Here, we adopt the EM algorithm to solve this problem by iteratively implementing E and M steps until convergence.

After the above procedure, what we get are only a set of labeled target point sets. To implement style learning and detail recovery, we would like to describe the writing trajectory of each stroke as a single directional curve with several key points including the start point, end point, and corner points (see Figure 7(d)). We achieve this goal by applying the CPD algorithm (Myronenko and Song 2010) again to establish correspondence between the point set of each target stroke trajectory and the corresponding reference stroke model. Here, 339 reference stroke models have been built offline by manually specifying the writing trajectory and above-mentioned key points for each stroke type once.

It should be pointed out that in real applications, incorrect stroke extraction results always exist due to the existence of special and cursive personal handwriting styles. Obviously, these incorrect extraction results could seriously affect the performance of handwriting synthesis and might even cause the breakdown of our system. To solve this problem, we evaluate the correctness of stroke extraction results for a character by computing

$$C = C_{rec} + C_{sim} + C_{rule}, \quad (7)$$

where C_{rec} denotes the overlap ratio of the reconstructed character image, which is obtained by continuously drawing discs whose diameters are equal to the average stroke width along extracted stroke trajectories, to the original one; C_{sim} measures the similarity between each target stroke trajectory and its corresponding reference using several shape descriptors (i.e., Sobel-Roberts feature (Khosravi and Kabir 2010) and angles of lines between key points); and C_{rule} is related to the correctness of each stroke based on its intrinsic properties (e.g., stroke "heng" and "shu" should look like horizontal and vertical line segments, respectively) by designing heuristic rules for each type of stroke. In our system, we choose to discard the extracted stroke trajectories of the 20% of the human-written characters whose correctness values are smaller than other 80%. This is due to the observation from our experiments that stroke extraction results are correct for more than 90% characters in the above-mentioned test set consisting of 639 characters.

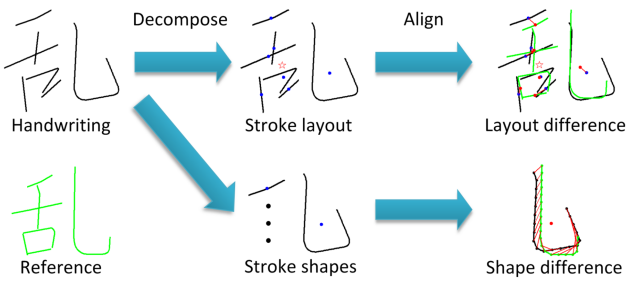


Fig. 8. Describing the handwriting style. Black and green trajectories denote handwriting and reference characters, respectively. Blue and red points are stroke centers of ref and $hand$. Pentacles denote character centers.

3.5 Overall Style Learning

In our system, the overall handwriting style is represented as the difference between trajectories of the reference character (ref) and handwritten character ($hand$). We decompose the Chinese character into a lower-level concept and structure, namely the stroke shape (SS) and stroke layout (SL) (see Figure 8). Hence, the overall handwriting style can be decomposed into the stroke shape style (SSS) and stroke layout style (SLS), which are represented by the differences of stroke shapes and stroke layouts (DSS and DSL), respectively, between ref and $hand$.

To calculate the DSS and DSL, we first sample the same number (N_P) of points $P_{ij}(k) = (x_{ij}(k), y_{ij}(k)), k = 1, 2, \ldots, N_P$ on the trajectory of each stroke for all reference and handwritten characters. Thus, strokes can be represented as points along stroke trajectories (for simplicity, unless otherwise specified, in this section, we directly use "stroke" to denote the "stroke trajectory"), e.g., $S_{ij} = (P_{ij}(1), P_{ij}(2), \ldots, P_{ij}(N_P))$. While, characters are represented as vectors of strokes $C_i = (S_{i1}, S_{i2}, \ldots, S_{iN_{S_i}})$, where N_{S_i} denotes the number of strokes in a character C_i. Then, the stroke center SC and character center CC can be easily computed. To better describe the shape of a stroke, we calculate the normalized stroke shape, which consists of relative positions of points on the stroke to the stroke's center of mass. Similarly, the normalized stroke layout can be obtained by calculating the relative positions of stroke centers to the character center. Thus, the normalized SS and SL for each stroke can be computed by $SS_{ij} = S_{ij} - SC_{ij}$ and $SL_{ij} = SC_{ij} - CC_i$, respectively. As mentioned above, the stroke shape style and stroke layout style are represented by DSS and DSL, which can be calculated as $DSS_{ij} = SS_{ij}^{hand} - SS_{ij}^{ref}$ and $DSL_{ij} = SL_{ij}^{hand} - SL_{ij}^{ref}$, where $SS_{ij}^{hand}, SL_{ij}^{hand}$ denote the normalized SS, SL for the stroke j of the handwritten character; and $SS_{ij}^{ref}, SL_{ij}^{ref}$ represent the normalized SS, SL of the corresponding reference.

3.5.1 Data Structures and Neural Networks. Figure 9 depicts an overview of how to learn and reconstruct the user's overall handwriting style using our method, which consists of the following

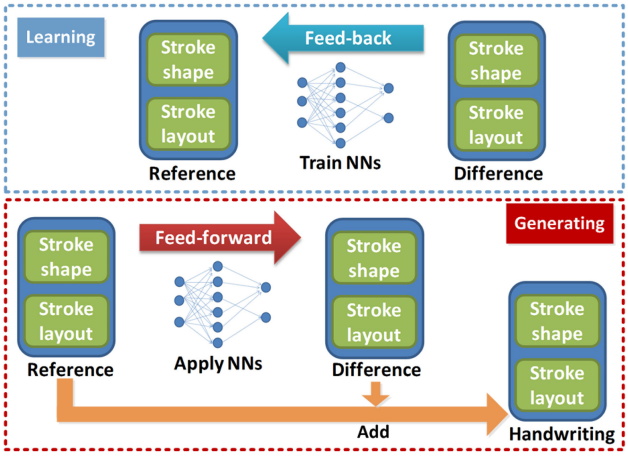


Fig. 9. Illustration of overall style learning and handwriting synthesis in our system.

Table 2. Classification of Data Structures

	Point-wise	Stroke-wise	Character-wise
Non-sequential	N	Y	Y
Sequential	Y	Y	N

two procedures: style learning and handwriting synthesis. In the style learning procedure, neural networks are utilized to capture overall handwriting style. The stroke shape of ref serves as input for learning stroke shape style, and the difference of stroke shapes between ref and $hand$ serves as output. Similarly, when learning stroke layout style, the input is the stroke layout of ref and the output is the difference of stroke layouts between ref and $hand$. Here, different data structures with various neural networks are employed and compared to learn stroke shape style, which is harder than the task of learning stroke layout style. The input and output data are built according to different data structures and neural networks chosen for our system.

Data structures for stroke shape style learning can be classified into sequential and non-sequential by time dependence, or point-wise, stroke-wise, and character-wise by granularity. The classification of data structures we use is shown in Table 2, in which Y means suitable and N means unsuitable.

For point-wise data structures, the input and output are $SS_{ij}^{ref}(k)$ and $DSS_{ij}(k)$ (i.e., each point), respectively. Non-recurrent neural networks (non-RNNs) are not suitable for this data structure, because it is hard to capture handwriting style from a single point. In recurrent neural networks (RNNs), we can treat points in each stroke as a sequence or points in each character as a sequence. For stroke-wise data structures, the input and output are SS_{ij}^{ref} and DSS_{ij} (i.e., each stroke), respectively. Both RNNs and non-RNNs are suitable to find the difference (style) of strokes between ref and $hand$. In RNNs, strokes in a character form a sequence. Although RNNs can find time dependence among strokes in a character, the limited number of training samples in our system restricts this ability of RNNs. For character-wise data structures, the input and output are SS_{i}^{ref} and DSS_{i} (i.e., each character), respectively. Only non-RNNs are suitable for this data

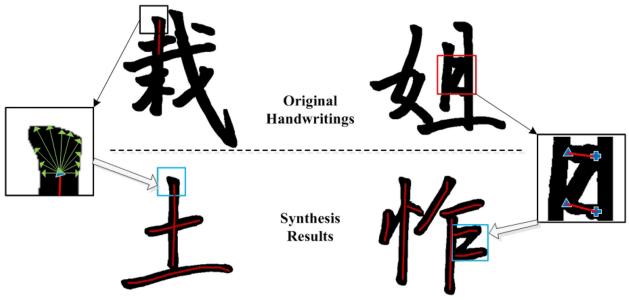


Fig. 10. Demonstration of how to recover handwriting details. Learned writing trajectories in the middle of glyphs are colored in red.

structure. In this case, the number of samples is much smaller than that in stroke-wise data structures.

We choose to learn overall handwriting style by the following three types of neural networks (NNs): Feed-forward neural network (FFNN) (Rumelhart et al. 1986), Elman recurrent neural network (Elman RNN) (Elman 1990), and Long Short-Time Memory (LSTM) network (Hochreiter and Schmidhuber 1997). FFNN is a non-RNN network while Elman RNN and LSTM are RNNs, which are able to capture time dependence. As we know, Elman RNN only has a short memory, but LSTM, a well-designed RNN, performs well in learning long-time dependence.

3.5.2 Learning Procedure. As shown in Figure 9, our learning procedure consists of three steps. First, suitable data structures and corresponding neural networks are selected based on the above-mentioned analyses. Then, we use the selected data structure to build the training data (including input and output data). Afterwards, we train the neural networks with prepared data to learn the stroke shape style and stroke layout style, respectively. Meanwhile, a validation set is used to automatically tune parameters during the training process. Finally, we obtain the trained neural networks, which can be used to synthesize handwritings in the generation procedure.

3.6 Recovering Handwriting Details

After capturing the user's overall handwriting style, writing trajectories of all characters can be generated. Then, the simplest way of creating synthesized glyphs is to render the trajectories with the average stroke width of human-written characters (see Figure 14 for some rendering examples).

However, as shown in Figure 10, the stroke width of each point on a writing trajectory may change greatly according to different handwriting behaviors, especially in the start and end regions of a stroke. Moreover, the connectivity of two sequential strokes is also an important feature of the user's handwriting style. To capture the handwriting details on the contour of a stroke, we divide the stroke into three regions, i.e., start, end, and middle regions. Locations of the start and end points are adjusted by ensuring that the largest distance between the start/end point and points on the stroke contour is equal to the twice of average stroke width w_s. Then, as shown in the left part of Figure 10, relative positions of points on a stroke's contour around the stroke trajectory's start point are obtained by emitting a number (e.g., 11) of rays evenly distributed in the half region opposed to the writing direction. If

any of these rays encounter the trajectories of other strokes before reaching the stroke's contour, we mark it as an invalid start region. Handwriting details at the end region can be captured in the same manner and the shape of the middle region is described by the stroke width values of a number (e.g., 10) of evenly-sampled points on the trajectory. As mentioned above, all strokes have been classified into 339 categories, so we can calculate the average values of the above-mentioned detail information for all valid regions in each type of stroke and use it to recover details on the contour when rendering the trajectory of a stroke that belongs to the same category.

To describe the connecting property for each pair of sequential strokes, we calculate a 339×339 matrix M_c in which the element m_{ij}^c denotes the probability of drawing trajectory between the end point of stroke i and the start point of stroke j. As shown in the right part of Figure 10, it is easy to judge whether two sequential strokes in a character have been written as a connected component or not by using our automatically-extracted stroke trajectories with key points. Then, the value of m_{ij}^c is obtained by computing the ratio of the number of connected pairs of strokes (i.e., stroke i and j) to the total number of this type of stroke pairs. When generating synthesis results, if the value of m_{ij}^c is larger than a random number P_c ($P_c \in [0, 1]$), a natural and smooth line with proper width values will be created to connect the end point of stroke i with the start point of stroke j.

3.7 Handwriting Synthesis and Font Generation

As shown in Figure 9, during the generation procedure (i.e., the handwriting synthesis procedure), reference data (i.e., SS_{ij}^{ref} and SL_{ij}^{ref}) are input into the trained NNs to estimate DSS_{ij} and DSL_{ij}. Then, we obtain SS_{ij}^{hand}, SL_{ij}^{hand} by applying DSS_{ij} and DSL_{ij} to SS_{ij}^{ref} and SL_{ij}^{ref}, respectively, i.e., $SS_{ij}^{hand} = DSS_{ij} + SS_{ij}^{ref}$ and $SL_{ij}^{hand} = DSL_{ij} + SL_{ij}^{ref}$. By setting the character center CC_i, positions of stroke centers and sampled points on strokes can be located using the calculated SS_{ij}^{hand} and SL_{ij}^{hand}.

After obtaining the synthesized writing trajectories for all characters that have not been written by the user, several beautification processes are implemented in our system for better visual effects. First, distortions of synthesis results can be reduced by smoothing generated trajectories slightly. Also, rendering trajectories properly using the above-mentioned technique to recover writing details could make synthesis results look more similar to real handwritings of the user. One of the most important merits of our method is that with the trained network arbitrarily large numbers of Chinese characters in the learned handwriting style can be generated automatically once the required reference data are provided. Finally, a Truetype font library in the user's personal handwriting style can be built by vectorizing and packaging both images of human-written samples and machine-generated handwritings for all other characters.

4 EXPERIMENTS

We carry out two groups of experiments in this section. The first group of experiments is mainly designed to investigate effects of different configurations and show performance comparison for the proposed overall style learning method that plays a key role in our system. The other group of experiments including Turing tests is conducted to demonstrate the effectiveness and superiority of our font generation system in real applications. Unless otherwise specified, settings of our experiments are chosen as follows: Characters in the "Kaiti" font library are adopted as reference data for style learning, because most Chinese people start to learn writing from imitating glyphs in the "Kaiti" style. The stroke layout style is learned by FFNN. The algorithms are implemented in Matlab on a PC with a 3.5GHz Intel i7-5930K CPU and 32.0GB RAM.

4.1 Font Generation without Handwriting Details

In the first group of experiments, we compare learning performance for methods with different configurations of data structures and neural networks. Mean square error (*MSE*) and correlation coefficient (R) are calculated to quantitatively evaluate the performance. To achieve accurate comparison and quantitative analyses, we manually specify the trajectory of each stroke for all characters written by two users (i.e., User 1 and 2) on an iPad with a fixed stroke width and adopt them as the ground truth in our experiments. The dataset is publicly available on our website so that other researchers can use it as benchmark database for handwriting synthesis. Here, the input character set consisting of 639 different Chinese characters (see more details in Section 3.1 and Table 1), which are able to cover all types of strokes and components of commonly used Chinese characters, is chosen in this group of experiments. These 639 characters are randomly divided into the training set, validation set and test set with partition ratios 4/6, 1/6, and 1/6, respectively.

4.1.1 Data Structures and Neural Networks. For stroke layout style learning, FFNN with five units in the hidden layer is adopted. We use SL_{ij}^{ref} as input each time, and the output is the difference of SL_{ij} between the reference and handwritten characters, i.e., DSL_{ij}. First, FFNN is trained on the training set to learn SLS. Then, the trained FFNN and the new input data are utilized to generate SL_{ij}^{hand} for new characters. We observe that during the learning procedure convergence occurs after about 8,000 iterations, and the R value evaluated on the test set is about 0.965, which means high similarity between the machine-generated data and ground truth data.

Next, we test NNs with different data structures to learn the stroke shape style, namely, Stroke-wise Learning with FFNN (SWL-FFNN), Stroke-wise Learning with Elman RNN (SWL-ERNN), Stroke-wise Learning with LSTM (SWL-LSTM), Point-wise Learning with LSTM in Per-stroke Sequence (PWL-LSTM-PSS), Point-wise Learning with LSTM in Per-character Sequence (PWL-LSTM-PCS), and Character-wise Learning with FFNN (CWL-FFNN).

Here, only one hidden layer is utilized in all NNs; that means the network structure is $I * H * O$, where I, H, and O denote the numbers of units in the input, hidden, and output layers, respectively. Table 3 lists the unit numbers in each NN, which are selected experimentally. We have tried using more layers and units in each NN, but the performance could not be improved. One possible

Table 3. *MSE* and *R* Values of Our Methods with Different Configurations Evaluated on the Test Set

	Point-wise			Stroke-wise			Character-wise		
	$I*H*O$	*MSE*	*R*	$I*H*O$	*MSE*	*R*	$I*H*O$	*MSE*	*R*
FFNN	–			$40*40*40$	0.129	*0.934*	$1000*400*1000$	0.153	*0.913*
Elman RNN	–			$40*50*40$	0.125	*0.937*	–		
LSTM-PSS	$2*20*2$	0.145	*0.926*	$40*50*40$	0.126	*0.936*	–		
LSTM-PCS	$2*5*2$	0.156	*0.919*						

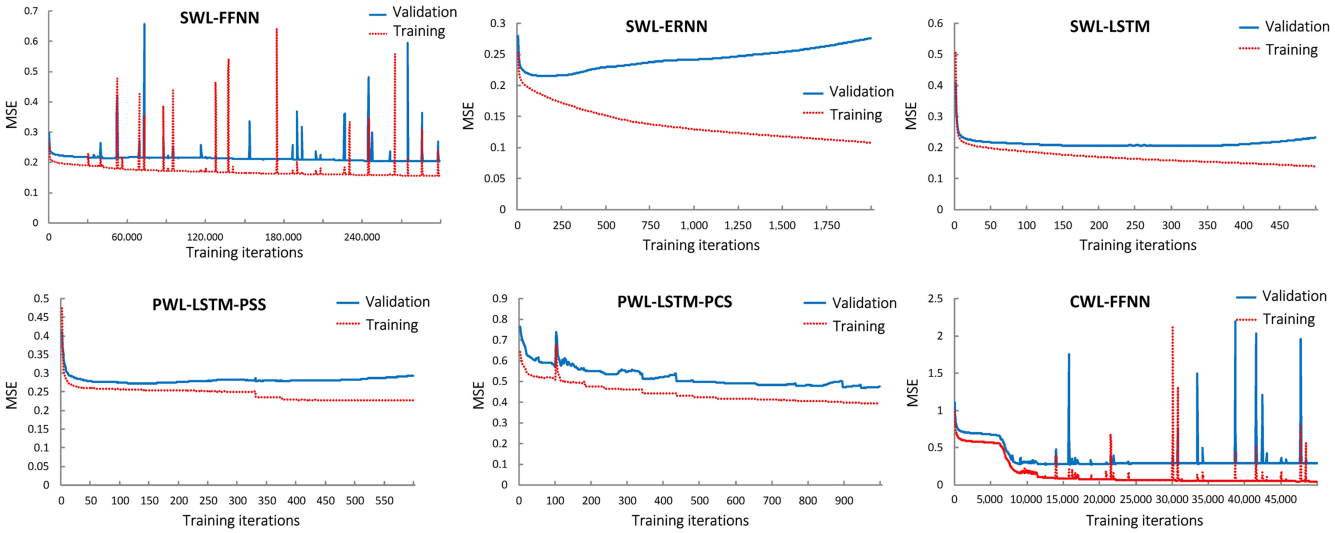


Fig. 11. Plots of MSE values evaluated on the training set and validation set of different methods for stroke shape style learning.

explanation is that, for small-scale data, shallow networks are often more suitable than deep networks.

It can be observed from Table 3 that methods with stroke-wise configurations perform better than those with point-wise and character-wise configurations. Therefore, stroke-wise configurations are more suitable for style learning when the number of training samples is small. Also, methods with different stroke-wise configurations perform similarly, which indicates that, for this small-sample learning task, recurrent neural networks (ERNN and LSTM) do not have obvious superiority against feed-forward neural networks (FFNN).

As we can see from Figure 11, which shows the performance of different methods, it takes a long time for the training error and validation error of FFNN to decrease until convergence. Therefore, we apply a trick by automatically adjusting the learning rate on time based on the tendency of errors. In this manner, the break of gradients due to large learning rate can be settled down soon. On the contrary, training errors of Elman RNN and LSTM continue to decrease quickly, and then overfitting occurs after hundreds of iterations in both Elman RNN and LSTM. Although it takes fewer iterations to reach the best condition for Elman RNN and LSTM, the time expenditure of each iteration is high because it cannot be computed in parallel.

Glyphs generated by different methods without smoothing are shown in Figure 12. We find that stroke-wise learning methods perform better than the others. It is interesting that stroke-wise learning approaches with different NNs achieve approximately similar visual effects. This indicates that data structures have a great impact on learning effect and our methods can learn personal handwriting style sufficiently from limited samples. More shakes appear in SWL-LSTM and SWL-ERNN, compared with SWL-FFNN, mainly because the limited number of samples are unable to stabilize more weights in LSTM and Elman RNN. Generally, SWL-FFNN achieves the best visual effect among these methods and thus is adopted to learn stroke shape style in other experiments.

4.1.2 Size of Input Character Set. We randomly choose different numbers of training samples from the above-mentioned 639 characters in which other samples are used as the testing data. Then, our methods trained on various numbers of samples are evaluated 10 times for each training sample size to calculate the mean values of *MSE* and *R*. It can be seen from Figure 13 that with only 270 samples our method can already achieve good enough performance with $MSE = 0.207$ and $R = 0.849$.

4.1.3 Comparison with a Concatenation Method. To validate the superiority of our system, we compare the proposed method with a concatenation method called Character Radical Composition Model (CRCM) presented in Zhou et al. (2011). Figure 14 shows the comparison of synthesis results obtained using CRCM and our approach, respectively. For the CRCM approach, we manually adjust the character segmentation results to ensure that all

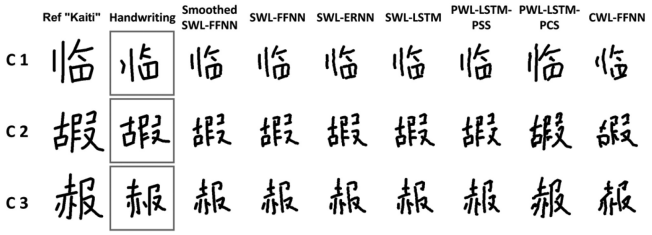


Fig. 12. Comparison of synthesis results for "User 1" using our methods with different configurations. C1, C2, and C3 denote different characters.

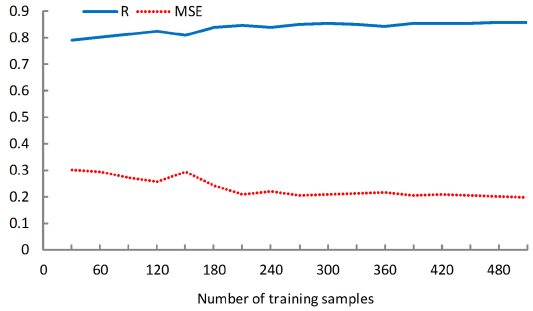


Fig. 13. Plots of MSE and R values evaluated on the same test set with different training sample sizes.

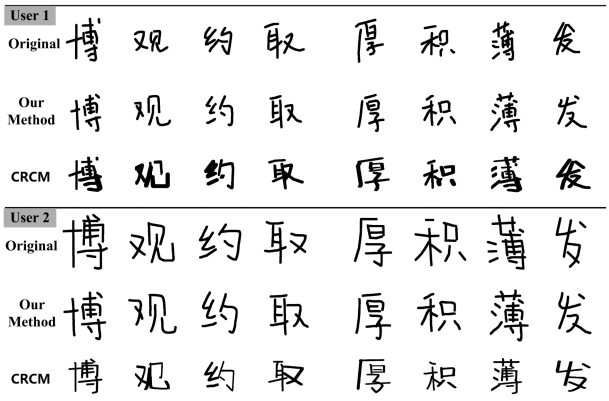


Fig. 14. Comparison of synthesis results for two users. The first line shows human-written characters. The second and third lines show synthesis results of our method and CRCM, respectively.

radicals are extracted correctly. It takes CRCM about $13\frac{1}{3}$ hours to generate a GB2312 font library that contains 6,763 simplified Chinese characters. But the font library generated by CRCM cannot be used directly without manual modifications. As we can see, there exist many obvious defects in the synthesis results of CRCM, such as the unseemliness of radical sizes and radical layouts. On the contrary, it only takes about $2\frac{1}{6}$ hours for our method to learn handwriting style and build the font library. In fact, due to the inevitable existence of incorrect radical extraction results, CRCM cannot be implemented automatically without manual interference, but our system can. Moreover, the synthesis results of our method are more visually pleasing (see Figure 14).

4.2 Font Generation for Real Applications

In the second group of experiments, we would like to examine the performance of our system in practical use. More specifically, the system tested here utilizes "FFNN" to learn stroke layout style and "SWL-FFNN" to learn stroke shape style. The OptSet (see Table 1) that consists of 775 characters is chosen as the input character set. Also, the proposed text segmentation scheme, stroke extraction, and detail recovering algorithms are all implemented to make sure that the system can be run fully automatically after receiving handwritten text photos uploaded by the user and the synthesis results can be indistinguishable from original handwritings. Although the MinSet (see Table 1) with only 266 characters can also be adopted as the input character set, and in fact, considerably good results (see Figure 2) are already obtained by using the MinSet, we still recommend utilizing the OptSet to guarantee better and more stable performance in real applications where strange, very cursive, and even incorrect handwritings might also be input to our system. As a matter of fact, on average it takes only about 20–30 minutes for an educated Chinese person to correctly write out all characters in the OptSet on papers. According to user investigations, the time and work load are acceptable for them mainly due to the fact that most of those 775 characters are easy to be written and commonly-seen in our daily lives. Until now, several hundreds of users have uploaded their handwritings, which include those 775 characters, to our website. Among them, we choose the handwriting data provided by three users (i.e., User 3, 4, and 5) that have quite different handwriting styles (see Figure 16) as the input data of our system in this group of experiments.

4.2.1 Rendering Results. With the three users' personal handwriting fonts generated by our system, we use them to render two poems and a paragraph in a famous piece of prose. As we can see from Figure 15, in which two poems are rendered using the three font libraries, the quality of machine-generated glyphs, underlined in the figure, is considerably high. Furthermore, both overall handwriting styles and handwriting details of our synthesis results look quite similar to the original handwritings. Figure 16 shows rendering results of a paragraph using four fonts including a font library only consisting of 775 characters written by "User 3" and the three users' complete font libraries generated by our system. As we know, if the selected font library does not include some characters in the text to be rendered, typically the word processor we use will apply a default font style (for example, "Songti") to render those characters. One such kind of example is shown in Figure 16(a), the mixture of two different font styles in one paragraph markedly reduces the readability of rendering results. On the contrary, texts rendered using the three font libraries generated by our system not only have high readability but also look quite similar to real handwritten texts in corresponding personal styles. Results presented here together with other more experimental results we obtained demonstrate clearly that the large-scale handwriting font libraries automatically generated by our system can be directly used in real applications.

4.2.2 Comparison with Deep Learning Based Methods. In this section, we first compare the performance of our approach with two deep learning based end-to-end font synthesizing meth-

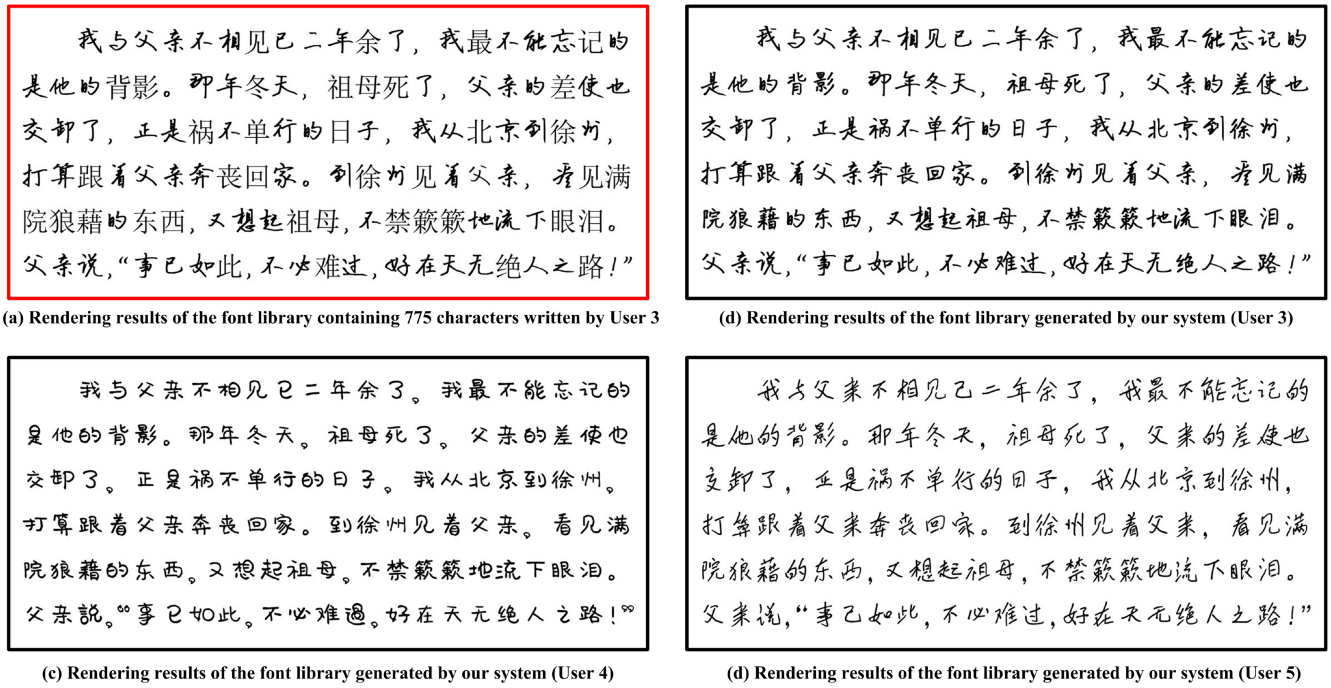
Fig. 15. Rendering results of two famous Chinese poems using three handwriting font libraries generated by our system. Characters underlined are synthesized handwritings while others are written by corresponding users.

ods (i.e., "Rewrite" (Tian 2016) and "pix2pix" (Isola et al. 2017)). We directly utilize the source code provided by the authors to implement the two existing approaches. More specifically, the OptSet consisting of 775 Chinese characters is selected as the training data and the standard "Kaiti" font style is chosen as the input reference for all methods compared here. Namely, during the offline training period, 775 character images in the standard "Kaiti" style are imported as the input of Neural Networks and the corresponding character images in a user's handwriting style are considered to be the ideal target output. Online, all other characters that are not included in the OptSet will be sequentially input into the trained networks to generate synthesized character images in the user's handwriting style.

Figure 17 shows some examples of synthesized results obtained by using our method and the above-mentioned two approaches, respectively. As we can see, "Rewrite" can only synthesize some coarse character shapes for User 4 and fails to generate any reasonable results for the other two users. The method "pix2pix" performs much better than "Rewrite" mainly due to the introduction of adversarial networks that can markedly improve the synthesizing performance of generative networks. However, as we can see from Figure 17, although the details of shapes synthesized by "pix2pix" possess the similar style as the input character images written by corresponding users, most of these synthesized characters are unreadable. On the contrary, synthesis results obtained using our method not only precisely inherit the users' overall and detailed handwriting styles but also clearly represent the correct meanings of corresponding characters. As we know, most of the existing deep learning based approaches including the two methods compared here adopt a so-called end-to-end learning architecture, which relies heavily on the interpreting capability of networks for the training data. The end-to-end architecture has been proved to work perfectly well for global or coarse data interpreting tasks (e.g., classification, detection, segmentation), but is still not able to automatically and precisely interpret high-level and detailed knowledge contained in the training images with elegant and complicated structures. Thus, without the correct interpretation of training character images, there is no way for these end-to-end methods to generate high-quality synthesized handwritings with correct meanings, especially for complex characters.

Although above-mentioned experiments validate the effectiveness and superiority of our system when handling glyphs written by ordinary people, the EasyFont system that only adopts the reference data in the standard "Kaiti" style is still not able to generate satisfactory results for handwritten/designed glyphs with either complicated shapes of outlines or very cursive writing trajectories (see Figure 18). Thereby, during offline processing, another 57 representative GB2312 Chinese fonts are chosen to be used to build the font skeleton manifold. Namely, we need to manually specify the writing trajectory of each stroke for all 6,763 Chinese characters in 85 selected font libraries in various styles. Online, one of these fonts that has the most similar style as the input character images will be chosen as the reference font to replace the "Kaiti" font used in the original system. The style similarity is measured by the sum of shape similarities of corresponding glyph pairs calculated via the same method described in Section 3.4.

To examine the effectiveness of our EasyFont system with reference data in multiple font styles, we conduct experiments on five Chinese font libraries in quite different styles (see Figure 18) and compare our system (EasyFont) with other existing approaches, which can be classified into two categories: Nearest-Neighbor (NN) retrieval based methods (i.e., NN-Fonts and NN-Manifold) and CNN based end-to-end methods (i.e., Rewrite, pix2pix, and zi2zi). NN-Fonts and NN-Manifold are methods whose synthesis results are nearest neighbors retrieved from a given database with the above-mentioned 85 fonts and our font skeleton manifold, respectively. "zi2zi" (Tian 2017) that specifically aims to transfer a glyph image in one font style to another is actually a GAN model modified from "pix2pix." From Figure 18, we can see that Nearest-Neighbor retrieval based methods work poorly due to their high dependency on training data while those CNN based end-to-end methods often synthesize incorrect glyphs especially for characters with complicated shapes due to the reason mentioned above. It can also be observed that the proposed EasyFont system with reference data in multiple font styles is not only suitable for synthesizing handwriting fonts for ordinary people, but also can generate high-quality synthesis results for fonts in artistic styles designed by professionals. However, there still exist some limitations in our system. As we can see from the last row in Figure 18 (please zoom in for better inspection), tiny artifacts still appear in synthesized glyphs for fonts (e.g., "FZQKBYSJW") designed by professionals and some randomly-appeared writing details still cannot be precisely imitated for personal handwriting fonts with cursive styles (e.g., "FZZJ-HJYBXCJW" and "FZZJ-ZSXKJT").



(a) Rendering results of the font library containing 775 characters written by User 3

(d) Rendering results of the font library generated by our system (User 3)

(c) Rendering results of the font library generated by our system (User 4)

(d) Rendering results of the font library generated by our system (User 5)

Fig. 16. Rendering results of a paragraph using four different font libraries. If the font library used does not contain some characters in the paragraph, a default font style (e.g., "Songti") of the word processor we use will be adopted to render those characters (see Figure 16(a)).

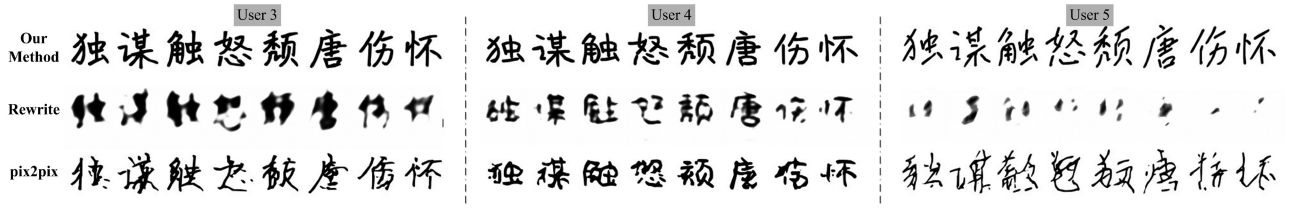


Fig. 17. Comparison of synthesis handwritings generated using our method and two other deep learning based end-to-end approaches (i.e., Rewrite (Tian 2016) and pix2pix (Isola et al. 2017)).

4.2.3 Turing Tests. Finally, in order to quantitatively measure the similarity of styles between human-written characters and synthesized handwritings generated by our system, Turing tests are conducted in this section. Specifically, we built a website that shows a random test paper (see Figure 19 (b)–(d) for some examples of test papers and Figure 19(a) for the corresponding answer paper) for each participant, on which 100 machine-generated characters and 100 human-written characters in a user's personal style are randomly chosen and placed. Meanwhile, 50 randomly-chosen characters written by the user are also displayed to the participant as reference. Each participant is asked to pick out as many characters as possible, which they think are imitated by computers, with sufficient time. Obviously, if the writing style of a machine-generated character is different compared to the original handwriting, it will be quite easy for educated Chinese people to find it. Meanwhile, a machine-generated glyph is hard to be picked out from the test paper if it not only looks like a human-written character but is also similar to the character written by the same person. Therefore, the Turing tests we conduct here can illustrate whether synthesis results possess the required personal handwriting style or not.

We invited 97 educated Chinese people with different ages (from 16–51) and occupations (e.g., students, teachers, company employees) to participate in our Turing tests via the internet. The average accuracy of distinguishing machine-generated characters from original ones is 52.17%, which is close to the accuracy of random guessing (50%), and the 95% confidence interval of the accuracy values is $[52.17\% - 1.74\%, 52.17\% + 1.74\%]$. Results of our Turing tests verify that synthesized handwritings generated by our system are hard to be distinguished from the corresponding user's original handwritings. This is because, as shown in Figure 19, not only the overall style but also many important details of the users' handwritings can be imitated well by the proposed method.

5 DISCUSSION

Randomness exists in every character written by a person; there are even also some people who do not have a stable handwriting style. Basically, the handwriting style captured by many machine

Fig. 18. Comparison of synthesis results generated using our method (EasyFont), NN-Fonts (nearest-neighbor retrieval on a given database with 85 fonts), NN-Manifold (nearest-neighbor retrieval on our font skeleton manifold), and other three deep learning based end-to-end approaches (i.e., Rewrite (Tian 2016), pix2pix (Isola et al. 2017) and zi2zi (Tian 2017)).

(a) Answer paper (b) Test paper (User 5) (c) Test paper (User 4) (d) Test paper (User 3)

Fig. 19. Examples of a region of test papers in three handwriting styles and their corresponding answer paper (a) utilized in Turing tests. Characters in colored blocks are generated by our system and others are human-written characters.

learning methods, such as our approach, is the user's average style (i.e., a kind of statistical handwriting style). It has been reported (Zitnick 2013) that the average of multiple instances of the same handwriting shape typically looks better than most of the individual instances. Therefore, synthesized characters generated by our method are intrinsically well-suited to build font libraries that require more in readability of rendering results and stability of writing styles. However, texts consisting of pure machine-generated characters lack random variations and typically look too uniform compared to real handwritten texts. We solve this problem by combining small amounts of human-written characters with synthesized handwritings for large numbers of other characters to generate the complete font library. Since the average rate of coverage for all characters written by a user is about 50% in normal articles, in this manner, human-written and machine-generated characters will be evenly distributed in an article rendered using the font library generated by our system. That makes the rendering result similar to a real handwritten text.

In our system, the overall handwriting style of a user can be quantified as a set of values measuring the difference between the user's handwritings and corresponding characters in the reference style (e.g., "Kaiti"). By default, the weight of personal handwriting style is selected as $w = 1$ so that synthesized glyphs (e.g., the character within the red square in Figure 20) could have the same style as characters written by the user. Intuitively, we can adjust the style weighting values ($w \geq 0$) to get various synthesis results (see Figure 20). This enables an interesting application in

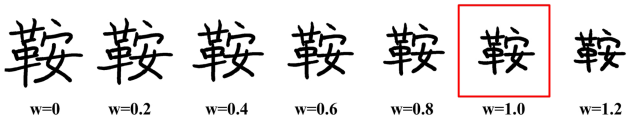


Fig. 20. Synthesized results of a character obtained using our system with different style weighting values for "User 4."

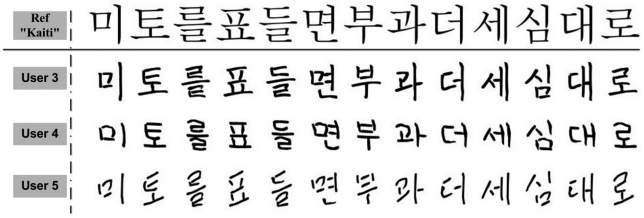


Fig. 21. Synthesized results of some Korean characters in the three users' handwriting styles.

that, after learning a user's handwriting style, our system is able to provide the user with a series of handwriting font products in smoothly changing styles corresponding to different weighting values. Obviously, the larger style weighting values that the user selects, the stronger personal handwriting styles the synthesis results will have. Thus, selecting smaller weighting values could make the style of synthesized handwritings become more similar to the standard "Kaiti" style. People who have ugly handwriting styles might appreciate this function, since they can use it to beautify their handwritings so that personal font libraries that are more visually pleasing can be obtained.

It should be pointed out that although our current method is already able to recover important writing details and handle strange and cursive handwritings, synthesis results generated by our system for very cursive handwriting styles are still not as cursive as the original ones since what the method has learned is the user's average handwriting style. If required, some existing techniques (Lin and Wan 2007) can be easily applied to make those synthesized characters more cursive and thus become more similar as the original ones.

Last but not the least, as mentioned before, our method can be easily extended to many other writing systems. For example, Figure 21 shows some samples of synthesized Korean characters in the three users' handwriting styles obtained by directly using the learned models mentioned in Section 4.2. We believe that the proposed methodology is not only suitable for the generation of Chinese handwriting fonts, but can also be utilized to easily build large-scale handwriting font libraries in other languages.

6 CONCLUSION

This article presented a novel system that is able to learn the handwriting style from a small number of input samples written by an ordinary person and generate the personal handwriting font library, which can have arbitrarily large numbers of Chinese characters, for the user. Experimental results demonstrated that our system can be used to automatically generate high-quality handwriting font libraries, which include huge amounts of machine-generated characters that are indistinguishable from original handwritings. In the future, we are planing to further improve the synthesis performance by integrating powerful deep learning approaches with professional domain knowledge on calligraphy.

ACKNOWLEDGMENTS

We would like to especially thank Prof. Paul L. Rosin from Cardiff University, UK for valuable discussions and helping us improve the presentation quality of this article.

REFERENCES

M. Arjovsky, S. Chintala, and L. Bottou. 2017. Wasserstein GAN. *arXiv preprint* arXiv:1701.07875 (2017).
S. Baluja. 2016. Learning typographic style. *CoRR* abs/1603.04000 (2016). http://arxiv.org/abs/1603.04000.
W. Baxter and N. Govindaraju. 2010. Simple data-driven modeling of brushes. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 135–142.
E. Bernhardsson. 2016. Analyzing 50k fonts using deep neural networks. Retrieved from https://erikbern.com/2016/01/21/analyzing-50k-fonts-using-deep-neural-networks/.
N. D. F. Campbell and J. Kautz. 2014. Learning a manifold of fonts. *ACM Transactions on Graphics* 33, 4 (2014), 91.
X. Chen, Z. Lian, Y. Tang, and J. Xiao. 2017. An automatic stroke extraction method using manifold learning. In *Proc. Eurographics 2017 Short Paper.*
Já. Dolinsky and H. Takagi. 2009. Analysis and modeling of naturalness in handwritten characters. *IEEE Transactions on Neural Networks* 20, 10 (2009), 1540–1553.
J. Dong, M. Xu, and Y. Pan. 2008. Statistic model-based simulation on calligraphy creation. *Chinese Journal of Computers* 31, 7 (2008), 1276–1282 (In Chinese).
J. L. Elman. 1990. Finding structure in time. *Cognitive Science* 14, 2 (1990), 179–211.
J. Fan. 1990a. Intelligent Chinese character design and an experimental system ICCDS. *JCIP* 4, 3 (1990), 1–11 (In Chinese).
J. Fan. 1990b. A method of computerizing the calligraphical rules basing on CC structure code. *JCIP* 4, 4 (1990), 43–52 (In Chinese).
FontCreator. 2017. High logic. Retrieved from http://www.high-logic.com/.
FontLab. 2017. Fontlab. Retrieved from http://www.fontlab.com/.
Founder. 2017. Founder group. Retrieved from http://www.foundertype.com/.
L. A. Gatys, A. S. Ecker, M. Bethge, S. Hertzmann, and E. Shechtman. 2017. Controlling perceptual factors in neural style transfer. In *Proc. CVPR 2017.*
I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. Generative adversarial networks. In *Proc. NIPS 2014.*
T. S. F. Haines, M. Aodha, and G. J. Brostow. 2016. My text in your handwriting. *ACM Transactions on Graphics (TOG)* 35, 3 (2016), 26.
K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Proc. CVPR 2016.* 770–778.
G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507.
S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
P. Isola, J. Y. Zhu, T. Zhou, and A. A. Efros. 2017. Image-to-image translation with conditional adversarial nets. In *Proc. CVPR 2017.*
B. K. Jang and R. T. Chin. 1990. Analysis of thinning algorithms using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12, 6 (1990), 541–551.
L. Jin and X. Gao. 2004. Study of several handwritten Chinese character directional feature extraction approaches. *Application Research of Computers* 21, 11 (2004), 38–40.
H. Khosravi and E. Kabir. 2010. Farsi font recognition based on Sobel–Roberts features. *Pattern Recognition Letters* 31, 1 (2010), 75–82.
A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25.* 1097–1105.
P. K. Lai, D. Y. Yeung, and M. C. Pong. 1996. A heuristic search approach to Chinese glyph generation using hierarchical character composition. *Computer Processing of Oriental Languages* 10, 3 (1996), 307–323.
B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350, 6266 (2015), 1332–1338.
N. Lawrence. 2005. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research* 6, Nov (2005), 1783–1816.
W. Li, Y. Song, and C. Zhou. 2014. Computationally evaluating and synthesizing Chinese calligraphy. *Neurocomputing* 135, 5 (2014), 299–305.
Z. Lian and J. Xiao. 2012. Automatic shape morphing for Chinese characters. In *Proc. SIGGRAPH Asia 2012 TB.* 2.

Z. Lian, B. Zhao, and J. Xiao. 2016. Automatic generation of large-scale handwriting fonts via style learning. In *Proc. SIGGRAPH Asia 2016 TB*. 12.
J. Lin, C. Wang, C. Ting, and R. Chang. 2014. Font generation of personal handwritten Chinese characters. In *Proc. IGIP 2014*.
Z. Lin and L. Wan. 2007. Style-preserving English handwriting synthesis. *Pattern Recognition* 40, 7 (2007), 2097–2109.
J. Long, E. Shelhamer, and T. Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proc. CVPR 2015*. 3431–3440.
J. Lu, C. Barnes, S. DiVerdi, and A. Finkelstein. 2013. RealBrush: Painting with examples of physical media. In *Proc. ACM SIGGRAPH 2013*.
J. Lu, C. Barnes, C. Wan, P. Asente, R. Mech, and A. Finkelstein. 2014. DecoBrush: Drawing structured decorative patterns by example. In *Proc. ACM SIGGRAPH 2014*.
J. Lu, F. Yu, A. Finkelstein, and S. DiVerdi. 2012. HelpingHand: Example-based stroke stylization. In *Proc. ACM SIGGRAPH 2012*.
A. Myronenko and X. Song. 2010. Point set registration: Coherent point drift. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 12 (2010), 2262–2275.
W. Pan, Z. Lian, R. Sun, Y. Tang, and J. Xiao. 2014. FlexiFont: A flexible system to generate personal font libraries. In *Proc. DocEng 2014*. 17–20.
H. Q. Phan, H. Fu, and A. B. Chan. 2015. FlexyFont: Learning transferring rules for flexible typeface synthesis. *Computer Graphics Forum* 34, 7 (2015), 245–256.
D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–536.
O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. 2015. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.
D. Silver, A. Huang, and C. J. Maddison. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.
K. Simonyan and A. Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *CoRR* abs/1409.1556 (2014).
A. A. Soltani, H. Huang, J. Wu, T. Kulkarni, and J. Tenenbaum. 2017. Synthesizing 3D shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *Proc. CVPR 2017*.
S. Strassmann. 1986. Hairy brushes. In *Proc. ACM SIGGRAPH 1986*, Vol. 20. 225–232.
Z. Sun, L. Jin, Z. Xie, Z. Feng, and S. Zhang. 2016. Convolutional multi-directional recurrent network for offline handwritten text recognition. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 240–245.
R. Suveeranont and T. Igarashi. 2010. Example-based automatic font generation. In *Proc. Smart Graphics*. 127–138.
Y. Tian. 2016. ReWrite. Retrieved from https://github.com/kaonashi-tyc/Rewrite/.
Y. Tian. 2017. ReWrite. Retrieved from https://github.com/kaonashi-tyc/zi2zi/.
Y. Wang, H. Wang, C. Pan, and L. Fang. 2008. Style preserving Chinese character synthesis based on hierarchical representation of character. In *Proc. ICASSP 2008*. 1097–1100.
Z. Wang and Y. Pang. 1991. A computer calligraphy system CCCS. *Journal of Computer Aided Design and Computer Graphics* 3, 1 (1991), 35–40 (In Chinese).
S. T. Wong, H. Leung, and H. H. S. Ip. 2008. Model-based analysis of Chinese calligraphy images. *Computer Vision and Image Understanding* 109, 1 (2008), 69–85.
W. Xia and L. Jin. 2009. A Kai style calligraphic beautification method for handwriting chinese character. In *Proc. ICDAR 2009*. 798–802.
S. Xu, H. Jiang, T. Jin, F. Lau, and Y. Pan. 2008. Automatic facsimile of Chinese calligraphic writings. In *Computer Graphics Forum*, Vol. 27. 1879–1886.
S. Xu, H. Jiang, F. C. M. Lau, and Y. Pan. 2007. An intelligent system for Chinese calligraphy. In *Proc. The National Conference on Artificial Intelligence*, Vol. 22. 1578.
S. Xu, T. Jin, H. Jiang, and F. C. M. Lau. 2009. Automatic generation of personal chinese handwriting by capturing the characteristics of personal handwriting. In *Proc. IAAI 2009*.
S. Xu, F. Lau, F. Tang, and Y. Pan. 2003. Advanced design for a realistic virtual brush. In *Computer Graphics Forum*, Vol. 22. 533–542.
S. Xu, F. C. M. Lau, W. K. Cheung, and Y. Pan. 2005. Automatic generation of artistic Chinese calligraphy. *IEEE Intelligent Systems* 20, 3 (2005), 32–39.
T. Yi, Z. Lian, Y. Tang, and J. Xiao. 2014. A data-driven personalized digital ink for Chinese characters. In *Proc. MultiMedia Modeling 2014*. 254–265.
K. Yu, J. Wu, and Y. Zhuang. 2009. Style-consistency calligraphy synthesis system in digital library. In *Proc. the 9th ACM/IEEE-CS Joint Conference on Digital Libraries*. 145–152.
Z. Zhang, C. Zhang, W. Shen, C. Yao, W. Liu, and X. Bai. 2016. Multi-oriented text detection with fully convolutional networks. In *Proc. CVPR 2016*. 4159–4167.
B. Zhou, W. Wang, and Z. Chen. 2011. Easy generation of personal chinese handwritten fonts. In *Proc. ICME 2011*. 1–6.
C. L. Zitnick. 2013. Handwriting beautification using token means. In *Proc. ACM SIGGRAPH 2013*.

Received June 2017; revised September 2018; accepted September 2018