Learning Adaptive Basis Fonts to Fuse Content Features for Few-shot Font Generation

Keyang Lin[1], Zhijun Fang[1,2*], Sicong Zang[1*], Hang Wu[1]
[1] School of Computer Science and Technology, Donghua University, Shanghai, China
lin_ke_yang@163.com, {zjfang, sczang}@dhu.edu.cn, 1439111092@qq.com
[2] School of Electronic and Electrical Engineering, Shanghai University Of Engineering Science, Shanghai, China

Abstract

Few-shot font generation aims to generate all characters of a certain font by using its very few seen characters as references. Recent studies assumed that a target font can be regarded as a mixture of several source fonts, namely basis fonts, and the style of the target font can be generated by combining several latent representations respectively captured from a group of other fonts with given weights. However, these basis fonts along with their corresponding mixing weights are unlearnable as they are determined by a pre-trained network for font feature extraction. The styles in their basis fonts also have significant differences so it is not flexible to fusion them. In this paper, we present an adaptive basis fonts and weights learning (ABW) module to learn more appropriate basis fonts and weights, making the results more similar to the target. We project the reference characters into a Gaussian Mixture Model (GMM) distributed latent space, where each latent Gaussian collects latent representations of fonts with similar styles. A Rival Penalized Competitive Learning (RPCL) enhanced Expectation Maximization (EM) like learning algorithm is introduced to learn the GMM-structured latent space, jointly with the training of the network. The ABW module is also able to automatically determine an appropriate number of Gaussians in GMM, i.e., the number of basis fonts, making font mixing more flexible. We collect a dataset of 500 Chinese fonts with 6.5k characters each and evaluate our method on it. Experimental results demonstrate that our method outperforms recent few-shot font generation methods.

Keywords: Font generation, few-shot learning, content fusion, Gaussian mixture model

*Corresponding author.

1. Introduction

A font is a collection of characters with a certain style, and it is used everywhere in our production and life. Traditional font design work is time-consuming and laborious. It relies heavily on professional designers, especially in writing systems that have numerous characters, such as Chinese (over 60K characters), Japanese (over 50K characters), and Korean(over 11K characters). Few-shot font generation aims to automatically generate all characters of a certain font only using very few given characters as references. It can greatly improve the efficiency of font design work, and it also has important significance in font library creation, personalized font generation, historical handwriting imitation, handwriting identification, and the restoration of ancient calligraphy relics.

With the development of neural networks, a large number of methods have been applied in the field of font generation. Early methods [1, 2, 26, 32] are usually based on improvements to traditional Convolutional Neural Networks (CNN) and Generative Adversarial Networks (GAN) [12]. These methods can only generate seen fonts (fonts that have appeared in the training set), but cannot handle the style of unseen fonts (fonts that have not appeared in the dataset). They also only support style transfer between two domains, and need to train many models when generating multiple fonts. Another disadvantage is that they require a large amount of data, and some require strictly paired data that is difficult to collect. The paired data means that, for a certain character, it is necessary to input its source font image and target font image together during training to meet the requirement of the early image-to-image translation methods. All of these limit their practical applications.

To address these issues, SA-VAE [31] and EMD [44] separate the representations of style and content, and many following methods [38, 5, 28, 24, 27, 33, 21] verify that it is an effective approach in the font generation task. On this basis, as the style information can be extracted from only a few reference characters, it is possible to complete the few-

shot font generation task. In these methods, the input of the content feature is a manually selected source font. Since it is challenging to achieve a complete disentanglement between the content and style features [18, 22], the style information of the content features is always mixed to a certain extent, which makes the selection of the source font have a significant impact on the generated results.

A prominent task is CF-Font [35], which uses multiple fonts (basis fonts) in the training set to replace the source font and weight them at multiple scales of the content encoder outputs. So that the style information in the content features will be closer to the target style, resulting in better scores. However, their approach still has some limitations. Specifically, They calculate the L1 distance between the few-shot content feature maps each other and then use k-means clustering on the concatenated vectors to select the basis fonts. Their basis fonts and content fusion weights are no longer changed once they are calculated. All of the above issues may lead their method to be stuck at local optimal.

In this paper, we propose adaptive basis fonts and weights learning Font (ABW-Font). It is a few-shot font generation method and can generate a new font using only several given reference characters. Our ABW module can learn more appropriate basis fonts and content fusion weights to make the results more similar to the target. We project few-shot reference character images into a Gaussian Mixture Model (GMM) distributed latent space, where each Gaussian component represents a generalized font style. The font with the minimum L2 distance from the Gaussian centers will be selected as the basis fonts. Our content fusion weights are obtained by a softmax operation on the L2 distance between the content feature of the current font and all basis fonts. Inspired by [42], we adopt its Rival Penalized Competitive Learning (RPCL) [40] enhanced Expectation Maximization (EM) like learning algorithm to iterate them, which not only makes the basis fonts and content fusion weights learnable via the network training but also automatically determines an appropriate number of basis fonts. To summarize, we make the following contributions:

1. We proposed an adaptive basis fonts and weights learning module to make basis fonts and content fusion weights learnable, yielding a better content fusion.

2. We add a compact clustering loss term to make the content vector closer to the Gaussian centers, making the network training more stable and accelerating the convergence of the network.

3. We collect a dataset containing 500 Chinese fonts and evaluate our method on it. Experimental results demonstrate our method outperforms recent few-shot font generation methods.

2. Related works

The early font generation task is related to the image-to-image translation (I2I) task, which tries to preserve the content of the source font while merging the style of the target domain at the same time. Many I2I methods[15, 7, 30, 36, 45, 8] are used to achieve the font generation task. As some early I2I methods can only learn the mapping between two domains and they rely on a lot of paired data, CycleGAN[45] is a noticeable work. It introduces the cycle consistency into generative models and enables I2I methods to train cross-domain translation without paired data.

For few-shot font generation, many I2I methods[4, 19, 14, 6, 23, 3] have provided references. FUNIT[23] accomplishes the style transfer task by encoding content and style respectively and combining them with adaptive instance normalization (AdaIN)[14]. Recent studies mainly focus on disentangling content and styles from the target characters, enriching character components with structure information, or mixing the selected basis fonts by content fusion.

2.1. Disentangling content and style

Learning content features and style features separately is the foundation of few-shot font generation methods. In early font generation methods [1, 2, 26, 32], they learn the mapping relationship between the source font domain and the target font domain, and rely on strictly paired data for training. Also, they can not generate the unseen font. To solve this problem, EMD [44], SA-VAE [31], and DC-Font [16] disentangle the style and content representations. Style features can be extracted from only a few samples, which provides the network possibility to achieve the few-shot font generation task. On this basis, SC-Font [17] transfers the writing trajectories with separated strokes in the reference font style into those in the target style and renders synthesized skeletons of characters with a specific handwriting style via a GAN model. Gao and Wu [11] proposed a stacked cGAN model for unpaired multi-chirography Chinese character image translation based on skeleton transformation and stroke rendering. DG-Font[39] designs a network structure mainly composed of deformable convolutions[9, 46], which more effectively utilize universal information.

2.2. Enriching character components with structure information

A Chinese character can be separated into several components by its structure information. Many methods enriched character components with their semantic features by prior knowledge to improve font generation performance. CalliGAN [38] is the first to decompose Chinese characters into components and encode them through a recurrent neural network. DM-Font [5] employs the dual memory

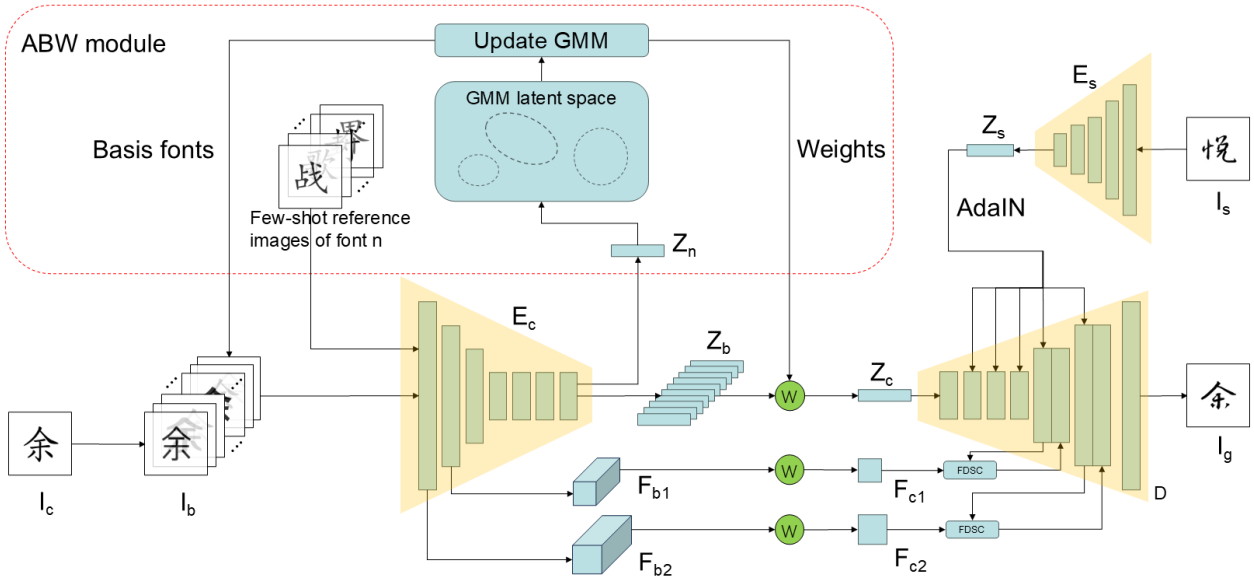

Figure 1. The overview of ABW-Font, which consists of a content encoder E_c, a style encoder E_s, a decoder D and an ABW module for learning basis fonts and content fusion weights in a GMM-structured latent space. I_C is the content image, It will be replaced by I_b, several images that have the same content but from the current basis fonts. E_C extract I_b and outputs muti-scaled features Z_b, F_{b1} and F_{b2}. Calculating the weighted sum of Z_b, F_{b1} and F_{b2} based on the content fusion weights obtained by ABW module to get muti-scaled content feature Z_c, F_{c1} and F_{c2}. The FDSC module receives a concatenation of two feature maps and outputs a new one through a deformable convolution layer. E_S extract the style image I_S and output the style feature Z_S. D mixes the content feature and the style feature. Finally, D outputs the generated font image with the content of I_C but the style of I_S.

structure (persistent memory and dynamic memory) to efficiently capture the global glyph structure and the local component-wise styles for Korean and Thai fonts. LF-Font [27] learn to disentangle complex glyph structures and use localized style representations, instead of universal style representations. MX-Font [28] extracts multiple style features by multiple experts, each of which is guided by utilizing component labels as weak supervision to be specialized for different local concepts. XMP-Font [24] propose a self-supervised cross-modality pre-training strategy and a cross-modality transformer-based encoder that is conditioned jointly on the glyph image and the corresponding stroke labels. Fs-Font [33] learn fine-grained local styles and combine cross-attention mechanisms[34] and global context awareness to get a higher score. CG-GAN [21] introduces a Component-Aware Module which enables content-style decoupling at a finer granularity, specifically at the component level, thereby providing more precise guidance to the generator. VQ-Font [41] proposes a VQGAN-based[10] framework to enhance glyph fidelity through token prior refinement and structure-aware enhancement. However, although component-level supervision can improve generation quality, annotating components relies on prior knowledge and is a labor-intensive task. Besides, component labels ensure the correctness of the structure, thereby limiting the degree of deformation. When generating fonts with strong artistic styles and large deformations (such as cursive script), it will affect the generation effect.

2.3. Mixing Basis Fonts by Content Fusion

Some studies assumed that a target font can be regarded as a mixture of several source fonts, namely basis fonts, and the style of the target font can be generated by combining several latent representations respectively captured from a group of other fonts with given weights. CF-Font [35] design a content fusion module to represent the source font as a weighted sum of basis fonts. They make the style in content features similar to the target, greatly reducing the dependence on selecting the source font. However, their methodology exhibits some limitations. They calculate the L1 distance between the few-shot content feature maps each other and then use k-means clustering on the concatenated vectors to select the basis fonts. That makes the vector representing one font contain information about other fonts. Their basis fonts and content fusion weights are also fixed once calculated so they can not adapt to the network updating. All of these might make their method not flexible to generate results similar to the target. Starting from this perspective, we propose ABW-Font to make the basis fonts and content fusion weights learnable.

3. Method

Our network is shown in Fig. 1. Firstly, we train the neural network without ABW module to to ensure that our network has the ability to extract features from character images, thereby ensuring that the subsequent ABW module can perform clustering operations on reasonable features. Then the training of our ABW-Font can be devided into 2 phases: in phase 1, the content encoder receives few-shot reference images of all training fonts, projects them into a GMM distributed latent space, and calculates the posterior probability of each font belonging to these distributions. We select the fonts that have the minimum L2 distance to the Gaussian centers as the basis fonts, and obtain the content fusion weights after a softmax operation on the L2 distance between the content feature of the current font and all basis fonts. We adopt the RPCL enhanced EM-like learning algorithm from [42] to iterate the above process. In phase 2, the content image will be replaced by images that represent the same character but from the basis fonts. The outputs of the encoder at different scales will be weighted and summed. Then they are sent to the decoder. Style encoder extracts the style image and adds it to the decoder through AdaIN [14].

3.1. Model Structure

The ABW-Font consists of a content encoder E_c, a style encoder E_s, a decoder D, and an ABW module for learning basis fonts and content fusion weights in a GMM-structured latent space. Our model is based on DG-Font and CF-Font. We keep the U-net[29] structure and use 2 feature deformation skip connection[39], which followed the approach of DG-Font. Differently, we have modified the shape of the output features of the content encoder. In DG-Font and CF-Font, the output of their E_c is a 256×20×20 feature map. Its shape is so large for clustering. Therefore CF-Font calculates the L1 distance between the few-shot content feature maps each other first, and then uses k-means clustering on the obtained vectors. This means the vector representing one font contains information about other fonts. In ABW-Font, the output of I_c is a 128 vector. For the 16 few-shot reference images representing a certain training font, we can directly perform the clustering operation on the concatenated vectors, which ensures the feature representing a certain font do not contain information from other fonts, thus maintaining the rationality of clustering.

3.2. Learning Adaptive Basis and Weights

Our ABW module aims to get the learnable basis fonts and content fusion weights. It is the core of the phase 1 in the training process. We project few-shot reference character images of all training fonts into a GMM distributed latent space and divide them into several categories. Then we calculate the posterior probabilities that each font belongs to different Gaussian distributions. We use the fonts with the minimum L2 distance to each Gaussian center as the basis fonts, and use the output after a softmax operation on the L2 distance between the content feature of the current font and all basis fonts for later content feature weighted sum operation. During training, the GMM will be iterated via an RPCL enhanced EM algorithm, and the basis fonts and content fusion weights will also change accordingly. Specifically, our ABW module is implemented in the following:

For the number n train font, the content encoder E_C accepts its 16 few-shot reference character images and outputs a concatenated vector $\boldsymbol{z}_n$. So for all N kinds of fonts in the training set, we get a content feature vector set $\{\boldsymbol{z}_n\}_{n=1}^N$.

Initialization. We need to calculate the basis fonts and content fusion weights at the beginning of training for the initialization of our GMM. We perform a traditional k-means clustering on the content feature vector set $\{\boldsymbol{z}_n\}_{n=1}^N$ and get M clusters corresponding to Eq. (1).

$$\{C_i\}_{i=1}^M = k\text{-means}(\{\boldsymbol{z}_n\}_{n=1}^N). \tag{1}$$

Each $\boldsymbol{z}_n$ can be regarded as a point from a cluster in high-dimensional space. Since a high-dimensional Gaussian distribution can be determined by its mean and covariance matrix, we need to calculate the means and covariance matrices of these clusters to construct the GMM. For the number i cluster C_i, we use all points that belonged to C_i to calculate the mean vector $\boldsymbol{\mu}_i^{(0)}$ and the covariance matrix $\boldsymbol{\Sigma}_i^{(0)}$ according to Eq. (2) .

$$\boldsymbol{\mu}_i^{(0)}, \boldsymbol{\Sigma}_i^{(0)} = f_{\boldsymbol{\mu},\boldsymbol{\Sigma}}(\{\boldsymbol{z}_n | \boldsymbol{z}_n \in C_i\}), \tag{2}$$

where $f_{\boldsymbol{\mu},\boldsymbol{\Sigma}}$ indicates the operation of calculating the mean vector and the covariance matrix.

We assume that the features of each dimension are independent of each other, so we can use the diagonal matrix of $\boldsymbol{\Sigma}_i^{(0)}$ to simplify the calculation. We obtain M means and covariance matrices that determine M high-dimensional Gaussian distributions. Then we calculated the L2 distance between $\{\boldsymbol{z}_n\}_{n=1}^N$ and these M Gaussian centers. As described in Eq. (3), we choose the fonts represented by $\boldsymbol{z}_n$ that have the minimum L2 distance to M Gaussian distribution centers as the initial basis fonts. B_i indicates the number of the font that is selected to be the basis font of C_i.

$$B_i^{(0)} = \arg\min_n(||\boldsymbol{z}_n - \boldsymbol{\mu}_i^{(0)}||_2^2). \tag{3}$$

Next, we need to get the weights of all fonts in the training set for the content feature fusion operation in phase 2.

The weights of font n are calculated by Eq. (4).

$$\begin{aligned} \boldsymbol{d}^{(0)} &= (d_1^{(0)}, d_2^{(0)}, \ldots, d_M^{(0)}), \quad d_i^{(0)} = ||\boldsymbol{z}_n - \boldsymbol{\mu}_i^{(0)}||_2^2, \\ \boldsymbol{w}_n^{(0)} &= \text{softmax}(\frac{-\boldsymbol{d}^{(0)}}{t}), \end{aligned} \tag{4}$$

where t is the temperature of the softmax operation.

Through the above process, the basis fonts $\{B_i^{(0)}\}_{i=1}^M$ and the content fusion weights $\{\boldsymbol{w}_n^{(0)}\}_{n=1}^N$ needed in stage 2 of the training process have been obtained. When the network performs stage 1 in the next execution, an EM process will be used to update our GMM and obtain new basis fonts and content fusion weights.

E-step. When we are in phase 1 at time τ, we start the Expectation step (E-step) of our RPCL enhanced EM algorithm. We first calculated the conditional probability of the few-shot reference character point $\boldsymbol{z}_n$ being assigned to C_i based on Eq. (5).

$$\begin{aligned} P^{(\tau-1)}(\boldsymbol{z}_n|C_i) &= \frac{1}{(2\pi)^{\frac{d}{2}} \cdot |\boldsymbol{\Sigma}_i^{(\tau-1)}|^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}\boldsymbol{\kappa}}, \\ \boldsymbol{\kappa} &= \left(\boldsymbol{z}_n - \boldsymbol{\mu}_i^{(\tau-1)}\right)^T \left(\boldsymbol{\Sigma}_i^{(\tau-1)}\right)^{-1} \left(\boldsymbol{z}_n - \boldsymbol{\mu}_i^{(\tau-1)}\right). \end{aligned} \tag{5}$$

Then, following the Eq. (6), we calculate the posterior probability that $\boldsymbol{z}_n$ belongs to C_i according to the prior probability $P^{(\tau-1)}(C_i)$:

$$\begin{aligned} \gamma_{ni}^{(\tau)} &= P^{(\tau)}(C_i|\boldsymbol{z}_n) \\ &= \frac{P^{(\tau-1)}(\boldsymbol{z}_n|C_i) \cdot P^{(\tau-1)}(C_i)}{\sum_{i=1}^M [P^{(\tau-1)}(\boldsymbol{z}_n|C_i) \cdot P^{(\tau-1)}(C_i)]}. \end{aligned} \tag{6}$$

We add an RPCL operation before we update the GMM and the prior probability $P(C_i)$ according to Eq. (7).

$$\tilde{\gamma}_{ni}^{(\tau)} = \begin{cases} 1, \text{ winner: } i = i^*, i^* = \arg\max_i \gamma_{ni}^{(\tau)}; \\ -\zeta, \text{ rival: } i = u, u = \arg\max_{i \neq i^*} \gamma_{ni}^{(\tau)}; \\ 0, \text{ otherwise.} \end{cases} \tag{7}$$

In Eq. (7), the winner and rival indicate the cluster with the largest and second-largest probabilities, respectively, that font n belongs to. The basic idea of RPCL is that for each input not only the winner is modified to adapt to the input, but also its rival is delearned by a smaller learning rate. Our motivation for using RPCL instead of conventional EM is to avoid the gradual overlap of different distributions during training to keep the basis fonts containing as many generalized styles as possible. Also, RPCL is able to automatically determine an appropriate cluster number which weakens the influence of selecting the basis fonts numbers M.

M-step. Then we execute the Maximization step (M-step) of EM, updating the Gaussian distribution and the prior probability $P(C_i)$ according to Eq. (8) to (10).

$$\tilde{\boldsymbol{\mu}}_i^{(\tau)} = \frac{\sum_n \boldsymbol{z}_n \cdot \tilde{\gamma}_{ni}^{(\tau)}}{\sum_n \tilde{\gamma}_{ni}^{(\tau)}}, \tag{8}$$

$$\tilde{\boldsymbol{\Sigma}}_i^{(\tau)} = \frac{\sum_n [(\boldsymbol{z}_n - \tilde{\boldsymbol{\mu}}_i^{(\tau)})^2 \cdot \tilde{\gamma}_{ni}^{(\tau)}]}{\sum_n \tilde{\gamma}_{ni}^{(\tau)}}, \tag{9}$$

$$\tilde{P}^{(\tau)}(C_i) = \frac{\sum_n \tilde{\gamma}_{ni}^{(\tau)}}{N}. \tag{10}$$

We adopt a smooth update strategy to reduce the fluctuations in our GMM parameter updates and try to avoid getting stuck in local optima. We do not directly replace the old parameters with new ones but update them with a learning rate η using Eq. (11) to (13).

$$\mu_i^{(\tau)} = (1-\eta)\tilde{\mu}_i^{(\tau-1)} + \eta\mu_i^{(\tau)}, \tag{11}$$

$$\Sigma_i^{(\tau)} = (1-\eta)\tilde{\Sigma}_i^{(\tau-1)} + \eta\Sigma_i^{(\tau)}, \tag{12}$$

$$P^{(\tau)}(C_i) = (1-\eta)\tilde{P}^{(\tau-1)}(C_i) + \eta\, P^{(\tau)}(C_i). \tag{13}$$

Now we can calculate the basis fonts and content fusion weights at time τ using the new Gaussian centers:

$$\begin{aligned} B_i^{(\tau)} &= \arg\min_n (||\boldsymbol{z}_n - \boldsymbol{\mu}_i^{(\tau)}||_2^2), \\ \boldsymbol{d}^{(\tau)} &= (d_1^{(\tau)}, d_2^{(\tau)}, \ldots, d_M^{(\tau)}), \quad d_i^{(\tau)} = ||\boldsymbol{z}_n - \boldsymbol{\mu}_i^{(\tau)}||_2^2, \\ \boldsymbol{w}_n^{(\tau)} &= \text{softmax}(\frac{-\boldsymbol{d}^{(\tau)}}{t}). \end{aligned} \tag{14}$$

Finally, we have the basis fonts $\{B_i^{(\tau)}\}_{i=1}^M$ and the content fusion weights $\{\boldsymbol{w}_n^{(\tau)}\}_{n=1}^N$. They will be used for feature fusion in phase 2 of the training. When the next time phase 1 is activated, Eq. (5) to (14) will be executed to repeat this process. When generating unseen font during the inference, the content fusion weights can be calculated using the few-shot reference character images of the unseen characters and the Gaussian centers $\{\boldsymbol{\mu}_i\}_{n=1}^M$ according to Eq. (14).

3.3. Training an ABW-Font

Given the content image I_c and style image I_s, the network finally output the generated image I_g. We first train a base network without ABW module to ensure that the model has the ability to extract content features for our ABW module initialization. When training the base network, as the basis fonts and weights do not yet exist, I_c will be directly input to E_c without being replaced by I_b. Similarly, E_c will directly output Z_c, F_{c1}, and F_{c2}.

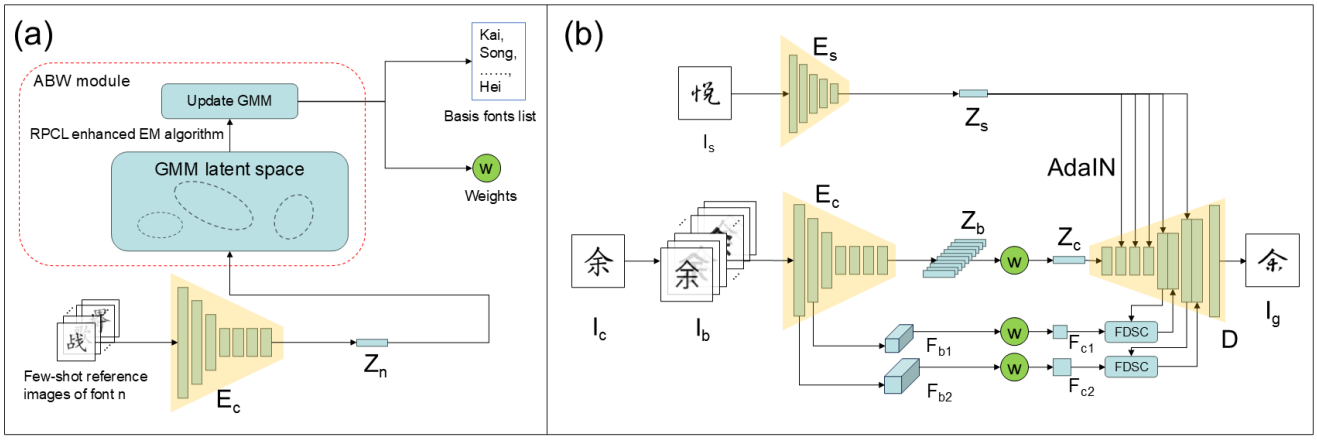

Figure 2. Two phases of the ABW-Font training. (a) shows the phase 1: The content encoder extracts few-shot reference characters of font n and outputs the concatenated feature vector Z_n. The ABW module accepts the few-shot reference vectors and outputs the current basis fonts and content fusion weights. (b) shows the phase 2: I_C is the content image, It will be replaced by I_b, several images that have the same content but from the current basis fonts. E_C extract I_b and outputs muti-scaled features Z_b, F_{b1} and F_{b2}. Calculating the weighted sum of Z_b, F_{b1} and F_{b2} based on the content fusion weights obtained by ABW module to get muti-scaled content feature Z_c, F_{c1} and F_{c2}. The FDSC module receives a concatenation of two feature maps and outputs a new one through a deformable convolution layer. E_S extract the style image I_S and output the style feature Z_S. D mixes the content features and the style features. Finally, D outputs the generated font image which has the content of I_C but has the style of I_S.

Then we add our ABW module, and the training details are divided into two phases as Fig. 2 shows: In phase 1: The content encoder first accepts 16 few-shot reference character images of all N train fonts and sends the output latent codes to the ABW module. The ABW module will project them to a GMM-distributed latent space and use an RPCL enhanced EM algorithm to iterate them. At last, the ABW module outputs M basis fonts and N content fusion weights $\boldsymbol{w}$. They will be used in phase 2. Phase 2 will be executed once every 500 iterations to prevent overfitting and meanwhile ensure the training speed is not too slow.

In phase 2: The content image I_c will be replaced by M images I_b which reference the same character as I_c but from basis fonts. The content encoder outputs at multiple scales to get the vector Z_b, and the feature maps F_{b1} andF_{b2}. They will be weighted and summed base on w, and we get a vector Z_c and two feature maps F_1 and F_2. The feature deformation skip connection (FDSC) is from DG-Font. It receives a concatenation of two feature maps and outputs a new one through a deformable convolution layer. The style encoder transfers I_s to a vector Z_s and adds it to some layers of the decoder through AdaIN.

Additionally, we add a compact clustering loss term $\mathcal{L}_{CCL}$ to clearly classify the latent codes $\boldsymbol{z}_n$ into a certain cluster, making the initial training stages more stable. This loss term can enable points at multiple class boundaries to be more clearly classified into a certain class, thereby preventing these points from being classified into different classes during the RPCL enhanced EM process and causing significant fluctuations in GMM. For the number n train font, we calculate the L2 distance between $\boldsymbol{z}_n$ and the Gaussian center $\boldsymbol{\mu_i}$ corresponding to the maximum term in M values in w_n. The compact clustering loss term $\mathcal{L}_{CCL}$ can be calculated as follows:

$$\mathcal{L}_{CCL} = \Sigma_{n=1}^{N} ||\boldsymbol{z}_n - \text{sg}(\boldsymbol{\mu}_j)||_2^2, \quad j = \arg\max(w_n^{(\tau)}), \tag{15}$$

where sg$(\cdot)$ stands for the stop-gradient operator. It reduces the distance between the few-shot reference character latent codes and their nearest Gaussian center, thereby separating the points of different clusters and allowing the selected basis fonts to represent better style information. Adding $\mathcal{L}_{CCL}$ to the generator loss, we have the following overall loss function for training:

$$\begin{aligned}\mathcal{L} =& \mathcal{L}_{\text{adv}} + \lambda_{\text{img}} \left(\mathcal{L}_{\text{img}} + \lambda_{\text{pcl}} \mathcal{L}_{\text{pcl}}\right) + \lambda_{\text{offset}} \mathcal{L}_{\text{offset}} \\ &+ \lambda_{\text{CCL}} \mathcal{L}_{\text{CCL}},\end{aligned} \tag{16}$$

where $\mathcal{L}_{\text{adv}}$ is the adversarial loss in GAN [12]. $\mathcal{L}_{\text{img}}$ is image reconstruction loss to ensure the generator can reconstruct the source image when given its origin style.$\mathcal{L}_{\text{pcl}}$ is the projected character loss to better supervise the skeleton [35]. $\mathcal{L}_{\text{offset}}$ is the deformation offset normalization to avoid excessive offsets in the deformable convolution in FDSC [39]. λ_{img}, λ_{offset}, λ_{pcl}, and λ_{CCL} are coefficients

corresponding to the loss terms. The tuning strategy is to ensure that these losses are on a reasonable order of magnitude. Specifically, $\mathcal{L}_{\text{adv}}$ should dominate, followed by $\lambda_{\text{img}}\left(\mathcal{L}_{\text{img}}+\lambda_{\text{pcl}}\mathcal{L}_{\text{pcl}}\right)$ and $\lambda_{\text{offset}}\,\mathcal{L}_{\text{offset}}$, while $\lambda_{\text{CCL}}\mathcal{L}_{\text{CCL}}$ should be smaller.

4. Experiment

We evaluate our ABW-Font for the Chinese few-shot font generation task. The inputs and outputs of network are both 80×80 images. We have implemented our method on an NVIDIA GeForce RTX 3080Ti GPU with 16G of video memory. We first train a network without our ABW module for 180k iterations and it requires about 48 hours. Then we further train the network with ABW module and the compact clustering loss term for 40k iterations, which requires around 12 hours. Our code is based on DG-Font [39] and CF-Font [35]. In the following, we introduce the dataset, baselines, evaluation metrics, and various experimental results to verify the effectiveness of our method.

4.1. Experimental Settings

Dataset. Due to copyright issues with fonts, the datasets used in many previous font generation methods were non-public datasets. In this paper, We have collected 500 types of fonts, which is currently the dataset with the largest number of fonts as we know. It includes commonly used standard fonts such as Kai, Song, and Hei, as well as numerous variants of them. There are also many fonts with article styles and creativeness in our dataset, such as clerical script fonts, cursive script fonts, grass script fonts, cartoon fonts, and so on. The diversity of our collected fonts allows our model to learn and generate a wide range of calligraphic styles and enhances the model's ability to capture the nuances of Chinese handwriting. The fonts are sourced from various repositories, including commercial and free fonts. We ensure that all fonts are used in compliance with their respective license agreements. As the original font files are not distributed with this work. Users can obtain the fonts from the provided sources. A list of the fonts and their sources is provided in the supplementary material for reference. For further details on the specific licenses, please refer to the original font providers.

Our character set is based on GB/T 2312 [25] which includes commonly used Chinese characters. We followed the approach of CF-Font in constructing the dataset, allowing for the most fair comparison with it. We removed some characters not supported by comparison methods and ultimately left 6446 characters. Then we divided all the characters into two parts: the training characters include 800 randomly picked characters, and the testing characters include the remaining 5646 characters. We divided all 500 fonts into 400 seen fonts and 100 unseen fonts. Finally, our dataset is as follows: The training set contains 400 seen fonts, and each font uses the 800 seen characters. The test set consists of 2 parts: One part is 400 seen fonts with the 5646 unseen characters, Another part is 100 unseen fonts with 5646 unseen characters. As our model is a few-shot font generation method, The reference images of a font are 16 randomly selected characters from the training characters. We followed the approach of CF-Font in constructing the dataset, allowing for the most fair comparison with it.

Baselines. We compare our model with 5 recent few-shot font generation methods, including LF-Font [27], MX-Font [28], VQ-Font [41]), DG-Font [39], and CF-Font [35]).

LF-Font simplifies component-wise styles by a product of component factor and style factor, without utilizing strong locality supervision, e.g., the location of each component, skeleton, or stroke. Many works are innovative based on it, such as FS-Font[33] and VQ-Font[41]. MX-Font extracts multiple style features not explicitly conditioned on component-level labels, but automatically by multiple experts to represent different local concepts. LF-Font and MX-Font are both representative weak supervision methods using component-level information.

VQ-Font proposes a VQGAN-based[10] framework to enhance glyph fidelity through token prior refinement and structure-aware enhancement. It leverages the inherent design of Chinese characters, combining structural components like radicals in specific arrangements to recalibrate fine-grained styles, improving style matching and fusion at the structural level. It is currently one of the most effective methods of using component-level supervision.

The remaining two methods, DG-Font and CF-Font only use universal information for few-shot font generation, without introducing any component-level information. DG-Font designs a network structure mainly composed of deformable convolutions, which more effectively utilize universal information. CF-Font [35] designs a content fusion module to represent the source font as a weighted sum of basis fonts. It makes the style in content features similar to the target, greatly reducing the dependence on selecting the source font. It introduces the basis fonts and weight, which is closely related to our work.

Metrics. We employ the following metrics to evaluate the quality of generation both at the pixel level and perception. Specifically, pixel-level metrics are L1, root mean square error (RMSE), and structural similarity index measure (SSIM) [37]. They focus on per-pixel consistency between the generated images and ground truth. Perceptual metrics include Fréchet Inception Distance (FID) [13] and Learned Perceptual Image Patch Similarity (LPIPS) [43], both of which measure the similarity of features and are closer to human vision.

Training details. For our dataset, font number N is 400, the initial basis number M is 10. The input character image

Method	Seen Fonts					Unseen Fonts				
	L1↓	RMSE↓	SSIM↑	LPIPS↓	FID↓	L1↓	RMSE↓	SSIM↑	LPIPS↓	FID↓
LF-Font [27]	0.08312	0.2505	0.6682	0.1220	48.08	0.08595	0.2565	0.6536	0.1231	44.21
MX-Font [28]	0.07348	0.2317	0.7007	0.1042	33.47	0.08288	0.2504	0.6743	0.1180	40.57
DG-Font [39]	0.06362	0.2103	0.7392	0.0867	31.46	0.06950	0.2218	0.7149	0.0967	42.24
VQ-Font[41]	0.06232	0.2098	0.7495	0.1030	18.41	0.07061	0.2268	0.7183	0.1142	24.90
CF-Font [35]	0.05676	0.1938	0.7711	0.0777	23.00	0.06298	0.2072	0.7485	**0.0858**	23.94
ABW-Font	**0.05545**	**0.1904**	**0.7770**	**0.0758**	**17.61**	**0.06068**	**0.2023**	**0.7554**	0.0870	**20.87**

Table 1. Quantitative evaluation on the whole dataset. We evaluate the methods on seen and unseen fonts. The bold numbers indicate the best performance in each column.

Figure 3. Qualitative comparison with competing methods on unseen characters of seen fonts and unseen fonts. The samples in the red box represent structural errors.

size is 50×50 in the middle of an 80×80 blank background, so the input image size is 80×80. Throughout the entire training, We use Adam[20] with β_1 being set to 0.9 and β_2 being set to 0.99 for the style encoder, and RMSprop with α being set to 0.99 for the content encoder. The weight decay is set to 1e-4.

When training a base model without ABW module, the batch size is 16, the learning rate is 1e-4, the iterations is 180k, the coefficients λ_{img}, λ_{offset}, λ_{pcl}, for the respective loss terms are set to 0.1, 0.5, 0.05.

After adding ABW module and the compact clustering loss term, the batch size is 16, the learning rate is 1e-4, the iterations is 40k, the coefficients λ_{img}, λ_{offset} , λ_{pcl}, λ_{CCL} for the respective loss terms are set to 0.1, 0.5, 0.05, 1e-5. In phase 1 of the training, the learning rate of EM η is set to 0.05, and the learning rate of RPCL ζ is set to 1e-4.

4.2. Quantitative and qualitative evaluation

We compare our method with the baselines. To be fair, we use font Kai as the source font in LF-Font, MX-Font, and VQ-Font, and use font Song in DG-Font which followed the approach in their article. We retrain all these methods using their default settings on our dataset. When training VQ-Font, as some characters in our dataset are not being labeled, we generate labels according to their method.

Quantitative comparison. As Table. 1 illustrates, our ABW-Font outperforms these recent methods. For L1 loss, RMSE, and SSIM, the metrics focus on the pixel-level differences between the generated image and the ground truth, our method achieved the best results in all of them both in

seen fonts and unseen fonts. This represents that the images generated by our method are closer to the real images both in terms of style features and spatial positions. For the metrics closer to human perceptions, LPIPS and FID, our method also achieves great improvements in FID of seen fonts and unseen fonts. Our only metric worse than CF-Font is the LPIPS on unseen fonts. We loses to it by a disadvantage of -1.4% but still outperforms the third place by 10.0%.

Qualitative comparison. In Fig. 3, we show the generated results using different few-shot font generation methods both on seen fonts and unseen fonts. Overall, all methods generate better results on seen fonts than on unseen fonts. It can be seen that the results generated by LF-Font and MX-Font often have global structural errors despite using weak supervision about components. Their generated characters often miss some parts. For some complex characters, the entire character may be blurred and it is difficult to recognize the content information. VQ-Font classifies characters based on prior knowledge and enhances the use of component information using a VQGAN-based framework. DG-Font does not add supervision about components but uses a network that introduces deformable convolutions to complete the generation task. However, the results generated by these two methods exhibit stylistic differences from the ground truth and also contain some local structural errors. The content input of the above methods requires a manually selected source font, which will have an impact on their results to some extent. CF-Font has introduced a content fusion module that uses multiple basis fonts as content input rather than using a single source font. But there are also some structural errors in its samples, and some strokes are incoherent. Our ABW-Font learns more appropriate basis fonts and content fusion weights, and they will be continuously updated during training. Characters generated by ours are of high quality in terms of style consistency and structural correctness.

4.3. Comprision of Basis fonts and weights

We compare the basis fonts and content fusion weights learned/calculated by ABW-Font and CF-Font in Fig. 4 and Fig 5. Fig. 4 (a) shows the basis fonts obtained by two methods. We take the Chinese character "Ji" as an example to illustrate the styles of the basis fonts obtained by two methods. Fig. 4 (b) and Fig. 5 show the details of the basis fonts and content fusion weights obtained by two methods. A character displayed at the center of a pie chart is generated by combining features from a group of basis fonts with different weights. Each colored region in a pie chart represents a single basis font, and its size reveals the importance (i.e., the content fusion weight) for composing the target font. Besides, basis fonts with content fusion weights less than 10% are omitted.

CF-Font calculates the L1 distance between the few-shot content feature maps each other of all fonts in the training set and then uses k-means clustering on the obtained vectors to get their basis fonts. For font n, given the concatenated feature map F_n of its few-shot reference characters, the basis fonts of CF-Font can be calculated by Eq. (17).

$$\begin{aligned} \boldsymbol{d}_n &= (d_{n1}, d_{n2}, \ldots, d_{nN}), \quad d_{nj} = \|\boldsymbol{F}_n - \boldsymbol{F}_j\|_1, \\ \boldsymbol{e}_n &= \sigma(\boldsymbol{d}_n), \\ \{B_i\}_{i=1}^{M} &= k\text{-means}(\{\boldsymbol{e}_n\}_{n=1}^{N}), \end{aligned} \tag{17}$$

where σ is the softmax operation, d_{nj} is the L1 distance between two fonts, and set $\{B_i\}_{i=1}^{M}$ contains the indices of the selected basis fonts. The basis fonts calculated by CF-Font will no longer change, while our basis fonts will be updated by ABW module during training. Images of our basis fonts shown in Fig. (4) are obtained at the end of training. It can be seen that the basis fonts determined by CF-Font have a stronger style. However, fonts with a stronger style are not suitable for combination because if the difference among the basis fonts is too large, their combination will lead to an unreasonable result, such as Fig. 4 (b-2) and (b-10), and Fig. 5 (1) and (6). Our basis fonts are iterated via ABW module. We use an RPCL enhanced EM algorithm to learn basis fonts with more generalized styles. In Fig. 4 (b-1), (b-2), (b-7), (b-8), (b-9), and (b-10); and in Fig. 5 (5), (6), (7), and (8), there is a significant difference in style between the results obtained by CF-Font and the target, indicating that the style represented by their basis fonts is less generalized. The difference between the fonts generated by ABW-Font and the target is smaller, reflecting the superiority of our method in obtaining the basis fonts.

For the target font n and its content feature F_n, CF-Font measures its similarity to the basis fonts $\{B_i\}_{i=1}^{M}$. Then the content fusion weight w_n of CF-Font is calculated as follows:

$$\begin{aligned} \boldsymbol{d}'_n &= (d_{n1}, d_{n2}, \ldots, d_{nM}), \quad d_{im} = \|\boldsymbol{F}_n - \boldsymbol{F}_m\|_1, \\ \boldsymbol{w}_n &= \sigma(-\boldsymbol{d}'_n / t), \end{aligned} \tag{18}$$

where t is the temperature of the softmax operation, $\boldsymbol{F}_m$ is the concatenated feature map of the few-shot reference characters for font m, which is one of the basis fonts. It can be seen that the content fusion weights determined by CF-Font are sparser than ours, and their basis fonts are not as representative as ours. If there is a significant gap between the basis font with the highest proportion and the target, the generated results will be greatly influenced by the basis font that has the highest weights, resulting in a decrease in the similarity between the generated results and the target. As shown in Fig. 4 (b-1), (b-2), (b-5), (b-6), (b-7), and (b-8); and in Fig. 5 (3), (4), (7), and (8), the results generated by CF-Font are more like the basis font with the highest proportion rather than the target font. In this case, their method

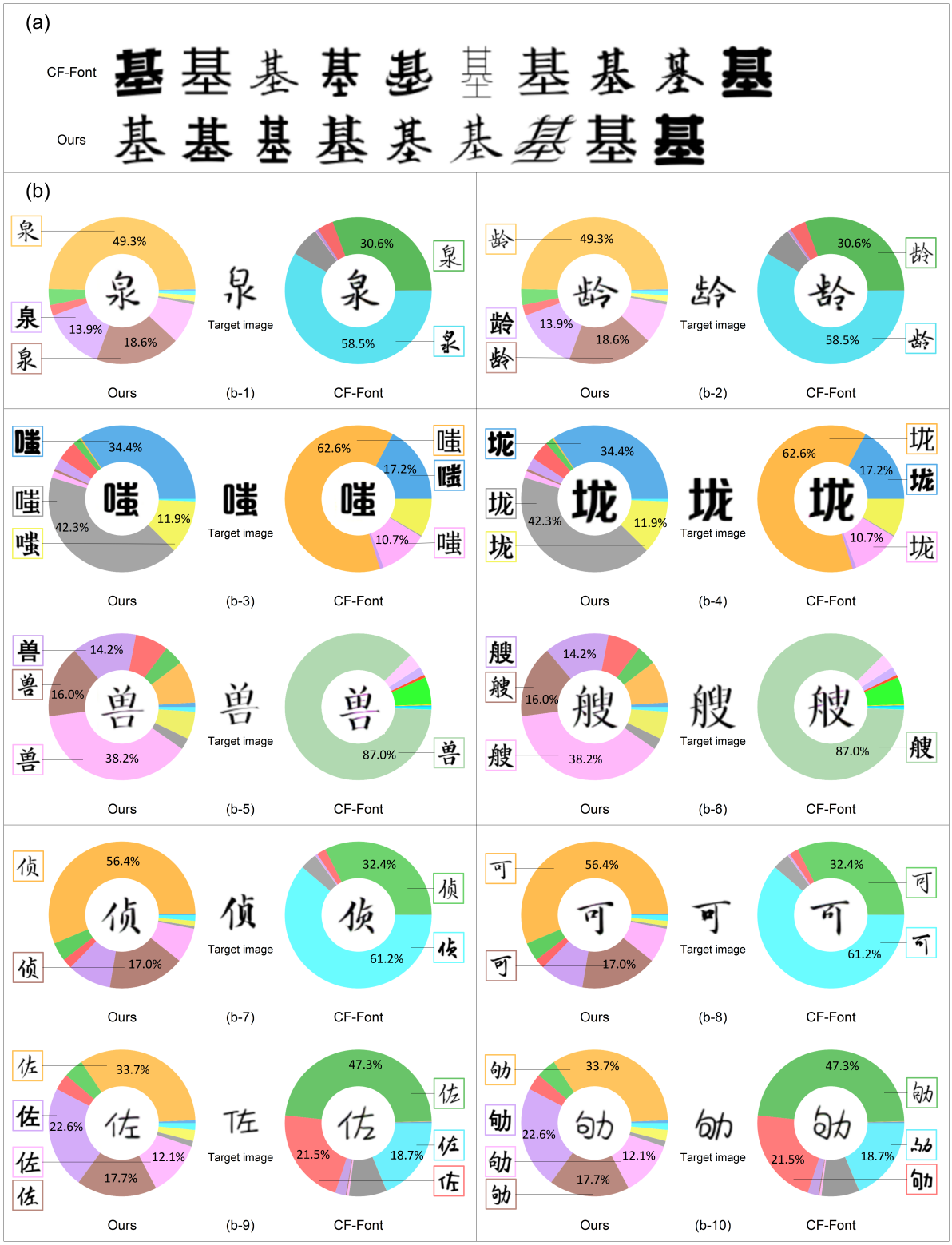

Figure 4. Comparisons of basis fonts selection and content fusion weights for ABW-Font and CF-Font on font generation. (a) The basis fonts selected/learned via CF-Font and ABW-Font. We take the Chinese character "Ji" as an example to demonstrate the styles of the basis fonts obtained by two methods. (b) Comparisons of the content fusion details for ABW-Font and CF-Font on unseen fonts. A character displayed at the center of a pie chart is generated by combining features from a group of basis fonts with different weights. Each colored region in a pie chart represents a single basis font, and its size reveals the importance (i.e., the content fusion weight) for composing the target font. Besides, basis fonts with content fusion weights less than 10% are omitted.

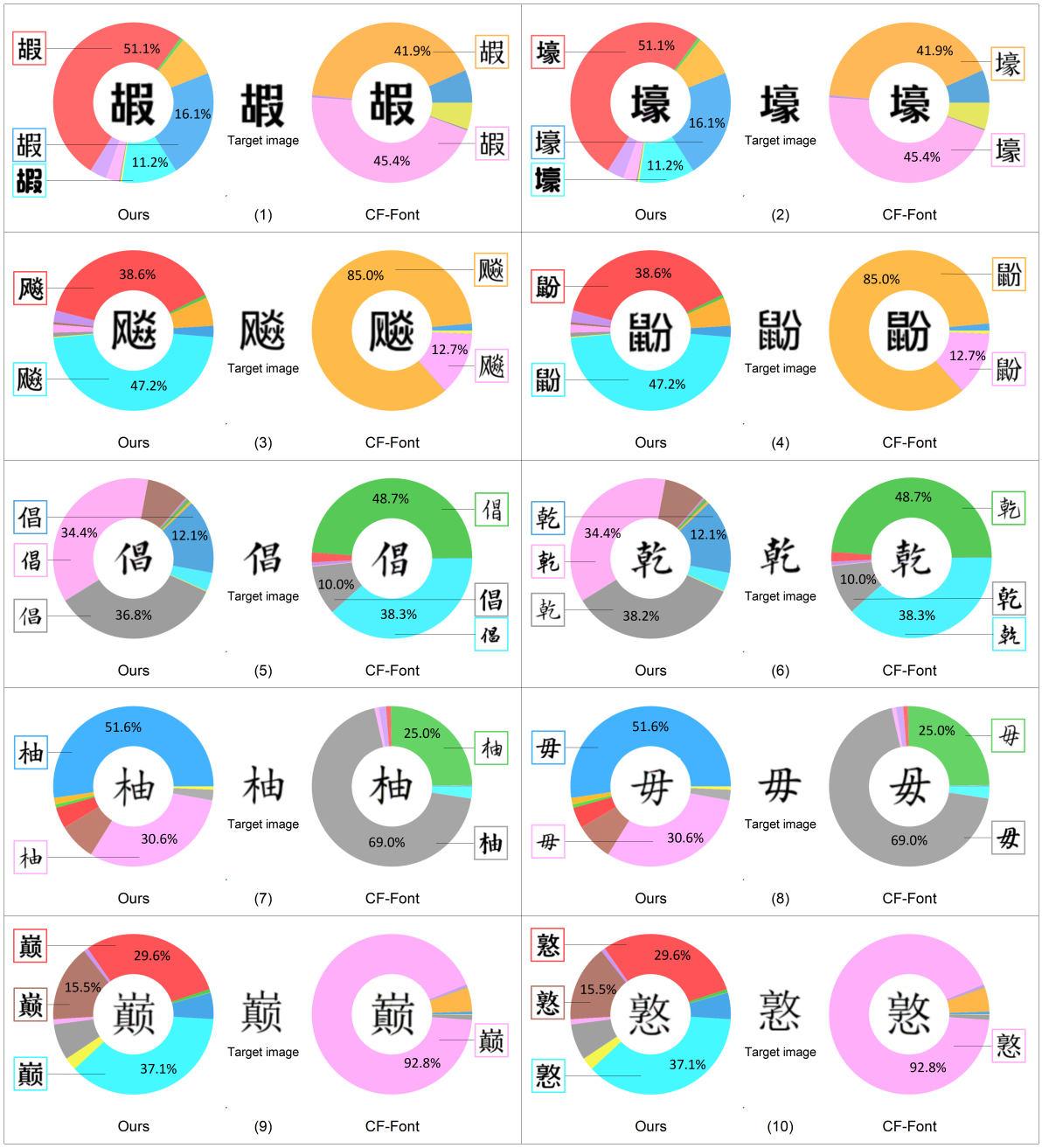

Figure 5. Comparisons of basis fonts selection and content fusion weights for ABW-Font and CF-Font on seen fonts. A character displayed at the center of a pie chart is generated by combining features from a group of basis fonts with different weights. Each colored region in a pie chart represents a single basis font, and its size reveals the importance (i.e., the content fusion weight) for composing the target font. Besides, basis fonts with content fusion weights less than 10% are omitted.

can be seen as degenerating into using a single source font. Since using L1 distance will result in such sparser weights, we choose to use L2 distance when calculating the basis fonts and content fusion weights. Also, the basis fonts and content fusion weights in CF-Font are fixed once they are calculated, while ours can adapt to the network updating via the ABW module. In Fig. 4 (b-3) and (b-4), and in Fig. 5 (9) and (10), the results generated by two methods are both close to the target in style, but our method performs better in structural details.

Method		Seen Fonts					Unseen Fonts				
B	W	L1↓	RMSE↓	SSIM↑	LPIPS↓	FID↓	L1↓	RMSE↓	SSIM↑	LPIPS↓	FID↓
		0.06732	0.2180	0.7249	0.0916	26.48	0.07272	0.2283	0.7052	0.1018	31.39
✓		0.05795	0.1964	0.7666	0.0782	19.45	0.06186	0.2048	0.7507	0.0896	24.14
	✓	**0.05543**	**0.1907**	**0.7766**	**0.0752**	19.56	0.06134	0.2043	0.7526	**0.0868**	22.51
✓	✓	0.05672	0.1935	0.7706	0.0780	**17.98**	**0.06077**	**0.2022**	**0.7542**	0.0896	**21.99**

Table 2. Effectiveness of different components in ABW module. The first row is the result of the base network. W and B represent learning the content fusion weights with ABW module and learning the basis fonts with ABW module.

Method	Seen Fonts					Unseen Fonts				
	L1↓	RMSE↓	SSIM↑	LPIPS↓	FID↓	L1↓	RMSE↓	SSIM↑	LPIPS↓	FID↓
LF-Font [27]	0.08312	0.2505	0.6682	0.1220	48.08	0.08595	0.2565	0.6536	0.1231	44.21
LF-Font*	0.08264	0.2461	0.6732	0.1210	43.75	0.08482	0.2507	0.6607	0.1223	39.48
MX-Font [28]	0.07348	0.2317	0.7007	0.1042	33.47	0.08288	0.2504	0.6743	0.1180	40.57
MX-Font*	0.07287	0.2275	0.7063	0.1028	32.56	0.08202	0.2464	0.6813	0.1139	36.81
DG-Font [39]	0.06362	0.2103	0.7392	0.0867	31.46	0.06950	0.2218	0.7149	0.0967	42.24
DG-Font*	0.06282	0.2083	0.7435	0.0869	30.20	0.06897	0.2206	0.7219	0.0957	31.00

Table 3. Quantitative evaluation of using different source fonts in LF-Font, MX-Font, and DG-Font. * represents our method, which uses a single basis font we have learned as the source font on the whole dataset. We evaluate the methods on seen and unseen fonts.

4.4. Ablation Study

In this section, we discuss the effectiveness of the attendance of different parts in our ABW module. We first train a model without adding the ABW module as the baseline. Then, we gradually add our ABW module to the baseline for validation. It is worth noting that when verifying the effect of our ABW module, we separately control the ABW module to learn only the content fusion weights, or learn only the basis fonts, or learn both the basis fonts and content fusion weights simultaneously to verify which part takes a role or both. We validate the metrics achieved by the above model on seen and unseen fonts, and the results are shown in Table 2.

Learning basis fonts or content fusion weights separately has improved the relevant performance compared to the base model. Among them, the improvement brought by learning the content fusion weights separately is greater. On seen fonts, the results of learning both basis fonts and content fusion weights simultaneously are inferior to the model that only learns weights in many metrics but still achieves better in FID. On unseen fonts, except for a small decrease in LPIPS, better results were achieved in other indicators. We need to emphasize that all the styles of seen fonts have appeared during training. As the early font style transfer task required the inclusion of all font styles in the training set, the test on seen fonts was necessary at that time. However, for the few-shot font generation task, the original intention was to generate a whole new set of unseen fonts based on several character images provided to us. Therefore, the experimental results on the unseen fonts are more valuable for reference.

The inferior performance on unseen fonts represents that it is lack of robustness. The method of learning both the basis fonts and content fusion weights simultaneously performs better on unseen fonts, so we believe it has a stronger generalization ability to adapt to the few-shot font generation task. The data in the last row of Table 2 is different from our data in Table 1 because lack of our compact clustering loss term.

4.5. Determining Robust Source Font by ABW Module

As emphasized in this paper, the selection of the source font can greatly affect the results in the font generation task. In numerous previous works, font Kai and Song were usually used as the source font, because they are the most commonly used fonts in Chinese text editing. However, this approach may not be the best. If the source font differs significantly from most of the fonts in the dataset, the generated results will deteriorate.

In this paper, our ABW module can learn a better set of basis fonts with their corresponding weights to represent the source font, thereby avoiding the impact of selecting it. For many tasks that still require a source font as the content input, we can propose a simple method to determine a more robust source font. We have verified on our dataset that this method can achieve better results in some previous works.

Specifically, following the approach in Subsection 3.3, we first train a base model without ABW module. Then we pick up 16 few-shot reference characters from all train fonts and add them to the encoder. Then we set the num-

ber of basis fonts to 1 and perform only 1 iteration, using the calculated basis font as the source font. For our dataset, the source font determined by this method is a Kai-like font that is coarser and more angular. We test it in LF-Font, MX-Font, and DG-Font, and the results using our method are better than those obtained in their articles using font Song or Kai as the source font, respectively. The experimental results are shown in the Table 3. We hope that this simple method of determining the source font can provide a reference for researchers who need to use the source font to participate in the font generation task.

5. Conclusion

In this paper, we propose an ABW module to make the basis fonts and content fusion weights learnable, improving the quality of results in few-shot font generation. We also add a compact clustering loss term making the training more stable and accelerating the convergence of the network. The experiment proves that our method outperforms recent advanced few-shot font generation methods.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (62406064) and the Fundamental Research Funds for the Central Universities (2232024D-28).

References

[1] GitHub - kaonashi-tyc/Rewrite: Neural Style Transfer For Chinese Characters — github.com. https://github.com/kaonashi-tyc/Rewrite. [Accessed 18-05-2024]. 1, 2

[2] GitHub - kaonashi-tyc/zi2zi: Learning Chinese Character style with conditional GAN — github.com. https://github.com/kaonashi-tyc/zi2zi. [Accessed 18-05-2024]. 1, 2

[3] K. Baek, Y. Choi, Y. Uh, J. Yoo, and H. Shim. Rethinking the truly unsupervised image-to-image translation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 14154–14163, 2021. 2

[4] S. Benaim and L. Wolf. One-sided unsupervised domain mapping. *Advances in neural information processing systems*, 30, 2017. 2

[5] J. Cha, S. Chun, G. Lee, B. Lee, S. Kim, and H. Lee. Few-shot compositional font generation with dual memory. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16*, pages 735–751. Springer, 2020. 1, 2

[6] X. Chen, C. Xu, X. Yang, L. Song, and D. Tao. Gated-gan: Adversarial gated networks for multi-collection style transfer. *IEEE Transactions on Image Processing*, 28(2):546–560, 2018. 2

[7] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797, 2018. 2

[8] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8188–8197, 2020. 2

[9] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017. 2

[10] P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021. 3, 7

[11] Y. Gao and J. Wu. Gan-based unpaired chinese character image translation via skeleton transformation and stroke rendering. In *proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 646–653, 2020. 2

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 1, 6

[13] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 7

[14] X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510, 2017. 2, 4

[15] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 2

[16] Y. Jiang, Z. Lian, Y. Tang, and J. Xiao. Dcfont: an end-to-end deep chinese font generation system. In *SIGGRAPH Asia 2017 Technical Briefs*, pages 1–4. 2017. 2

[17] Y. Jiang, Z. Lian, Y. Tang, and J. Xiao. Scfont: Structure-guided chinese font generation via deep stacked networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4015–4022, 2019. 2

[18] H. Kazemi, S. M. Iranmanesh, and N. Nasrabadi. Style and content disentanglement in generative adversarial networks. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 848–856. IEEE, 2019. 2

[19] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International conference on machine learning*, pages 1857–1865. PMLR, 2017. 2

[20] D. P. Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 8

[21] Y. Kong, C. Luo, W. Ma, Q. Zhu, S. Zhu, N. Yuan, and L. Jin. Look closer to supervise better: One-shot font generation via component-based discriminator. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13482–13491, 2022. 1, 3

[22] G. Kwon and J. C. Ye. Diagonal attention and style-based gan for content-style disentanglement in image generation and translation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13980–13989, 2021. 2

[23] M.-Y. Liu, X. Huang, A. Mallya, T. Karras, T. Aila, J. Lehtinen, and J. Kautz. Few-shot unsupervised image-to-image translation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10551–10560, 2019. 2

[24] W. Liu, F. Liu, F. Ding, Q. He, and Z. Yi. Xmp-font: Self-supervised cross-modality pre-training for few-shot font generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7905–7914, 2022. 1, 3

[25] K. Lunde. *CJKV information processing*. " O'Reilly Media, Inc.", 2008. 7

[26] P. Lyu, X. Bai, C. Yao, Z. Zhu, T. Huang, and W. Liu. Auto-encoder guided gan for chinese calligraphy synthesis. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 1095–1100. IEEE, 2017. 1, 2

[27] S. Park, S. Chun, J. Cha, B. Lee, and H. Shim. Few-shot font generation with localized style representations and factorization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 2393–2402, 2021. 1, 3, 7, 8, 12

[28] S. Park, S. Chun, J. Cha, B. Lee, and H. Shim. Multiple heads are better than one: Few-shot font generation with multiple localized experts. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 13900–13909, 2021. 1, 3, 7, 8, 12

[29] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015. 4

[30] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116, 2017. 2

[31] D. Sun, T. Ren, C. Li, H. Su, and J. Zhu. Learning to write stylized chinese characters by reading a handful of examples. *arXiv preprint arXiv:1712.06424*, 2017. 1, 2

[32] D. Sun, Q. Zhang, and J. Yang. Pyramid embedded generative adversarial network for automated font generation. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 976–981. IEEE, 2018. 1, 2

[33] L. Tang, Y. Cai, J. Liu, Z. Hong, M. Gong, M. Fan, J. Han, J. Liu, E. Ding, and J. Wang. Few-shot font generation by learning fine-grained local styles. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7895–7904, 2022. 1, 3, 7

[34] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017. 3

[35] C. Wang, M. Zhou, T. Ge, Y. Jiang, H. Bao, and W. Xu. Cf-font: Content fusion for few-shot font generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1858–1867, 2023. 2, 3, 6, 7, 8

[36] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018. 2

[37] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 7

[38] S.-J. Wu, C.-Y. Yang, and J. Y.-j. Hsu. Calligan: Style and structure-aware chinese calligraphy character generator. *arXiv preprint arXiv:2005.12500*, 2020. 1, 2

[39] Y. Xie, X. Chen, L. Sun, and Y. Lu. Dg-font: Deformable generative networks for unsupervised font generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5130–5140, 2021. 2, 4, 6, 7, 8, 12

[40] L. Xu, A. Krzyzak, and E. Oja. Rival penalized competitive learning for clustering analysis, rbf net, and curve detection. *IEEE Transactions on Neural networks*, 4(4):636–649, 1993. 2

[41] M. Yao, Y. Zhang, X. Lin, X. Li, and W. Zuo. Vq-font: Few-shot font generation with structure-aware enhancement and quantization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16407–16415, 2024. 3, 7, 8

[42] S. Zang, S. Tu, and L. Xu. Controllable stroke-based sketch synthesis from a self-organized latent space. *Neural Networks*, 137:138–150, 2021. 2, 4

[43] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 7

[44] Y. Zhang, Y. Zhang, and W. Cai. Separating style and content for generalized style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8447–8455, 2018. 1, 2

[45] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017. 2

[46] X. Zhu, H. Hu, S. Lin, and J. Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9308–9316, 2019. 2