Few-Shot Font Generation by Learning Fine-Grained Local Styles (Supplementary Materials)

Licheng Tang[1*] Yiyang Cai[2*] Jiaming Liu[1†] Zhibin Hong[1] Mingming Gong[3]
Minhu Fan[1] Junyu Han[1] Jingtuo Liu[1] Errui Ding[1] Jingdong Wang[1]
[1]Baidu Inc. [2]University of California, Berkeley [3]University of Melbourne
{tanglicheng,liujiaming03,hongzhibin,fanminhu,liujingtuo,dingerrui,wangjingdong}@baidu.com
frank_cai@berkeley.edu, mingming.gong@unimelb.edu.au

A. Implementation Details

A.1. Reference Selection

In this section, we will discuss the implementation details of reference selection, which can be divided into two steps: **reference set determination** and **content-reference mapping**.

Reference set determination As mentioned in paper's section 3.4, our target is to search for a reference glyph set which covers the most conspicuous-level components, and our detailed algorithm can be demonstrated by Algorithm 1. In our algorithm, X denotes the entire character list sorted by glyphs' occurrence frequency, whose total capacity is 20K. T denotes the single level decomposition look-up tables for all characters in X. For one key x_i in T, their corresponding value χ_i is a subset of X, consisting of their decomposing components' name list and corresponding decomposing form. By exploring this components list, we employ breadth first search to recursively obtain all the components at different levels, which forming the component tree in our work. C denotes the set that contains all conspicuous-level components, whose total capacity is **374**. N_{ref} denotes the total capacity of our target reference set, which is is fixed to **100** in our experiments.

As an initialization, we set our target reference set $\hat{U}$ and corresponding covering components $\hat{C}$ as $\emptyset$. Then we begin searching process in X. For every glyph x_i in X, we obtain its conspicuous-level components c_i via the function *searchComponents* in 1. If there is unique components in c_i which is not in current component pool $\hat{C}$, we regard this glyph x_i as our target reference glyph, and we add it and its unique components to $\hat{U}$ and $\hat{C}$ respectively. In our experiment, we ensure that every latest adding reference have **2** or more unique components. Once the capacity of $\hat{U}$ reaches N_{ref}, we terminate the searching process.

Content-reference mapping Since we have determined the reference set. It's more convenient for us to find the mapping relationship between a arbitrary glyph and its ref-

Algorithm 1: Reference set selection

input : $X = \{x_i\}$: Common-used glyph list.
$T = \{x_i : \chi_i\}$: Single-level decomposition look-up table. $\chi_i \subset X$.
C: Conspicuous-level component set.
N_{ref}: Max capacity of Reference set.
d: Max search depth in component tree.
output: $\hat{U}$: Reference set.
$\hat{C}$: All components which U can cover.

Function `searchComponents(`x_i`)`**:**
  $queue \leftarrow \{x_i\}$; $i \leftarrow 0$; $c_i \leftarrow \emptyset$
  repeat
    $queue^* \leftarrow \emptyset$
    for $x \in queue$ **do**
      $queue^* \leftarrow queue^* \cup T(x)$
      $c_i \leftarrow T(x) \cap C$
    end
    $queue \leftarrow queue^*$; $i \leftarrow i + 1$
  until $i \geq d$;
  return c_i
End Function

Function `Main()`**:**
  $\hat{U} \leftarrow \emptyset$; $\hat{C} \leftarrow \emptyset$
  for $x_i \in X$ **do**
    $c_i \leftarrow$ searchComponents(x_i)
    if $\exists \hat{c} \in c_i$ *s.t.* $\hat{c} \notin \hat{C}$ **then** $\hat{C} \leftarrow \hat{C} \cup c_i$, add x_i to $\hat{U}$;
    if *Len*($\hat{U}$) $\geq N_{ref}$ **then break**;
  end
  return $\hat{U}, \hat{C}$
End Function

*Equal contribution.
†Corresponding author.

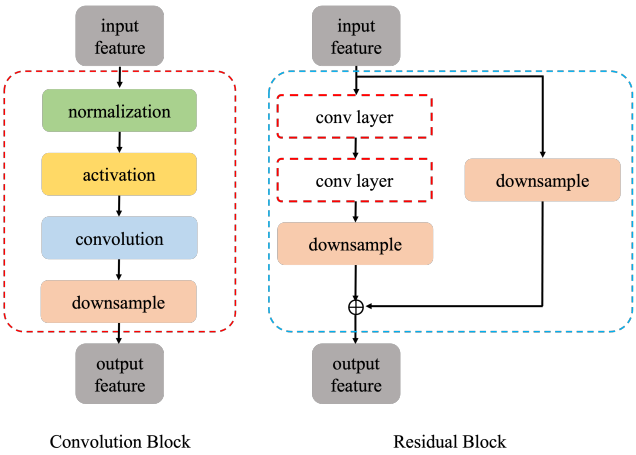


Figure 1. Structures of the Convolution Block and the Residual Block in Tab.1

erences whose conspicuous components are in common. For k-shot reference mapping (one content glyph corresponds to k style references), we do the following searching process for k times: We find the glyph in reference set which share the most conspicuous component, and remove it from the original reference set. If there is multiple choice, we leave the one whose component's composition form is the same with the content glyph.

We show some of the finding mappings between content and references in Fig. 2.

A.2. Model Architecture

Our entire model can be divided into two parts: the generator and discriminator. Both of them are built up of two typical types of blocks: the Convolution block and the Residual block, which are illustrated in Fig 1. The Residual block contains two identical Convolution blocks. Since the Residual block has its own downsampling operator, its Convolution blocks skip the downsampling step.

Generator As mentioned in our method, the generator model consists of three respective modules: reference encoder E_r, content encoder E_c, and decoder D. Both E_r and E_c are also made up of the following two types of blocks: the Convolution block and Residual block. The detailed architecture is illustrated in Table. 1. In D, every Convolution block is followed by a spectral normalization.

Discriminator The discriminator is also made up of the Convolution block and the Residual block. The detailed architecture is illustrated in Table. 1.

A.3. Training details

We use Adam optimizer to optimize the FSFont's parameters. The generator and discriminator's learning rates are 0.0002 and 0.0008, respectively. Kaiming initialization is applied for the model. During training, by default we set the reference to be 3-shot for training. If one glyph's reference set's capacity is less than 3, we duplicate one of reference glyph until the capacity equals to 3 for purpose of batch training. In our proposed SAM, empirically we set number of attention heads to be 8 and batch size to be 32. We train our model with full objective function for 500k iterations.

B. Additional Experimental Results

In Fig.2, we demonstrate more experimental results on unseen fonts. The results show that FSFont can deal with variant font styles, including typewriter fonts, artistic fonts, and handwriting fonts. In addition, we show the content font we used in our experiment in Fig.3.

We also demonstrate more visualization results about the attention maps in Fig. 4. We show the attention maps on characters with variant structures. It is shown that the attention is able to capture the correct correspondence in different structures.

C. User Study Examples

We show the sample images used for the user study in Fig.5 and Fig.6. A content image, a reference set, and shuffled results from five methods, i.e. FUNIT [2], AGIS-Net [1], LF-Font [3], DFFont [5], MX-Font [4], and FSFont, are displayed to users for every query. Users are required to choose the most consistent one according the style of the reference set. As the orders of the methods are shuffled, we also provide the answers for each case.

References

[1] Jie Chang, Yujun Gu, Ya Zhang, Yan-Feng Wang, and CM Innovation. Chinese handwriting imitation with hierarchical generative adversarial network. In *BMVC*, page 290, 2018. 2

[2] Ming-Yu Liu, Xun Huang, Arun Mallya, Tero Karras, Timo Aila, Jaakko Lehtinen, and Jan Kautz. Few-shot unsupervised image-to-image translation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10551–10560, 2019. 2

[3] Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. Few-shot font generation with localized style representations and factorization. *arXiv preprint arxiv:2009.11042*, 2020. 2

[4] Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. Multiple heads are better than one: Few-shot font generation with multiple localized experts. *arXiv preprint arXiv:2104.00887*, 2021. 2

[5] Yangchen Xie, Xinyuan Chen, Li Sun, and Yue 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

Reference Encoder E_r							
Layer Type	Normalization	Activation	Paddding	Kernel Size	Stride	Downsample	Feature maps
Convolution block	IN	ReLU	1	3	1	-	32
Convolution block	IN	ReLU	1	3	1	AvgPool	64
Convolution block	IN	ReLU	1	3	1	AvgPool	128
Residual block	IN	ReLU	1	3	1	-	128
Residual block	IN	ReLU	1	3	1	-	128
Residual block	IN	ReLU	1	3	1	AvgPool	256
Residual block	IN	ReLU	1	3	1	-	256
Output Layer	-	Sigmoid	-	-	-	-	256

Content Encoder E_c							
Layer Type	Normalization	Activation	Paddding	Kernel Size	Stride	Downsample	Feature maps
Convolution block	IN	ReLU	1	3	1	-	32
Convolution block	IN	ReLU	1	3	2	-	64
Convolution block	IN	ReLU	1	3	2	-	128
Convolution block	IN	ReLU	1	3	2	-	256
Convolution block	IN	ReLU	1	3	1	-	256

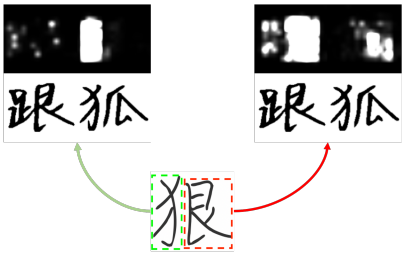
Decoder D							
Layer Type	Normalization	Activation	Paddding	Kernel Size	Stride	Upsample	Feature maps
Residual block	IN	ReLU	1	3	1	-	256
Residual block	IN	ReLU	1	3	1	-	256
Residual block	IN	ReLU	1	3	1	-	256
Convolution block	IN	ReLU	1	3	1	Nearest	128
Convolution block	IN	ReLU	1	3	1	Nearest	64
Convolution block	IN	ReLU	1	3	1	Nearest	32
Convolution block	IN	ReLU	1	3	1	-	1
Output Layer	-	Sigmoid	-	-	-	-	1

Discriminator D							
Layer Type	Normalization	Activation	Paddding	Kernel Size	Stride	Downsample	Feature maps
Convolution block	IN	-	1	3	2	-	32
Residual block	IN	ReLU	1	3	1	AvgPool	64
Residual block	IN	ReLU	1	3	1	AvgPool	128
Residual block	IN	ReLU	1	3	1	AvgPool	256
Residual block	IN	ReLU	1	3	1	-	256
Residual block	IN	ReLU	1	3	1	AdaAvgPool	512
Output layer	Multi-task embedding						

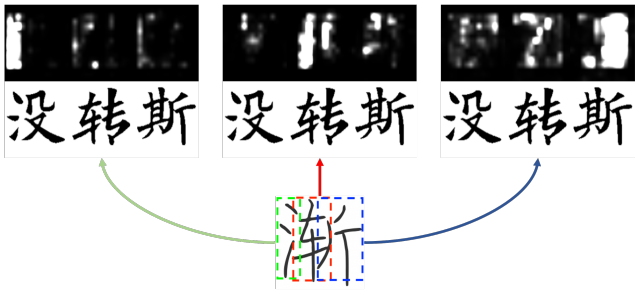
Table 1. **Architecture of the generator modules E_r, E_c, D and the discriminator.** Convolution block and Residual block denote the module mentioned in Fig 1. IN denotes instance normalization. All padding operation is zero-padding. AvgPool and AdaptiveAvgPool denotes average pooling and adaptive average pooling. In discriminator's output layer, we use two embedding operators to embed the output feature map into two prediction vector of the font style and the character's name.

Figure 2. **More Results**. This Figure shows the Content-Reference mapping and generated results on all test fonts.

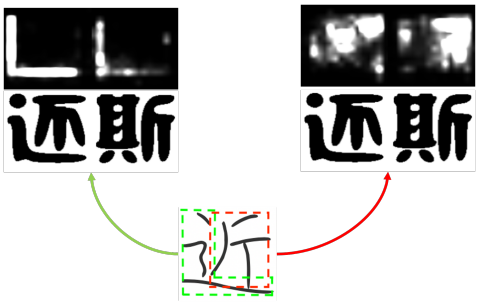
Figure 3. **The content font we used in our experiment.**



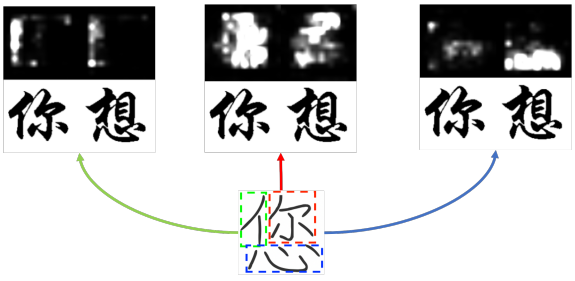
(a) Left-Right structure



(b) Left-Middle-Right structure



(c) Half-Surrounded Structure



(d) Compound Structure (Left-Right + Top-down)

Figure 4. **Visualization on Attention maps**. This Figure shows the attention maps given the different regions in content glyph.

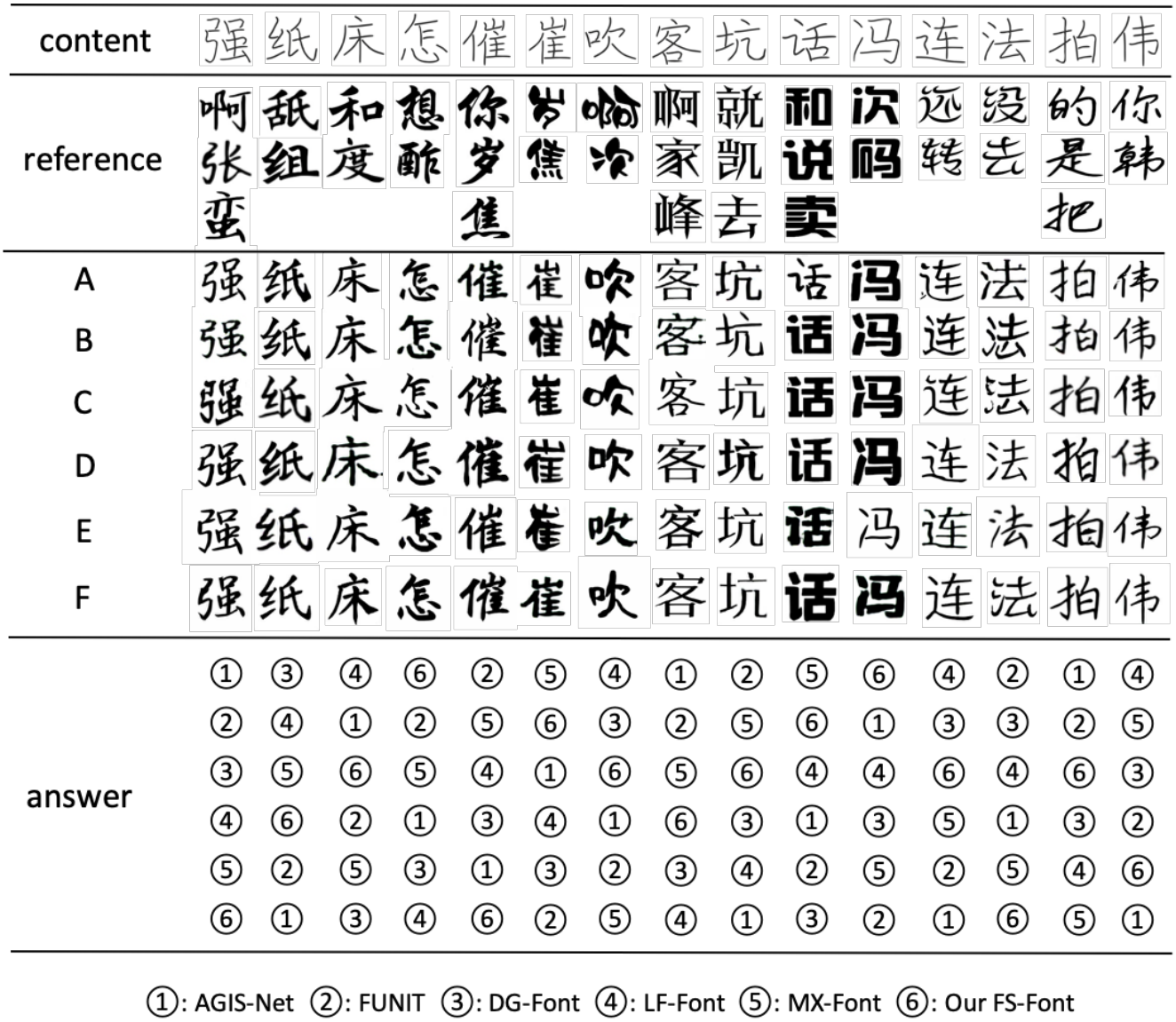


Figure 5. **User studies on UFSC (unseen font seen character) data**. This Figure shows how do we conduct the user study.

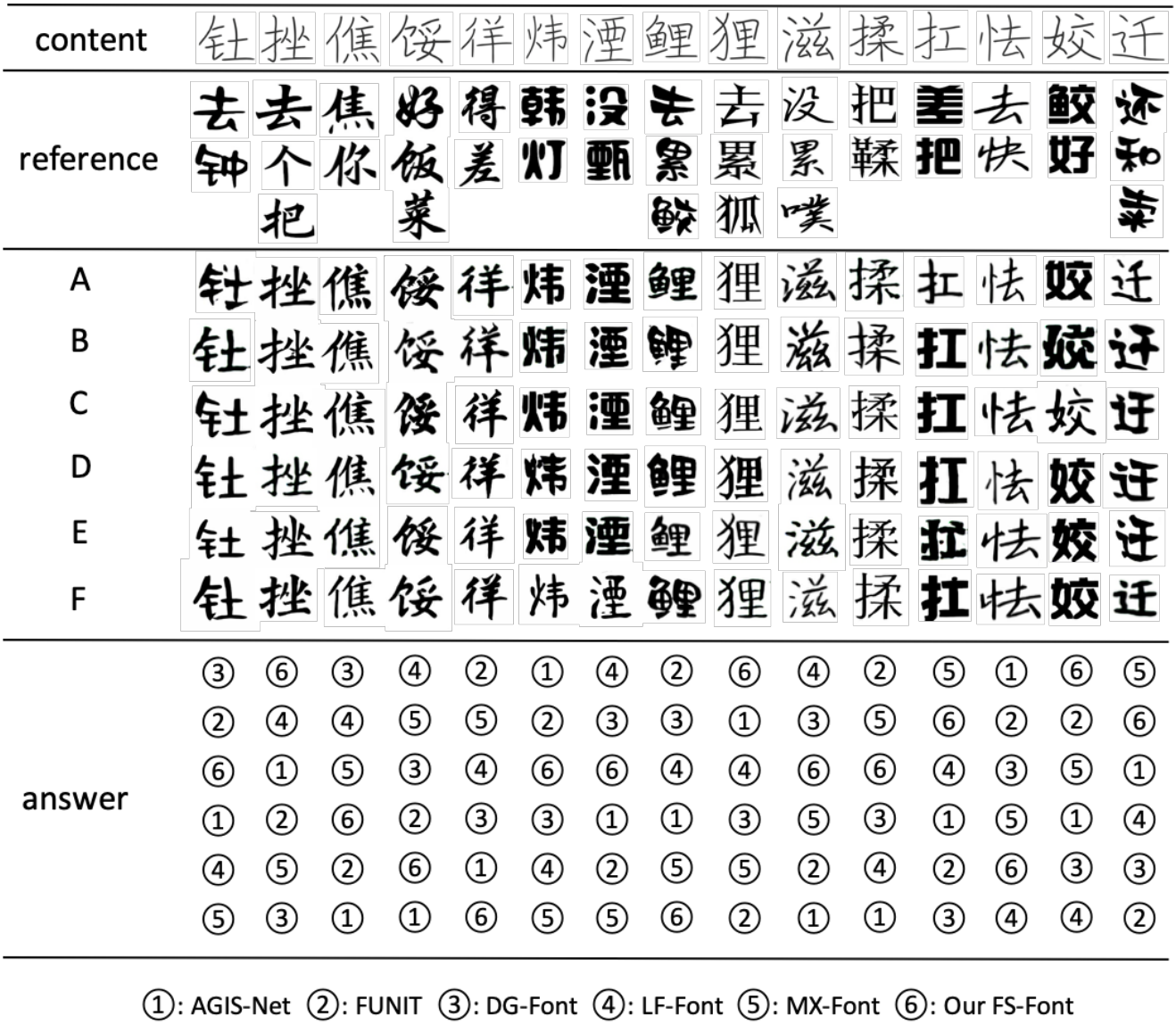


Figure 6. **User studies on UFUC (unseen font unseen character) data**. This Figure shows how do we conduct the user study.