Font Representation Learning via Paired-glyph Matching

Junho Cho\textsuperscript{1,2}  
junhocho@snu.ac.kr  
Kyuewang Lee\textsuperscript{1}  
kyuewang@snu.ac.kr  
Jin Young Choi\textsuperscript{1}  
jychoi@snu.ac.kr  
\textsuperscript{1} Department of ECE, ASRI, Seoul National University, Seoul, Korea  
\textsuperscript{2} Samsung Advanced Institute of Technology, Samsung Electronics, Suwon, Korea

Abstract

Fonts can convey profound meanings of words in various forms of glyphs. Without typography knowledge, manually selecting an appropriate font or designing a new font is a tedious and painful task. To allow users to explore vast font styles and create new font styles, font retrieval and font style transfer methods have been proposed. These tasks increase the need for learning high-quality font representations. Therefore, we propose a novel font representation learning scheme to embed font styles into the latent space. For the discriminative representation of a font from others, we propose a paired-glyph matching-based font representation learning model that attracts the representations of glyphs in the same font to one another, but pushes away those of other fonts. Through evaluations on font retrieval with query glyphs on new fonts, we show our font representation learning scheme achieves better generalization performance than the existing font representation learning techniques. Finally on the downstream font style transfer and generation tasks, we confirm the benefits of transfer learning with the proposed method.

1 Introduction

A font, which is a graphical representation of text, delivers certain visual feelings in multimedia through its matching style set of glyphs. Professional designers carefully choose fonts to convey their design intent. However, it is challenging to search for a specific font in the vast number of fonts available. Moreover, designing fonts requires typography knowledge, and aspiring designers can take months to learn typography. To cope with these difficulties, fonts should be easier to search for and create. There has been active research on font retrieval \cite{4, 15, 16, 19, 24}, font style transfer and generation \cite{1, 11, 40, 42}.

Font retrieval is a task that allows users to find similar looking fonts. Users can browse the fonts in the latent space to find the font they want. Through recognizing font style and generating new glyphs with the corresponding style, font style transfer and generation can ease the labor-intensive job of creating numerous glyphs with a certain font style. Font retrieval, style transfer and generation have historically focused on their own specific goals. However, if a powerful font representation learning method is devised, these tasks are considered downstream tasks, and performance gains can be expected through transfer learning \cite{21}. There-
We present a novel font representation learning scheme for the broader generalization on font-related downstream tasks. However, learning fonts is not as easy as one might think. Five fonts shown in Figure 1 (a), ShareTech, UbuntuCondensed, Strait, Telex, Signika are very difficult to distinguish with our eyes. Unlike general objects with textures [8], fonts have typographic elements (e.g., cap, x-height, serif, stem, stroke, descender, ascender, aperture) which are shape-based representations. Therefore, distinguishing these nuances is important for learning high-quality font representations.

In this paper, to mitigate the aforementioned difficulties, we approach how to learn these nuances through pairwise glyph similarity learning. More specifically, we try to learn the style representation of a font regardless of the shape of the character. That is, each font style keeps its unique nuance though the glyphs in the font have diverse shapes, which is referred to as Glyph-font-consistency. Paying attention to this unique nuance, we propose a new representation learning scheme to learn font features, keeping Glyph-font-consistency through a paired-glyph matching strategy. The proposed scheme attracts the font representations of glyphs in the same font to one another, but pushes away those of other fonts. We study generalization ability of our discriminative font representation learning scheme compared to existing font representation learning techniques. Finally, we evaluate performance improvement by transfer learning of our font representation learning scheme in the downstream font style transfer and generation tasks.

2 Related Works

2.1 Font Classification & Retrieval

Font classification and recognition models [3, 31, 32, 44, 45] are used to increase performance in text detection and recognition [2, 27], to make difficult calligraphy easier for users to recognize [25]. These methods of font classification only work with fixed sets of fonts, so they lack generalization to countless number of unseen fonts. Therefore, various retrieval-based methods [4, 15, 16, 19, 24] have been proposed for learning font representation and various related applications. Before the deep learning-based method appeared, Kataria et al. [16] extracted the SIFT (Scale-Invariant Feature Transform) [20] feature from each glyph of the font and defined the concatenation of glyphs as the font embedding. O’Donovan et al. [24] defined the attributes (e.g., artistic, attractive, pretentious) of fonts and used crowd-
sourced way to annotate the font attributes. And by learning a model to predict the attributes of fonts, O’Donovan et al. predicted attributes even for unseen fonts. However, specifying font attributes and determining their values is a rather subjective task, and the cost of annotations is very high, which limited annotations for small number of fonts. In light of this, tag-based font retrieval websites with relatively low annotation costs (e.g., dafont.com, myfonts.com, 10001fonts.com) appeared.

These websites provide a tag-based font search service that allows users to select and download selected fonts. Figure 1 (b) shows how users can search for fonts based on a query (e.g., cute, techno, Old English). However, the tag-based font search has the disadvantage that, much like the problem with tag-based image searches, the tag does not sufficiently describe the font, and even appropriate tags may be subjective. With the advent of deep learning, some tag-based font retrieval studies [4, 15, 19] have tried to associate font tags to learn font representation in a data-driven manner. These studies proposed a method to perform tag classification [4] on fonts or to share the font latent space with the tag representation through Word2vec [15, 19, 22]. They investigated how the specific glyph shape of a font was related to a specific emotional font tag. However, these methods cannot learn font embedding without font tags.

2.2 Font Style Transfer & Font Generation

The necessity of font style transfer methods comes from the tedious and labor-intensive job of creating numerous glyphs with font style. For example, Chinese contains more than 60,000 characters and Korean contains 11,172 characters. Early font style transfer methods [10, 33, 39, 40] were based on image-to-image translation models [13, 23, 29] with the advance of generative adversarial networks [9]. These methods transferred the font style of one glyph image to another glyph image. These methods typically extracted font style features from glyph images for reference via a font style encoder model. Each method focused on the structural design of the font style encoder, because the style encoder needed to learn a good font representation so the font style was represented well in the output image. That is, better font representation learning was helpful for better quality font generation.

3 Methodology

3.1 Notations and Our Research Objective

To establish appropriate context, it is important to outline how we denote characters, glyphs and fonts. A character set is defined by a class of characters, for instance, $C_{0-9} = \{0, 1, 2, ..., 9\}$, $C_{a-Z} = \{a, b, c, ..., X, Y, Z\}$ and $C_{0-Z} = \{0, 1, 2, ..., 9, a, b, c, ..., X, Y, Z\}$. A glyph is an image form of a character that has a specific style in a font. For example, if a glyph describes the character “Z” with a certain font $f_1$, we denote the glyph as $g_{Zf_1}$. Figure 1 (c) shows that a font $f_1$ includes a matched set of glyphs for a character set $C$. For example, the glyph set with font $f_1$ of character set $C_{0-Z}$ is denoted by

$$G_{f_1}^{C_{0-Z}} = \{g_{cf_1} \mid c \in C_{0-Z}\} = \{g_{0f_1}, g_{1f_1}, g_{2f_1}, ..., g_{Xf_1}, g_{Yf_1}, g_{Zf_1}\} \subset G_{f_1}. \tag{1}$$

Denoting the set of all fonts in the world by $\mathcal{F}$, two different fonts $f_i, f_j \in \mathcal{F}$ convey different styles through two glyph sets $(G_{f_i} = \{g_{cf_i} \mid c \in C\}$ and $G_{f_j} = \{g_{cf_j} \mid c \in C\}$).
Based on the intrinsic relationship between fonts and glyphs, our research objective is to embed the fonts to representation space so that the glyphs in the same font are embedded into a small representation area far from those of the other fonts. To this end, we propose a \textit{Paired-glyph Matching} learning scheme to pull the font representations of all glyphs in $\mathbb{G}_{f_i}$ closer to one another but push away from the font representations of the glyphs in the other glyph sets $\mathbb{G}_{f_j\neq i}$ and vice versa, as shown in Figure 2.

### 3.2 Paired-glyph Matching Learning

In \textit{Paired-glyph Matching} learning, we randomly sample two fonts, $f_1$ and $f_2$, and two characters, $c_1$ and $c_2$. Then, we get a set of four glyphs $\{g_{f_i}^c \mid t = 1, 2; i = 1, 2\}$ expressing the font $f_i$ for the character $c_i$. For the objective function to train $F$, we use cosine similarity given by $\text{sim}(u, v) = \frac{u^T v}{\|u\|\|v\|}$ as the dot product between L2 normalized $u$ and $v$, where $u, v$ are the font representations. We train the model $F$ to map the glyphs from the same font into similar representations and those from different fonts into discriminative representations. That is, we maximize $\text{sim}(F(g_{f_i}^{c_1}), F(g_{f_i}^{c_2})), t = 1, 2$ and minimize $\text{sim}(F(g_{f_i}^{c_1}), F(g_{f_j}^{c_i})), i = 1, 2$. Glyphs of the same character look alike in the image space, even though their fonts are different from one another. However, the aforementioned objective drives the different font glyphs of the same character to be embedded far away from one another in the latent space. That is, we train the model $F$ to focus on the font style of a glyph more than the shape of a character.

To generalize \textit{Paired-glyph Matching} with a minibatch of $N$ fonts, we randomly sample fonts $\{f_1, f_2, \ldots, f_N\}$ from the training set. We randomly sample two different glyph images for each font as $\{(g_{f_n}^{C_1}, g_{f_n}^{C_2}) \mid n = 1, \ldots, N\}$. That is, for all $n$ in $1 \leq n \leq N$, there are $N$ positive glyph pairs in the minibatch. Therefore for each glyph, remaining $2(N - 1)$ glyphs are negative samples. Our model $F$ maps every glyph images in the minibatch into font representation vectors in the latent space. The similarity of the embedding fonts for positive pairs and for negative pairs are defined by

\[
\text{pos-sim}(f_n) = \exp \left( \text{sim} \left( F(g_{f_n}^{C_1}), F(g_{f_n}^{C_2}) \right) / \tau \right),
\]

\[
\text{neg-sim}_k(f_n, f_m \neq n) = \sum_{i=1}^{2} \exp \left( \text{sim} \left( F(g_{f_n}^{C_1}), F(g_{f_m}^{C_i}) \right) / \tau \right),
\]

where $\tau$ is temperature scaling parameter. Then final loss $L$ is sum of losses for each learning
font \( f_n \) is given by

\[
\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \left( -2 \log \frac{\text{pos-sim}(f_n)}{\text{pos-sim}(f_n) + \sum_{k=1}^{N} \text{neg-sim}_k(f_n, f_{m \neq n})} \right). \tag{4}
\]

The loss (4) is derived from “the normalized temperature-scaled cross entropy loss” [5].

4 Experiments

4.1 Baselines

Figure 3 shows baselines of font representation learning technique and our method Paired-glyph Matching. These methods all share font embedding network \( F \) as their backbone network. We consider the output of \( F \) from glyph \( g, \hat{f} = F(g) \) as font embedding. Comparing font representation learning baselines (i.e., Classification [32], Style Transfer [40, 42], Autoencoder [30, 37], Attribute Prediction [4, 24] and Srivatsan et al. [28]) are more described in Section A of the supplementary material.

4.2 Datasets

O’Donovan et al. [24] dataset contains 1,088 fonts for the training set (\( \mathcal{F}_{\text{train}} = \{ f_i | 1 \leq i \leq 1,088 \} \)) and 28 fonts for the validation set. Each font contains 62 alphanumeric characters (\( \mathcal{C}_{0:Z} \)). Thus, there are total 1,088 \( \times 62 \) glyph images in training set. Among the fonts in the training set, each font in \( \{ f_i | 1 \leq i \leq 120 \} \) is annotated by 37 attributes. Each attribute is described by a high-level expression, such as “dramatic” or “legible”. Each attribute value ranges from 0 to 1. The attribute value vector of each font in \( \{ f_i | 1 \leq i \leq 120 \} \) is denoted by \( a_i \in \mathbb{A} \), where \( \mathbb{A} \) is the attribute set, i.e., \( \mathbb{A} = \{ a_1, a_2, \ldots, a_{120} \} \), \( \forall a_i \in [0, 1]^{37} \). The remaining fonts of the training set (i.e., \( \{ f_i | 121 \leq i \leq 1,088 \} \)) are not annotated by any attributes.

Open Font Library (OFL), which is provided by Google Fonts\(^1\), provides 1,076 typefaces (font families). A typeface consists of several fonts that share a specific design. In this paper, we do not consider typeface, thus, fonts in a typeface are regarded as different fonts. For instance, the typeface “Bauer Bodoni” includes “regular”, “bold”, and “italic” fonts, which

\(^1\)https://github.com/google/fonts
are considered different fonts in our work. Finally, we collected 3,802 fonts for the alphanumer- 
cmic character set (C₀-Z). We randomly partitioned 3,702 fonts for the training set and the 
remaining 100 fonts for the validation set. Since these fonts are provided in “ttf” and “otf” 
file formats, we converted each font file into 62 glyph images.

Capitals64 [1], which was used by Srivatsan et al. [28], contains capital letters (Cₐ-Z). 
The dataset is split into train, validation, and test sets of 7,649, 1,473, and 1560 fonts, re-
spectively. We used this dataset to compare our method with Srivatsan et al. method.

### 4.3 Implementation Details

Throughout all experiments, we used a single NVIDIA 2080ti or 1080ti gpu. We did not 
observed a performance boost by tuning the last dimension of the projection head, as the 
previous research [5]. Random sized crop augmentation was only used in our Paired-glyph 
Matching and Attribute Prediction as it degrades the retrieval mean accuracy of other base-
lines. The batch size was 64 samples for each font, and the image input size was 64 × 64. 
Bigger image size did not gain benefit on the retrieval mean accuracy score. We used glyphs 
representing C₀-Z for the O’Donovan and OFL datasets and Cₐ-Z for the Capitals64 dataset. 
We used the Adam [17] optimizer with a learning rate of 2e−4 for all models and datasets. 
We used ResNet18 [12] as the backbone network of font embedding network F for all mod-
els because other deeper neural network architectures were not effective. Font embedding 
was average pooled vector from output of the backbone network. The temperature scaling 
parameter τ of Equation 2 has been used as 0.1 for the OFL and Capitals64 dataset and 0.2 
for the O’Donovan dataset.

Denoting feature dimension by feat_dim, we used feat_dim = 512 for all models for 
training the O’Donovan dataset and feat_dim = 1,024 for all models for the big-
ger OFL dataset. We used 5 transposed convolution layers and a last up-sample layer for 
the generator network G of Autoencoder and Style Transfer models to generate 64 × 64 di-
mensional images from font embedding vectors. The last 4 transposed convolutions were 
followed by self-attention modules [38, 43] and instance normalization [34]. The generator 
G of Style Transfer accepts (feat_dim + |C|)-dimensional vector, which is concatenation 
of font and one-hot character embedding. Denoting a fully connected layer of the weight ma-
trix d₁ × d₂ as FC(d₁×d₂), the Classification head (Fcls) is FC(font_dim×|F_train|), the Attribute 
Prediction head (Fattr) is FC(font_dim×37). Following previous research [5], we also used a 
projection head and L2 feature normalization on Paired-glyph Matching. The projection 
head is FC(font_dim×font_dim) − ReLU − FC(font_dim×70). More details (e.g., the gener-
ator G architecture of Style Transfer and Autoencoder) are presented in Section B of the 
supplementary material. Codes are available at https://github.com/junhocho/
paired-glyph-matching.

### 4.4 Experimental Results

To evaluate how well glyphs in a font are embedded in the latent space, we use the retrieval 
mean accuracy (MACC(Ĉₐ-Z)) as described in Section C of the supplementary material.

#### 4.4.1 Evaluation on Unseen Fonts (O’Donovan and OFL datasets)

Table 1 presents the performances on font embeddings ŵ of all methods (i.e., Paired-glyph 
Matching, Classification, Style Transfer, Autoencoder and Attribute Prediction) depending on
training data portion in the O’Donovan dataset. For every 100 epochs until 15,000 epochs, we evaluated the models on the O’Donovan validation set with the retrieval mean accuracy. Note that models had not seen fonts in the validation set. We found and reported the best score on the O’Donovan validation set and then evaluated the model with same weights on the OFL validation set. First of all, we compare when training only small portion in the O’Donovan dataset. For every 100 epochs until 15,000 epochs, we evaluated the models on the O’Donovan validation set with the retrieval mean accuracy. Note that models had not seen fonts in the validation set. We found and reported the best score on the O’Donovan validation set and then evaluated the model with same weights on the OFL validation set. Since, we jointly trained Paired-glyph Matching, Classification, Style Transfer, Autoencoder with Attribute Prediction ($F_{\text{attr}}$ in Figure 3 (b)) and reported as $\{f_i | 1 \leq i \leq 120\} + A$ in the data portion column of Table 1. We found training font attributes (+A) to have no significant difference in Paired-glyph Matching and Classification. This indicates that font attribute data may not be worth the high annotation cost to train font representations.

Table 2 presents the performances of all font embedding methods trained on the OFL dataset. We found and reported the best score on the OFL validation set until 25,000 epochs and then evaluated the model with same weights on the O’Donovan validation set. Since,
Figure 4: (a) Font latent space of (a) the OFL dataset and (b) the O’Donovan dataset annotated by font classes for each embedding methods. The Red and the cyan boxes respectively include font classes \( \{f_1, f_2, f_3, f_4, f_5\} \) and \( \{f_6, f_7, f_8, f_9\} \).

there are more possible solutions (e.g, triplet loss [26] or other self-supervised methods [6, 14, 35]) to learn similarities in paired-glyph matching learning, we include Paired Glyph Matching †, ‡ which are respectively trained with losses based on deep clustering algorithm (PICA) [14] and triplet loss [26]. We observed that paired-glyph matching learning with the loss (4) performed the best compare to other similarity learning approaches †, ‡ [14, 26].

To visually understand how comparing methods perform, we used T-SNE [36] projection on the font latent space as in Figure 4. From observations in the font latent space of the OFL dataset (Figure 4 (a)) and the O’Donovan dataset (Figure 4 (b)), glyphs in a font were better clustered in the order of Paired-glyph Matching, Classification, and Style Transfer. In particular, note the red and cyan boxes in Figure 4 (b). Style Transfer and Classification methods do not distinguish the glyphs of the fonts \( f_1, f_2, f_3 \) in the red box and \( f_6, f_7 \) in the cyan box, but our method distinguished them relatively well.
Table 3: Performance evaluation ($\text{MACC}_{\text{Ret}}(\mathcal{C}_{A-Z})$ and Font attribute prediction) on the Capitals64 dataset. All methods are trained on Capitals64, validated and tested on Capitals64. Font attribute prediction is evaluated on O’Donovan dataset with $L_1$-error.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Captitals64 valset $\text{MACC}<em>{\text{Ret}}(\mathcal{C}</em>{A-Z})$</th>
<th>Captitals64 testset $\text{MACC}<em>{\text{Ret}}(\mathcal{C}</em>{A-Z})$</th>
<th>O’Donovan $L_1$-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{f}$ of Paired-g Matching</td>
<td>61.38</td>
<td>62.66</td>
<td>0.09589</td>
</tr>
<tr>
<td>$\hat{f}$ of Classification [32]</td>
<td>55.27</td>
<td>56.31</td>
<td>0.1275</td>
</tr>
<tr>
<td>$\hat{f}$ of Style Transfer [40, 42]</td>
<td>32.22</td>
<td>32.53</td>
<td>0.1217</td>
</tr>
<tr>
<td>$\hat{f}$ of Autoencoder [30, 37]</td>
<td>13.60</td>
<td>14.16</td>
<td>0.1312</td>
</tr>
<tr>
<td>$\hat{f}$ of Srivatsan et al. [28]</td>
<td>11.72</td>
<td>11.56</td>
<td>0.1097</td>
</tr>
</tbody>
</table>

Figure 5: Font latent space of Paired-glyph Matching and Srivatsan et al. [28] annotated by font attributes (i.e., Angular, Attractive, Boring, Delicate, Modern, Strong, Thin). Both methods are trained on Capitals64 and tested on the O’Donovan dataset.

### 4.4.2 Evaluation on Unseen Fonts (Capitals64 dataset)

Table 3 presents the performances of font representation learning methods (i.e., Paired-glyph Matching, Classification, Style Transfer, Autoencoder and Srivatsan et al. [28]) on the Capitals64 dataset and the O’Donovan dataset. Similar to Table 1 and 2, our method performs the best in the retrieval mean accuracy $\text{MACC}_{\text{Ret}}(\mathcal{C}_{A-Z})$ measure. To more quantitatively evaluate representation power of $\hat{f}$, we trained font attribute ($\mathbb{A}$) prediction task, which is similar to linear evaluation protocol [5]. That is, we train a linear mapping from font embedding $\hat{f}$ of each method to 37 font attributes and validate with $L_1$-prediction error. We trained 120 fonts and validated 28 fonts in O’Donovan dataset, varying learning rate in range of $[1e^{-6}, 1e^{-5}, 1e^{-4}, 1e^{-3}, 1e^{-2}]$ and reported the lowest $L_1$-error in Table 3 last column. Our method outperformed Srivatsan et al. by predicting font attributes with lower error.

In Figure 5, we observed the latent space of the O’Donovan fonts with attribute annotations $\mathbb{A}$. Refer to Srivatsan et al., we took max-pooling operation on embeddings of glyphs in a font and regarded it as the font embedding. Each font in the O’Donovan dataset is colored with respective attribute value in Figure 5. Despite not training on font attribute data ($\mathbb{A}$), both methods gathered fonts according to values of the font attributes.
4.4.3 Transfer Learning to Font Style Transfer & Generation.

In this experiment, we checked the transfer learning performance in font style transfer (See Section A.4) and font generation (Attr2Font [41]) as downstream tasks. We used pretrained weights from the best-performing models $F$ (i.e., Paired-glyph Matching, Classification, Style Transfer and Autoencoder from Table 2) on the OFL dataset and applied transfer learning to O’Donovan dataset, which is smaller the OFL dataset. To evaluate the generation quality of font style transfer model, we calculated average $L_1$ errors for all images generated from a input glyph and an one-hot character embedding as follows:

$$L_1\text{-error} = \frac{1}{|\mathbb{F}_{\text{val}}| \times I_{\text{dim}}} \sum_{f \in \mathbb{F}_{\text{val}}} \sum_{C_i, C_j \in C} \| G \left( F(g^C_i), c^C_j \right) - g^C_j \|_1,$$

where $I_{\text{dim}} = H \times W \times C$ is number of pixels in an image. For the Attr2Font model [41], which performs attribute-based font generation as a downstream task, we initialized the “style encoder” with the aforementioned pretrained weights, and $L_1\text{-error}$ is similarly defined. Note that we scratch-train the generator $G$ weights of Autoencoder and Style Transfer. In Figures 6, we measured performance gains of pretrained models over random initialized baseline. Interestingly, the models trained in the generative way (i.e., Autoencoder, Style Transfer) on the OFL dataset seemed to be better in the downstream generative tasks than the model trained through Classification. As a result, we determined that Paired-glyph Matching performed the best, showing that our method can be useful as transfer learning to the generative tasks.

5 Conclusion

In this paper, we proposed a new discriminative font embedding method that attracts the representations of glyphs in the same font to one another but pushes away glyphs in other fonts. Our method needed neither a generator network nor font attribute tags because we actively take advantage of Glyph-font-consistency. Through extensive evaluation, we show our model outperformed the conventional representation learning techniques for generalization to unseen fonts. Finally, we confirmed the benefits of our method for transfer learning in the font style transfer and generation tasks.
Acknowledgement

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [NO.2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University)] and IITP grant funded by Korea government(MSIT) [No.B0101-15-0266, Development of High Performance Visual BigData Discovery Platform for Large-Scale Realtime Data Analysis].

References


