Coconditional Autoencoding Adversarial Networks for Chinese Font Feature Learning

Zhizhan Zheng
Hanyi Fonts
zhizhanzh@zju.edu.cn

Feiyun Zhang
Qiniu Atlab
zhangfeiyun@qiniu.com

Abstract

In this work, we propose a novel framework named Coconditional Autoencoding Adversarial Networks (CocoAAN) for Chinese font learning, which jointly learns a generation network and two encoding networks of different feature domains using an adversarial process. The encoding networks map the glyph images into style and content features respectively via the pairwise substitution optimization strategy, and the generation network maps these two kinds of features to glyph samples. Together with a discriminative network conditioned on the extracted features, our framework succeeds in producing realistic-looking Chinese glyph images flexibly. Unlike previous models relying on the complex segmentation of Chinese components or strokes, our model can "parse" structures in an unsupervised way, through which the content feature representation of each character is captured. Experiments demonstrate our framework has a powerful generalization capacity to other unseen fonts and characters.

1 Introduction

The Chinese writing system and through it a sense of a common literature and history is the very fabric that held China together for millennia. But unlike phonetic writing system which have very limited number of letters such as English, Chinese has a huge amount of ideographic characters(more than 80000). On the other hand, most Chinese characters have more complex shapes and structures than other symbolic characters. That is why designing a new Chinese font is such an expensive and difficult work, it needs a group of type designers and calligraphers working together for years for a typerface covering official character set like GB-18030, which contains 27533 unique characters. Due to this fact, we could seldomly see independent artists working on Chinese typerface design, which also leads to the status quo that there existed fewer well-designed Chinese digital fonts than alphabetic ones. To alleviate the labor intensive part in this design process, various automatic synthesizing approaches have been proposed these years, among them, machine learing method is a promising solution.

In this work, we break the Chinese font learing task down into 2 subtasks (the encoding part and the adversarial), and fufill each with 2 convolutional networks. The encoding networks are designed to disentangle content and style features separately, with a pairwise substitution optimization we force the networks to capture lattent embeddings of feature from different domains. The overall process of our proposed framwork is shown in Fig. 1. The main contributions of this work are:

- We propose a novel model which can disentangle each Chinese glyph image into content and style representations automatically, and with those two kinds of feature embedding, the model can generate specified glyph in clear appearance.
- We incoporate the adversarial netwroks with the auto-encoders, and propose an adversarial way to train the auto-encoders.

- We demonstrate the potention of our proposed framwork to be generalized to new fonts and other characters beyond the training sets.

2 Related Work

Early researches [15, 18] on synthesizing Chinese characterss are mostly stroke-based methods which formulate the target character generation as a process of assembling all needed components segmented from training sets. So one of the most important aims of their models can be sumarized as finding a suitable algorithm to decompose Chinese characters into hierarchical representation of simple radicals and strokes. This process resembles the human being's way to learn Chinese characters' structure, as it is apparent that the Chinese characters do constitute of repeated componets. However, the intrinsic assembling grammer is not that straightforward as it seemed, there always are some complicated characters that could hardly be correctly decomposed. Apart from the hevily rely on the preceding parsing, those stroke-based methods paid little attention on writing style transfering other than the local representations of the characters, which also prevents its application in the Chinese character generation.

Zhang et al.[16] proposed a sequence-based method which uses Recurrent Neural Networks (RNN) to extract temporal information of ordinal hand-written strokes to generate skeletons of Chinese characters. Since the generated skeletons by this method contain only the structure appearance but no style information, it can not generalize to different style glyphs.

zi2zi[1] and other relevant methods [1, 2] implements the Chinese characters style transfer based on pix2pix[5], which considers the generation as a problem of mapping from the source domain to the target pairwisely. Theoretically, different glyphs of a same font do share the same writing style, there should be a fixed mapping for any glyphs of two uniformly designed fonts. However, this kind of methods only concentrate on the mapping relationship in a specified font pair, while ignoring a lot more font resouces which would also lead to a lack of flexibility for new style learning, as for every new font pair, the netwroks must be retrained from scratch.

Sun et al.[13] invoked an intercross pairwise scheme to infer the common style feature, which takes advantage of the implicit style co-sharing nature of different fonts. But for the character content feature they used a manual encoding method based on the radical assembling knowledge of each character, which could hardly be generalized to sophisticated structure as the stroke-based methods. Moreover, the generated samples are often blurry, which should attribute to the disadvantage of its varitional auto-encoder (VAE) mechanism as it uses the pixel-wise loss for reconstruction objective.

3 Methodology

As shown in Fig. 1, our proposed CocoAAN consists of two stages, the encoding part and the adversarial part, each of which consists two subnets, i.e., the former includes the style and content subnets (the blue and red trapezoids in Fig. 1) while the latter involves the generator and discriminator subnets (the yellow and green trapezoids in Fig. 1) . In this section, we will present all details of each subnet and the process of optimization.

3.1 Assumptions and Encoding Networks

In contrast to the challenges mentioned above, Chinese fonts are particularly well suited for cross domain transfer learing, since they naturally are database of aligned style pairs and content pairs. A font is a collection of glyphs which representing different characters, from a same font, every glyph should share a specific weight, width, slant, ornamentation, i.e., they share a set of common design features, while across fonts, glyphs of same character should share a same radical assembing relationship and inherent stroke structure.

Assumption 1 *Glyphs of one same character but from different fonts share a same content feature;*

[1] https://github.com/kaonashi-tyc/zi2zi/

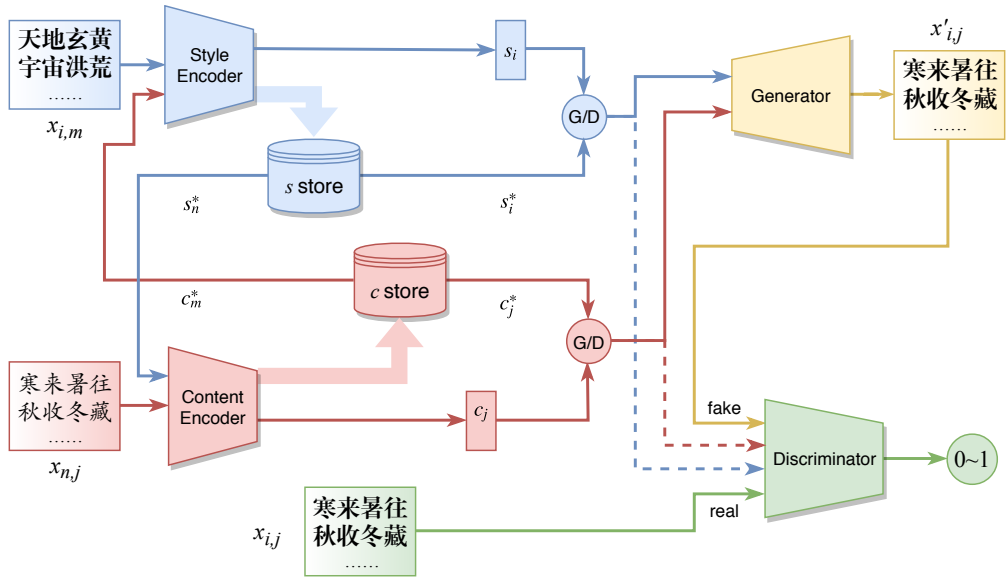

Figure 1: An overview of the process of CocoAAN. (Best viewed in colors). $x_{i,j}$, $x_{i,m}$ and $x_{n,j}$ denote training glyph samples of different fonts and characters, $x'_{i,j}$ denote the reconstructed sample of font i and character j, $\boldsymbol{s_i}$ and $\boldsymbol{c_j}$ denote the style and content representation extracted from corresponding glyph sample, while $\boldsymbol{s_i^*}$, $\boldsymbol{s_n^*}$, $\boldsymbol{c_j^*}$ and $\boldsymbol{c_m^*}$ are feature vectors feeding from the style store or content store.

Glyphs of different characters but from one same font share a same style feature.

Moreover, we agreed the assumption from Sun et al.[13] that a glyph can be characterized by, and only by two sides of feature:

Assumption 2 *Every glyph contains features from two independent domains, the style factor and the content factor, by which uniquely determined a glyph, which can be formulated as:*

$$x_{i,j} \rightleftharpoons (\boldsymbol{s_i}, \boldsymbol{c_j}) \tag{1}$$

where $x_{i,j}$ represents the glyph of font i and character j, $\boldsymbol{s_i}$ and $\boldsymbol{c_j}$ denote the style and content feature of the specified glyph respectively.

Upon Assumption 2, the key to solving this problem is decoupling every glyph $x_{i,j}$ into its style feature $\boldsymbol{s}$ and content feature $\boldsymbol{c}$ representations. As soon as we obtain the precise feature embeddings, we could reconstruct any glyph from existing style code $\boldsymbol{s}$ and content code $\boldsymbol{c}$ flexibly. In our approach, we use two encoding networks to capture feature codes respectively from glyph samples. To decisely disentangle these two hidden features, besides the glyph image $x_{i,j}$, either feature embedding $\boldsymbol{s_i}$ or $\boldsymbol{c_j}$ should also be as the input of these two networks. Let $\mathcal{C}$ and $\mathcal{S}$ be the corresponing encoding networks, the feature extracting process can be considered as a function of x and $\boldsymbol{s}$ or $\boldsymbol{c}$, then the co-sharing nature of Assumption 1 can be illustrated as:

$$\boldsymbol{c_j} = \mathcal{C}(x_{i,j}, \boldsymbol{s_i^*}) = \mathcal{C}(x_{n,j}, \boldsymbol{s_n^*}) \tag{2}$$

$$\boldsymbol{s_i} = \mathcal{S}(x_{i,j}, \boldsymbol{c_j^*}) = \mathcal{S}(x_{i,m}, \boldsymbol{c_m^*}) \tag{3}$$

where $\boldsymbol{s_i^*}$, $\boldsymbol{s_n^*}$, $\boldsymbol{c_j^*}$ and $\boldsymbol{c_m^*}$ denote different style or content features which were treated as a known input of the conditional networks, specifically, they were supplied from the encoding results of the previous stage of network $\mathcal{S}$ or $\mathcal{C}$.

3.2 Adversarial Networks

Generative adversarial networks (GAN)[3] establishes a min-max adversarial game between its generattor $\mathcal{G}$ and discriminator $\mathcal{D}$: $\mathcal{G}$ produces model distribution mapped from a latent random noise, and $\mathcal{D}$ distinguishes the model distribution from the target. By consecutively training the model with the minimax objective in turn:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{x\sim p_{\text{data}}}[\log \mathcal{D}(x)]+\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1-\mathcal{D}(\mathcal{G}(\boldsymbol{z})))] \tag{4}$$

it can achieve impressive results in a wide range of generative scenarios, as with its discriminator network it provides a more abstract metric for complex data distribution than other element-wise reconstruction errors.

The conditional GAN (cGAN)[8] is an extension of the vanilla GAN where both $\mathcal{D}$ and $\mathcal{G}$ receives an additional condition $\boldsymbol{y}$ as input, which controls the domain of the output distribution. Our approach is based on the framework of cGAN with two kinds of conditions, but no random noise $\boldsymbol{z}$ as input. Conditioned on $\boldsymbol{c_j}$ and $\boldsymbol{s_i}$ captured by the abovementioned encoding networks, the adversary between generator($\mathcal{G}$) and discriminator($\mathcal{D}$) can be expressed as alternatively maxmizing $L(\mathcal{G}^*, \mathcal{D})$ in Eq. (5) and minimizing $L(\mathcal{G}, \mathcal{D}^*)$ in Eq. (6):

$$\begin{aligned} L(\mathcal{G}^*, \mathcal{D}) = \mathbb{E}_{x_{i,j}\sim p_{\text{data}}}[\log \mathcal{D}(x_{i,j}|\boldsymbol{s_i}, \boldsymbol{c_j})]+ \\ \mathbb{E}_{x'_{i,j}\sim p_{\{G^*(\boldsymbol{s_i^*}, \boldsymbol{c_j^*})\}}}[\log(1-\mathcal{D}(x'_{i,j}|\boldsymbol{s_i}, \boldsymbol{c_j}))] \end{aligned} \tag{5}$$

$$L(\mathcal{G}, \mathcal{D}^*) = \mathbb{E}_{x'_{i,j}\sim p_{\{\mathcal{G}(\boldsymbol{s_i}, \boldsymbol{c_j})\}}}[\log(1-\mathcal{D}(x'_{i,j}|\boldsymbol{s_i^*}, \boldsymbol{c_j^*}))] \tag{6}$$

where p_{data} and $p_{\{\mathcal{G}(\boldsymbol{s_i}, \boldsymbol{c_j})\}}$ correspondingly denotes the distribution of true data and data generated from the given inputs $\boldsymbol{s_i}$ and $\boldsymbol{c_j}$. "$*$" labeled in upper right of the network's symbol indicates the network's parameters was fixed in current iteration.

By Equation (5), the parameters of $\mathcal{S}$ and $\mathcal{C}$ will be updated by gradient backpropped from $\mathcal{D}$, in other words, $\mathcal{S}$ and $\mathcal{C}$ act as auxiliary networks to make $\mathcal{D}$ a better discriminator in the discriminative iteration. On the other hand, while in the generative iteration, $\mathcal{S}$ and $\mathcal{C}$ help to make $\mathcal{G}$ a better generator to "fool" $\mathcal{D}$ by Eq. (6). From this point, both $\mathcal{S}$ and $\mathcal{C}$ of our approach are adversarial for their function in the training process, together with $\mathcal{G}$ and $\mathcal{D}$.

However, if $\mathcal{D}$ was excluded, the rest three subnets will constitute a conventional auto-encoder, for $\mathcal{G}$ to be the decoder while $\mathcal{S}$ together $\mathcal{C}$ to be the encoder's part. Thus make it posssible to add a pixel-wise reconstruction term to the loss function in additional fooling the discriminator. We choose L_1 loss as an additional term to enhance the similarity meric, this is a highly popular choice used in many related auto-encoder GAN variants, such as α-GAN, PPGN, and PPGN[12, 17, 10]. Then the hybrid objective function for generative iteration can be sumarized as:

$$\begin{aligned} L(\mathcal{G}, \mathcal{D}^*) =& \mathbb{E}_{x'_{i,j}\sim p_{\{\mathcal{G}(\boldsymbol{s_i}, \boldsymbol{c_j})\}}}[\log(1-\mathcal{D}(x'_{i,j}|\boldsymbol{s_i^*}, \boldsymbol{c_j^*}))+ \\ & \lambda\|x'_{i,j} - x_{i,j}\|_1] \end{aligned} \tag{7}$$

where λ is a scale parameter that balances two terms in Eq. (7). We found the reconstruction term benificial to alleviate the mode-collapse degree of the adversarial model and accelerate the convergence of the min-max process in practice.

3.3 Optimization Algorithm

However, only with adversarial optimization on Eq. (5) and Eq. (7) is not sufficient to achieve our task, it still won't ensure the features from $\mathcal{S}$ and $\mathcal{C}$ are reliable decoupled. This is because we haven't taken advantage of the feature co-sharing nature of glyph distribution yet, which is implied in Assumption 1.

As Eq. (3) suggests, let $\{x_{i,j}\}$ and $\{x_{i,m}\}$ be two mini-batches paired in style, i.e., every sample pairs in these two mini-batches come from a same font set, we could also infer the corresponing style embeddings of the glyphs in mini-batch $\{x_{i,j}\}$ by feeding $\{x_{i,m}\}$ into $\mathcal{S}$. For the same reason, the content feature could also be infered with another mini-batch $\{x_{n,j}\}$, which paired with $\{x_{i,j}\}$ in character, by the deep network $\mathcal{C}$.

Therefore, in Eq. (5), Eq. (6) and Eq. (7), we use $\boldsymbol{s_i}$ and $\boldsymbol{c_j}$ extracted from $\{x_{i,m}\}$ and $\{x_{n,j}\}$, to get the point of equilibrium of the adversarially training. Such a pairwise substitution extracting strategy prevents the well-trained $\mathcal{G}$ and $\mathcal{D}$ from relying on the low-level semantics of raw glyph images, thus forces

the high level features to be decoded during training. The overall flow of this substitution optimization is illustrated in Fig. 1, detailed algorithms are in the supplementary material.

During the optimization we keep two feature code stores of stlye and content subsects, i.e., $\mathbb{Z}_s$ and $\mathbb{Z}_c$, to feed the necessary known inputs as in Eq. (2-3) and Eq. (5-6). As we want the content of $\mathbb{Z}_s$ and $\mathbb{Z}_c$ newest to the current model, after updating the parameters of $\mathcal{S}$ and $\mathcal{C}$ in every iteration, we will randomly select two extra mini-batches to get new ranges of feature code. To eliminate errors as much as possible, each feature embedding vector of the specific style or content will be averaged:

$$\hat{\boldsymbol{s_i}} = \frac{1}{n_s}\sum_{k=1}^{n_s} s_i^{(k)} \tag{8}$$

$$\hat{\boldsymbol{c_j}} = \frac{1}{n_c}\sum_{l=1}^{n_c} c_j^{(l)} \tag{9}$$

n_s, n_c are the number of repeated glyph of font i and character j in two extra mini-batches, then $\mathbb{Z}_s$ and $\mathbb{Z}_c$ will be updated with these averaged codes. But from the very begining, nothing is in $\mathbb{Z}_s$ and $\mathbb{Z}_c$, so we randomly inialize all known feature codes($\boldsymbol{s_n^*}$, $\boldsymbol{s_i^*}$, $\boldsymbol{c_m^*}$, $\boldsymbol{c_j^*}$ in Fig. 1) from $\mathcal{N}(0, \boldsymbol{I})$; after the very first iteration, all the training mini-batch will be sampled in the range where $\mathbb{Z}_s$ and $\mathbb{Z}_c$ covers, so every necessary known inputs could be found in these two stores.

4 Experiments

4.1 Data Preparation

For this task, we build a glyph images (128×128 pixels) dataset from 60 regular Chinese fonts, 50 for traing while the rest for testing. Basically, these fonts are in the conventional Chinese font style categories known as songti, heiti, kaiti and fangsong, but each varies in its weight, condensation, width, ornamentation. Each font we selected includes at least the 6763 characters in GB-2312, which covers over 99% of the characters of contemporary usage. However, we believe that any glyph dataset that build on a sufficient number of Chinese text fonts covering thousands of characters would work in our proposed CocoAAN model, a small scale benchmark has validated this.

4.2 Basic Model Setting

All the four subnets are based on the general architecture of CNN models, details are sumarized in the supplementary material. To stabilize the training of GAN model, we applied spectral normalization[9] for the layers in discriminator, which will constrain the Lipschitz constant of the discriminator by restricting the spectral norm of each layer. As the encoding networks play a vital role in discriminative iteration, a drastic escalation of parameters may cause the model unstable, we also used spectral normalization in $\mathcal{S}$ and $\mathcal{C}$.

In most other conditional 2D CNN model's implementation[11], the 1D latent code is integrated into the input by expansion-concatenation operation, i.e., the latent is firstly expanded to the size of feature map, then concatenated into the input in channel. Such an incorporating method will bring a lot of invalid calculation in the subsequent convolution layer, since the newly concatenated feature maps were merely repetition of itself.

We introduce a new incorporating method for lattent code, the Fully Connect-Add (FC-Add) operation. The process is as follows (Fig. 2): the conditional lattent code will firstly be resized to the the current input's channel shape by a fully connect layer, then added to the current input channelwisely. For better conditional attribute learning, this FC-Add operation can be repeated several times at early stages of downsampling networks. As in our case, the very first three convolution layers of $\mathcal{S}$, $\mathcal{C}$, $\mathcal{D}$ are all followed with this modification.

No pre-processing was applied to training images besides scaling to the range of the tanh activation function $[-1, 1]$. All models were trained with Adam optimizer[6] with $\beta_1 = 0$ and $\beta_2 = 0.9$. By default, we use the the two-timescale update rule (TTUR) for the learning rate setting as it is proved to be effective in a regularized discriminator scenario[4]. Learning rate is set to 0.0001, 0.0001, 0.0002, 0.0004 for $\mathcal{S}$, $\mathcal{C}$,

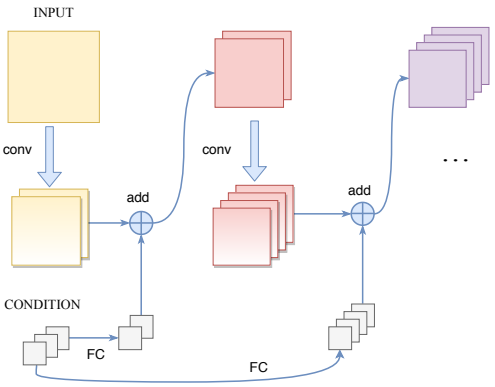

Figure 2: Our proposed Fully Connect-Add (FC-Add) operation.

$\mathcal{G}$ and $\mathcal{D}$, respectively. We set $\lambda = 10$ in Eq. (7) for all the experiments.

4.3 Results

4.3.1 Reconstruction Results

Fig. 3(a) shows parts of the reconstructed samples with the style and content features learned from the training dataset. We can see results of CocoGAN are almost as real as the ground truth, the style and content details are both perfectly recovered except some little flaws. We also compare our results with those trained with VAE model and GAN model within the same iterations, while the settings and optimization strategy keep the same. As shown in the rest parts of Fig. 3, VAE model produces blurry images, while the GAN (without L1 term as objective in generative iteration) method suffers from model collapse, our CocoAAN avoids these issues simultaneously. Considered the experimental networks were not that deep, and the training images were merely in resolution ratio of 128×128, it demonstrates that our model has a capacity to precisely disentangle and restore domain-cross features of Chinese fonts.

4.3.2 Style Learning

We test the capacity of learning new font style with the test dataset, based on the content feature embedding of the 6763 characters extracted from training dataset. Since each font possessed adequate glyphs, we averaged the newly learned style feature vector with a batchsize of 60 by Eq. (8). Fig. 4 lists samples drawn from the generator with the newly learned style features. Be critical, style details are not as fully acquired as the reconstruction results, e.g., glyphs in the 5th line of Fig. 4 totally missed the round serif

(a) CocoAAN samples vs. ground gruth

(b) CocoAAN (c) VAE (d) GAN (e) Ground truth

Figure 3: (a): Comparisons of reconstructed glyphs and the ground truth. Glyphs in odd lines are the reconstructed samples by our CocoAAN, and the adjacent background inversed lines are the ground truth. (b)-(e): Enlarged views of reconstructed glyphs of the same two Chinese characters from three models and the ground truth.

Figure 4: Comparisons of new stlye learning results and the ground truth. The black-background lines are the ground truth. The new style feature embeddings are infered from the test set.

in ground truth, which should be attributed to a lack of diversity of the training dataset. But overall, our CocoAAN can generalize to fonts beyond the training dataset very well.

Figure 5: Linear interpolation results in style latent space. We intentionally choose interpolation between distinct styles, the left side tends to be thin while the right to be bold.

Linear interpolation results in Fig. 5 also show that the discinct styles can gradually transform along the style latent embeding, which proves that our encoding networks can learn a meaningful representation in the style latent space.

4.3.3 Content Learning

Figure 6: Comparisons of new content learning results and the ground truth. The black-background lines are the ground truth. The new content features are infered from glyphs of traditional Chinese characters, which are never appeared in the training set.

Our CocoAAN also provides an automatic solution for encoding new character content features, instead of with manual encoding methods. In this test, we choose two conventional serif and sans-serif fonts as the input batch, which includes a range of GB-2312 unavailable characters, most of which are uncommon-used or more complicated traditional characters. As the manner in Sec. 4.3.2, each newly content embedding vector is averaged by Eq. (9). Fig. 6 shows some result samples with the newly content feature embedding and other style feature in $\mathbb{Z}_s$. Most of the characters can be well learned as a whole, but some results also showed an emergence of touched or overlapped strokes.

We then visualize the content feature embedings of CocoAAN using the t-SNE method [14] in Fig. 7. As shown in Fig. 7(b)-(f), feature embeding of characters with the same or similar radicals are inclined to gather together. Without any hints of structure knowledge about the Chinese characters, our model learns to cluster the data in an unsupervised fashion

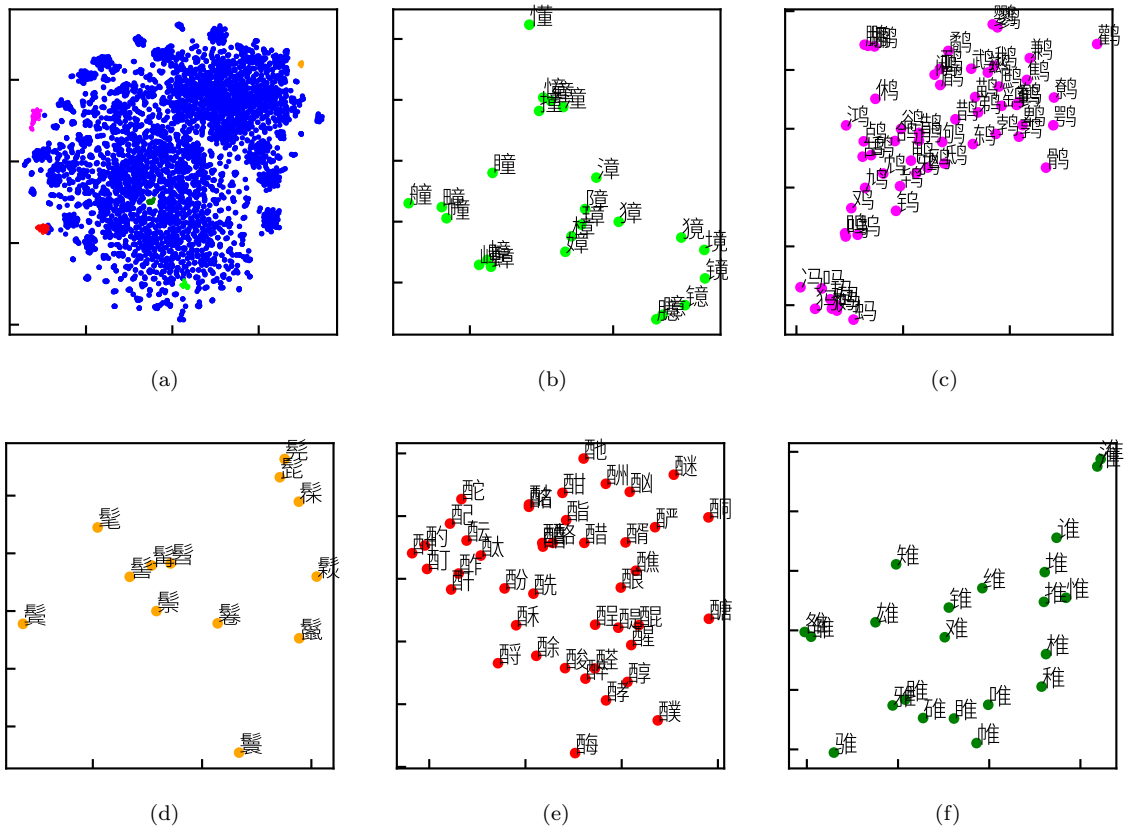

Figure 7: (Best viewed in colors.) 2D Visualizations of the extracted hidden features by the content encoding network. (b)-(f) are partial enlarged views of corresponding color regions in (a). Visualization is conducted with the t-SNE algorithm[14].

automatically. Thus our framwork has successfully driven its content encoding part to capture reliable content representations in a meaningful way.

5 Conclusions and future work

In this paper, we propose a novel framework which jointly learns a generation network to achieve Chinese glyph synthesis and two encoding networks of the style and content related features using an adversarial process. By the pairwise substitution optimization which based on the co-sharing nature of the style and content feature of glyphs, the model could get precise represntations from the style and content domains separately. This framwork shows an impressive ability to generalize to both other unseen fonts and characters.

However, this framework focus more on printed Chinese fonts rather than the handwrittern ones, as the printed fonts are usually more rigidly designed and thus the style feature can be seen as a determined latent embedding which grounds the Assumption 1. For handwritten fonts more uncertainty is around, therefore regarding the style feature as a stochastic distribution may be a better choice. In this scenario,

a varitional variant of the style inference network as VAE-GAN [7] may benefit to capture such an uncertain feature.

References

[1] B. Chang, Q. Zhang, S. Pan, and L. Meng. Generating Handwritten Chinese Characters using CycleGAN. *arXiv:1801.08624 [cs]*, Jan. 2018.

[2] J. Chang and Y. Gu. Chinese Typography Transfer. *arXiv:1707.04904 [cs]*, July 2017.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014.

[4] 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. *arXiv:1706.08500 [cs, stat]*, June 2017.

[5] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. *arXiv:1611.07004 [cs]*, Nov. 2016.

[6] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. Dec. 2014.

[7] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. Dec. 2015.

[8] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. *arXiv:1411.1784 [cs, stat]*, Nov. 2014.

[9] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral Normalization for Generative Adversarial Networks. *arXiv:1802.05957 [cs, stat]*, Feb. 2018.

[10] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space. *arXiv:1612.00005 [cs]*, Nov. 2016.

[11] G. Perarnau, J. van de Weijer, B. Raducanu, and J. M. Álvarez. Invertible Conditional GANs for image editing. *arXiv:1611.06355 [cs]*, Nov. 2016.

[12] M. Rosca, B. Lakshminarayanan, D. Warde-Farley, and S. Mohamed. Variational Approaches for Auto-Encoding Generative Adversarial Networks. *arXiv:1706.04987 [cs, stat]*, June 2017.

[13] H. Sun, Y. Luo, and Z. Lu. Unsupervised Typography Transfer. *arXiv:1802.02595 [cs]*, Feb. 2018.

[14] L. van der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[15] S. Xu, T. Jin, H. Jiang, and F. C. M. Lau. Automatic Generation of Personal Chinese Handwriting by Capturing the Characteristics of Personal Handwriting. In *IAAI*, 2009.

[16] X.-Y. Zhang, F. Yin, Y.-M. Zhang, C.-L. Liu, and Y. Bengio. Drawing and Recognizing Chinese Characters with Recurrent Neural Network. *arXiv:1606.06539 [cs]*, June 2016.

[17] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *arXiv:1703.10593 [cs]*, Mar. 2017.

[18] A. Zong and Y. Zhu. StrokeBank: Automating Personalized Chinese Handwriting Generation. 2014.

A Training Algorithms

Algorithm 1 Training algorithm of CocoAAN

Input: Images of Chinese glyph sets: $X = \{x_{i,j}\}$
Output: The model parameters: $\theta_{\mathcal{C}}$, $\theta_{\mathcal{S}}$, $\theta_{\mathcal{D}}$, $\theta_{\mathcal{G}}$; Stores of feature of style and content subsects: $\mathbb{Z}_s$ and $\mathbb{Z}_c$

Randomly inialize $\theta_{\mathcal{C}}$, $\theta_{\mathcal{S}}$, $\theta_{\mathcal{D}}$, $\theta_{\mathcal{G}}$
for number of training iterations **do**
  $\{x_{i,m}\}$, $\{x_{n,j}\}$, $\boldsymbol{s}_{\boldsymbol{i}}^*$, $\boldsymbol{s}_{\boldsymbol{n}}^*$, $\boldsymbol{c}_{\boldsymbol{j}}^*$, $\boldsymbol{c}_{\boldsymbol{m}}^*$ $\leftarrow$ Preparation mini-batches and known inputs by algorithm 2
  $\{x_{i,j}\} \leftarrow X$, select 1 mini-batch sharing the same style with $\{x_{i,m}\}$ and same content with $\{x_{n,j}\}$
  $\boldsymbol{s_i}, \boldsymbol{c_j} \leftarrow$ Get the feature code with $\mathcal{S}$ and $\mathcal{C}$
  $\theta_{\mathcal{S}}, \theta_{\mathcal{C}}, \theta_{\mathcal{D}} \leftarrow$ Update parameters by ascending with $L(\mathcal{G}^*, \mathcal{D})$:

$$\mathbb{E}_{x_{i,j}\sim p_{\text{data}}}[\log \mathcal{D}(x_{i,j}|\boldsymbol{s_i}, \boldsymbol{c_j})] + \mathbb{E}_{x'_{i,j}\sim p_{\{G^*(\boldsymbol{s}_{\boldsymbol{i}}^*, \boldsymbol{c}_{\boldsymbol{j}}^*)\}}}[\log(1 - \mathcal{D}(x'_{i,j}|\boldsymbol{s_i}, \boldsymbol{c_j}))]$$

  $\{x_{i,m}\}$, $\{x_{n,j}\}$, $\boldsymbol{s}_{\boldsymbol{n}}^*$, $\boldsymbol{c}_{\boldsymbol{m}}^*$ $\leftarrow$ Preparation mini-batches and known inputs by algorithm 3
  Update $\mathbb{Z}_s$ by the output of $\mathcal{S}(x_{i,m}, \boldsymbol{c}_{\boldsymbol{m}}^*)$
  Update $\mathbb{Z}_c$ by the output of $\mathcal{C}(x_{n,j}, \boldsymbol{s}_{\boldsymbol{n}}^*)$
  $\{x_{i,m}\}$, $\{x_{n,j}\}$, $\boldsymbol{s}_{\boldsymbol{i}}^*$, $\boldsymbol{s}_{\boldsymbol{n}}^*$, $\boldsymbol{c}_{\boldsymbol{j}}^*$, $\boldsymbol{c}_{\boldsymbol{m}}^*$ $\leftarrow$ Preparation mini-batches and known inputs by algorithm 2
  $\boldsymbol{s_i}, \boldsymbol{c_j} \leftarrow$ Get the feature code with $\mathcal{S}$ and $\mathcal{C}$
  $\theta_{\mathcal{S}}, \theta_{\mathcal{C}}, \theta_{\mathcal{D}} \leftarrow$ Update parameters by descending with $L(\mathcal{G}, \mathcal{D}^*)$:

$$\mathbb{E}_{x'_{i,j}\sim p_{\{\mathcal{G}(\boldsymbol{s_i}, \boldsymbol{c_j})\}}}[\log(1 - \mathcal{D}(x'_{i,j}|\boldsymbol{s}_{\boldsymbol{i}}^*, \boldsymbol{c}_{\boldsymbol{j}}^*)) + \lambda \|x'_{i,j} - x_{i,j}\|_1]$$

  $\{x_{i,m}\}$, $\{x_{n,j}\}$, $\boldsymbol{s}_{\boldsymbol{n}}^*$, $\boldsymbol{c}_{\boldsymbol{m}}^*$ $\leftarrow$ Preparation mini-batches and known inputs by algorithm 3
  Update $\mathbb{Z}_s$ by the output of $\mathcal{S}(x_{i,m}, \boldsymbol{c}_{\boldsymbol{m}}^*)$
  Update $\mathbb{Z}_c$ by the output of $\mathcal{C}(x_{n,j}, \boldsymbol{s}_{\boldsymbol{n}}^*)$
end for

Algorithm 2 Selecting algorithm of training mini-batches

Input: Images of Chinese glyph sets: $X = \{x_{i,j}\}$; Stores of feature of style and content subsects: $\mathbb{Z}_s$ and $\mathbb{Z}_c$; iteration number i

Output: Two mini-batches: $\{x_{i,m}\}$, $\{x_{n,j}\}$; The necessary knonwn inputs: $\boldsymbol{s_i^*}$, $\boldsymbol{s_n^*}$, $\boldsymbol{c_j^*}$, $\boldsymbol{c_m^*}$

if first iteration **then**

$\{x_{i,m}\}, \{x_{n,j}\} \leftarrow X$, Randomly select 2 mini-batches from X

$\boldsymbol{s_i^*}, \boldsymbol{s_n^*}, \boldsymbol{c_j^*}, \boldsymbol{c_m^*} \leftarrow \mathcal{N}(0, \boldsymbol{I})$, Export necessary known inputs by random noise

else

$\{x_{i,m}\}, \{x_{n,j}\} \leftarrow X$, Randomly select 2 mini-batches from X, in the range covered by both $\mathbb{Z}_s$ and $\mathbb{Z}_c$

$\boldsymbol{s_i^*}, \boldsymbol{s_n^*} \leftarrow \mathbb{Z}_s$, Export necessary known inputs according to corresponing style

$\boldsymbol{c_j^*}, \boldsymbol{c_m^*} \leftarrow \mathbb{Z}_c$, Export necessary known inputs according to corresponing content

end if

Algorithm 3 Selecting algorithm of updating mini-batches

Input: Images of Chinese glyph sets: $X = \{x_{i,j}\}$; Stores of feature of style and content subsects: $\mathbb{Z}_s$ and $\mathbb{Z}_c$; iteration number i

Output: Two mini-batches: $\{x_{i,m}\}$, $\{x_{n,j}\}$; The necessary knonwn inputs: $\boldsymbol{s_i^*}$, $\boldsymbol{s_n^*}$, $\boldsymbol{c_j^*}$, $\boldsymbol{c_m^*}$

if first iteration **then**

$\{x_{i,m}\}, \{x_{n,j}\} \leftarrow X$, Randomly select 2 mini-batches from X

$\boldsymbol{s_n^*}, \boldsymbol{c_m^*} \leftarrow \mathcal{N}(0, \boldsymbol{I})$, Export necessary known inputs by random noise

else

$\{x_{i,m}\} \leftarrow X$, Randomly select 1 mini-batch from X, in the range covered by $\mathbb{Z}_c$

$\{x_{n,j}\} \leftarrow X$, Randomly select 1 mini-batch from X, in the range covered by $\mathbb{Z}_s$

$\boldsymbol{s_n^*} \leftarrow \mathbb{Z}_s$, Export necessary known inputs according to corresponing style

$\boldsymbol{c_m^*} \leftarrow \mathbb{Z}_c$, Export necessary known inputs according to corresponing content

end if

B Network Architecture

Table 1: model architecture of $\mathcal{G}$

$\boldsymbol{s} \to 128 \times 1 \times 1$	$\boldsymbol{c} \to 128 \times 1 \times 1$
$[4 \times 4, 1]$, deconv $\to 512 \times 4 \times 4$, BN, ReLU	$[4 \times 4, 1]$, deconv $\to 512 \times 4 \times 4$, BN, ReLU
concatenate $\to 1024 \times 4 \times 4$	
$[4 \times 4, 2]$, deconv $\to 1024 \times 8 \times 8$, BN, ReLU	
$[4 \times 4, 2]$, deconv $\to 512 \times 16 \times 16$, BN, ReLU	
$[4 \times 4, 2]$, deconv $\to 256 \times 32 \times 32$, BN, ReLU	
$[4 \times 4, 2]$, deconv $\to 128 \times 64 \times 64$, BN, ReLU	
$[4 \times 4, 2]$, deconv $\to 1 \times 128 \times 128$, Tanh	

Table 2: model architecture of $\mathcal{S}$, $\mathcal{C}$

input: $1 \times 128 \times 128$	condition ($\boldsymbol{s}$ or $\boldsymbol{c}$): $\in \mathbb{R}^{128}$
$[4 \times 4, 2]$, SN conv $\to 64 \times 64 \times 64$	FC condition $\to 64 \times 1 \times 1$
add $\to 64 \times 64 \times 64$, LeakyReLU	
$[4 \times 4, 2]$, SN conv $\to 128 \times 32 \times 32$	FC condition $\to 128 \times 1 \times 1$
add $\to 128 \times 32 \times 32$	
$[4 \times 4, 2]$, SN conv $\to 256 \times 16 \times 16$	FC condition $\to 256 \times 1 \times 1$
add $\to 256 \times 16 \times 16$	
$[4 \times 4, 2]$, SN conv $\to 512 \times 8 \times 8$, LeakyReLU	
$[4 \times 4, 2]$, SN conv $\to 1024 \times 4 \times 4$, LeakyReLU	
$[4 \times 4, 2]$, SN conv $\to 128$	

Table 3: model architecture of $\mathcal{D}$

input: $1 \times 128 \times 128$	condition ($\boldsymbol{s}$ and $\boldsymbol{c}$): $\in \mathbb{R}^{128}$
$[4 \times 4, 2]$, SN conv $\rightarrow 128 \times 64 \times 64$	FC condition $\rightarrow 128 \times 1 \times 1$
add $\rightarrow 128 \times 64 \times 64$, LeakyReLU	
$[4 \times 4, 2]$, SN conv $\rightarrow 256 \times 32 \times 32$	FC condition $\rightarrow 256 \times 1 \times 1$
add $\rightarrow 256 \times 32 \times 32$	
$[4 \times 4, 2]$, SN conv $\rightarrow 512 \times 16 \times 16$	FC condition $\rightarrow 512 \times 1 \times 1$
add $\rightarrow 512 \times 16 \times 16$	
$[4 \times 4, 2]$, SN conv $\rightarrow 512 \times 8 \times 8$, LeakyReLU	
$[4 \times 4, 2]$, SN conv $\rightarrow 1024 \times 4 \times 4$, LeakyReLU	
$[4 \times 4, 2]$, SN conv $\rightarrow 1$	