

---

# Universal Solution to Extension and Translation of Fonts Across Languages

---

23<sup>rd</sup> Hong Kong Youth Science & Technology Innovation Competition

Category: Senior - Computer Science and Information Technology

Project ID: SSI0008

## Abstract

There have been numerous attempts at font generation and extension through Artificial Intelligence. Most recent works have adopted a convolutional transfer approach with probability vectors representing characteristics of the character and font styles, which comes with the drawback of a greatly limited character pool. Therefore, we propose an end-to-end model, namely *Universal Solution to Extension and Translation of Fonts Across Languages*, featuring a pairwise comparison methodology between two reconstructed images. Within, one image is the target itself and another image bears both font characteristics of a target and a character. Experiments have also been conducted in search of the optimum model structure to accomplish font extension without the limitations of language. Results have demonstrated the universality of our model, its effectiveness, and its broader impact.

## 1. Introduction

Nowadays, as digital character processing progressively gains greater popularity over handwriting, machines' abilities to rigorously preserve, generate and extend font styles justifies its practicality through a wide variety of applications. Not only does it cater for the growing demand for font-designers, now without the toil of designing letter-by-letter, but it also provides pivotal insight into fields such as handwriting style recognition through the preservation of font styles. This could assist investigative and forensic forces in identifying outlaws through their handwriting style.

Thus far, despite the availability of software capable of transferring font styles across languages, their functionalities are limited. Recent attempts on extending fonts [1][2] have been extracting strokes from input characters, then selected and reassembled into new calligraphy characters. The effectiveness of this approach is dubious, given the complexity of certain characters, the difficulty of stroke separation in cursive font styles and a large dependency on the extracted strokes. While *Generative Adversarial Networks* (GANs) such as the *Multi-content Generative Adversarial Network* (McGan) are commonly used to accomplish related tasks. Yet, as GANs typically have one image as the output, as the dimension of it increases, there is an exponential increase in its memory

usage, bounding the research by memory capacity. This makes GANs unsuitable for our project focusing on fonts with many characters.

Acknowledging the growing importance of font extension regardless of language, in cases such as designers and companies expanding their projects into various languages, this research aims to address such demands by further enabling a universal transfer of font styles across any language. We propose an end-to-end deep neural auto-encoder, utilizing a double *Kullback–Leibler divergence* (K-L divergence) approach, aiming to encode the input and generate an output based on a target image.

## 2. Related works

**Neural Style Transfer** The idea of a neural style transfer can be understood using the example of research by Yijun et al. [3]. In Figure 1, consider the house on the top left-hand corner. We focus on its style, which in this case is the environment with snowy weather and clouds. The next column shows the house for content preservation, of two same-sized houses located next to one another. With this, the end-result (stylized content) would combine the style of the first house and the content of the second. Similarly, characters, no matter in which language, can also be split into style (font) and content (meaning) in a similar manner.

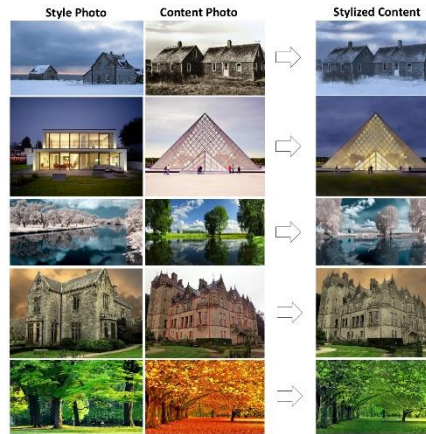


Figure 1: Image style-transfer retaining original content<sup>1</sup>

**Font Generation** The rendering of brand-new fonts is generally referenced as *font generation*. This is usually done by giving a character set and a randomized vector as inputs in the generation process. This category is well-developed, yet most projects focus on the Latin or English alphabet. Methodology-wise, GANs [4] are frequently used, typically consisting of a generator to produce images and a discriminator to grade the quality of the images. A broad spectrum of GANs has been recently applied, such as Multi-Content (MC) GAN where all 26 alphabets are generated mimicking the style of only five provided alphabets [5], GlyphGAN featuring its capability of creating legible, diverse, and consistent fonts [6], CycleGAN which specializes in cross-domain learning [7], DCGANs [8] and DCGAN with class discriminative variations [9].

<sup>1</sup> Yijun, L., Ming-Yu, L., Xueting, L., Ming-Hsuan, Y., & Jan, K. (2018). Stylization of image from given photo [Image]. In A Closed-form Solution to Photorealistic Image Stylization. arXiv:1802.06474 [cs.CV]

**Font Extension** The main idea of *font extension* is to create the rest of a character set, of the same language, given just a few samples of the font in question. This is relevant for languages with complicated characters, such as Chinese. In font extension, it is of pivotal importance to ensure the ability of the network to differentiate style characteristics from that of content. The renowned VGG19 network performs satisfactorily in such task with gram matrices which is capable to detect intricate features, based on calculating the loss of specific genres of content, style and total variation [10][11]. On the other hand, disentanglement methods may also be used, such as the *Kullback–Leibler divergence* (K-L divergence), by assessing the difference between two probability distributions. The K-L divergence has been proved to be apt for capturing characteristics of different font styles, even when involving languages with characters composed of complicated radicals and strokes [12].

**Font Translation** The last category is *font translation*. It refers to the extension of fonts from one language to another. This is the most challenging out of the three categories due to intrinsic differences in character composition and strokes between characters of different languages. This is also the reason for the scarcity of prior research and studies in this aspect. An attempt made by Fernandez **Error! Reference source not found.** investigated the ability of a neural network to transfer the characteristics of fonts from English to Japanese. While it successfully retains the visual characteristics of selected fonts, limitations exist in terms of its options of languages. Equally noteworthy is [14], introducing its methodology of *intercross pairwise reconstruction*, an approach that would increase the reliability of the content and style vectors and thus the consistency of font characteristics.

Inspired by previous works, our project aims to further investigate translating fonts across languages while retaining the features of the source characters. We decided to extend the K-L divergences to greater capabilities as it is possible to measure the mismatch of the standard distribution between the images.

### 3. Data collection and preprocessing

**Language Selection** Given the plethora of languages, we selected languages with characters manifesting distinct characteristics. The languages selected are Traditional and Simplified Chinese, English, Greek, Hebrew, and Japanese (both the *hiragana* and *katakana* letter systems inclusive). While for most languages we considered the entire set of alphabets, the extensive scope of Chinese characters made it hardly possible to take all into account. Thus, we manually selected 250 Chinese characters as seen in Figure 2, trying to include as many common components (radicals and strokes) of Chinese characters as possible.

世丞丹久事 亮仁伊伍似 但侯俗偏停 健僵再凋凹 功劫包卵原  
 及受史吹命 咒哀品哉哲 唐嘉器四垂 埃培堊堡堵 填外天奔套  
 妻娶孀存家 密寇實尖尼 屯岡岳崇崩 巫差巴希帽 平年序庖庚  
 度庭廟弩彗 彩待德志怖 恩悅情想愁 慶憩成我或 戲房才折拋  
 拜挫掃描撒 擦故斑斯施 昏易晨有栗 桃條梵棺極 楷榮模橫檣

次歌此武死 殃段毒毫氓 氣泰港渺滸 滿潮澀炮照 爵犯狗玩琴  
 甘申男疑疼 痰皇盤睡破 硬確福禽稱 童第簿糖統 經緩缺罩義  
 翹考耐耗聚 肅肇脅腐臂 臭舂舅舍舞 艘艱色菠蓉 蕊藕虐蜀蜥  
 融行衡補裝 褐覆觸訊誘 諷謬譽豫賢 赤路身轉辣 逸遞酒野錯  
 長關隊雅霸 靈靠韋顫飯 騷骨魁魚鸞 麟麾黍黝鼠 鼬齒龍龜龠

Figure 2: 250 Chinese Characters selected (Traditional variant adopted)

We gathered more than 3600 fonts and converted the font files of each language into images, pre-processing for training. Figure 3 exhibits how each character in each font is processed from part of a TrueType Font file into images.

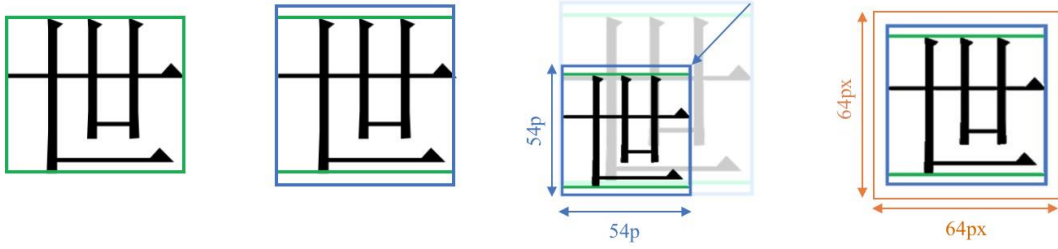


Figure 3: Example of font pre-processing. (From left to right: boundary-detecting & text plotting, square shape rendering, resizing of the image, adding margins)

**Text Plotting** To ensure our neural networks are taking purely the features of the characters and fonts into account, all characters are plotted in black on a white canvas. Instead of plotting the character at the origin (0, 0), we plotted it at (50, 50) due to the offsets certain characters have that would otherwise render it unable to show the full character.

**Boundaries-Trimming** We observed that one of the factors that would affect the performance of the neural network is the size of the fonts. Even when under the same font size, different fonts may have varying actual size for the same character in terms of pixels, making it another obstacle that the neural network would have to face.

Therefore, for each image, we search for the x-coordinates of the leftmost and rightmost black pixels, as well as y-coordinates of the uppermost and lowermost black pixels to determine the boundaries of the rectangle inscribing the character. Only the black pixels, rather than including the grey pixels around some parts of the borders of the characters, are considered as part of the character. This avoids a character from being excessively thick, blurring the image and hindering the neural network from identifying image features.

**Size Correction** The length of the longer side of the rectangle which describes the character is then taken as the side length of a new square canvas. The trimmed character is justified onto the new canvas to render the image with a square shape. The square images are then resized to (54, 54) to retain the consistency of data size. Margins of 5 pixels on all sides are also added, resulting in images of the final size of  $64 \times 64$  pixels.

## 4. Implementation

**The Model with Reinvented Architecture** In typical GAN models, only the designated number of characters, and no more, can be generated. We attempted to increase the number of characters generated, which falls short of expectations due to a substantial requirement for computer memory. Besides, this approach faces another restriction of language, where a *universal font transfer across languages* cannot be attained. With these in mind, we developed a new auto-encoding architecture.

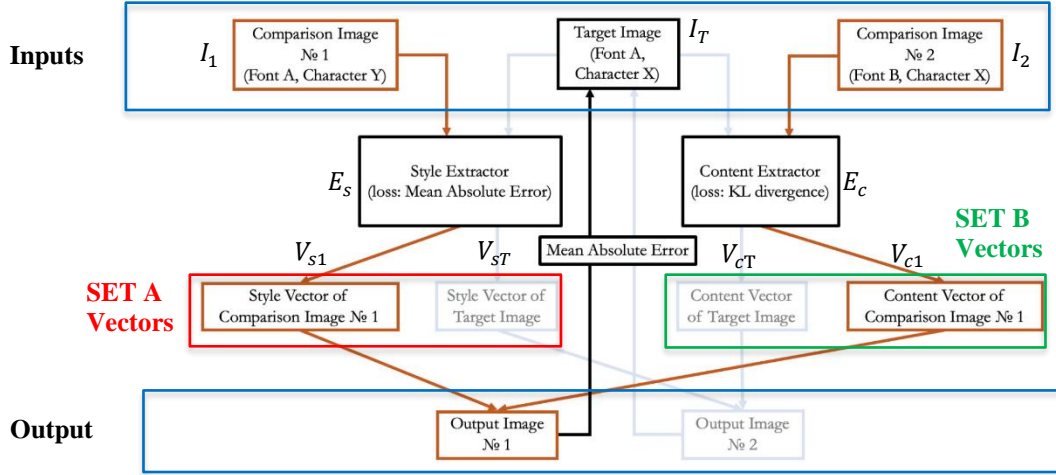


Figure 4: The flow of our model with reinvented architecture

**Ideology** Most auto-encoders either place no constraints on the encoders and the decoders, training them simultaneously [15], or they assign specific tasks to the encoders to classify input images [10]. As our task involves two images as input (in brown), one providing the content information, one providing the style, one simple auto-encoder cannot meet our requirements. Moreover, as our task involves generating characters across languages, we also face the challenge of enormous dictionaries and rigorous character structure requirements.

**Table 1: Mathematical terminology for expressing network components**

| Abbreviation | Description                             | Note                                |
|--------------|---|-------------------------------------|
| $I_T$        | Target Image                            |                                     |
| $I_1$        | Comparison Image No.1                   | Same <i>style</i> as Target Image   |
| $I_2$        | Comparison Image No.2                   | Same <i>content</i> as Target Image |
| $V_{S1}$     | Style vector of Comparison Image No.1   |                                     |
| $V_{ST}$     | Style vector of Target Image            |                                     |
| $V_{C1}$     | Content vector of Comparison Image No.1 |                                     |
| $V_{CT}$     | Content vector of Target Image          |                                     |
| $E_S$        | Style Extractor                         |                                     |
| $E_C$        | Content extractor                       |                                     |
| $D$          | Decoder                                 |                                     |

Therefore, we adopt the *inter-cross pairwise training* method as in Figure 4. The two extractors (in black) would skip the typical execution of classification tasks. In replacement, they are trained upon

their ability to encode the images' information and their ability to separate encoded content and style information. This is largely completed through the comparison of two sets of vectors generated by the two extractors:

**Detailed explanation** Our network can be separated into two sections. First, we train our style extractor, content extractor and decoder simultaneously through the auto-encoding and decoding of images. The output of the decoder is as follows:

$$Output_D = D(E_s(I_{T_n}), E_c(I_{T_n})) \quad (1)$$

This is because we feed our target image  $I_{T_n}$  into the style extractor  $E_s$ , getting  $E_s(I_{T_n})$  which is abbreviated as *style*. The same is carried out for the content extractor. They are decoded by the decoder  $D$ .

The loss is expressed as follows:

$$L(Output_D, I_{T_n}) = \sum_{i=1}^n \cdot \sum_{j=1}^n |z_{ij} - x_{ij}| \quad (2)$$

The  $Output_D$  is compared to  $I_{T_n}$  in a pixel-to-pixel manner. By taking the mean absolute error between them as the loss to minimize, where  $z$  represents the ground truth,  $x$  represents the decoder output and  $n$  is the length and width of  $z, x$ , this trains both the encoding and decoding abilities of the encoder and decoder, respectively.

Second, we introduce the loss of the auto-encoding in  $V_{CS}$  and  $V_{CT}$ . The output of the decoder is similar:

$$Output_D = D(E_s(V_{S1_n}), E_c(I_{T_n})) \quad (3)$$

The loss is expressed in the same form as (2). This is because  $Output_D$  is compared to  $I_{T_n}$  in the same pixel-to-pixel manner. N.B.:

$$E_s(V_{S1_n}) = E_s(I_{T_n}) \quad (4)$$

And

$$E_c(V_{CT_n}) = E_c(I_{T_n}) \quad (5)$$

This is because  $V_{S1}$  and 'Set A Vectors', (within the red box), extracted by the content extractor, from images of the same content and Set B Vectors (within the green box), extracted by the style extractor, from images with the same style *should be the same* (Yu et al. [14]), comparing using the K-L divergence. It is defined as:

$$KL(P \parallel Q) = \sum_{i=1}^N P(x) \cdot \log\left(\frac{P_i(x)}{Q_i(x)}\right) \quad (6)$$

It calculates the change in entropy when we change probability distributions, allowing us to compare two probability distributions (in (6),  $P$  in respect of  $Q$ ). In which, the lower the K-L value, the better matched our distribution is with our approximation.

Back to our model, as the two content vectors come from images with the same contents, they should also share the same distribution and mean in this distribution. Similarly, the style extractor is also trained in such a way to ensure that only style features are extracted from the input images.

**Remark** Using this method of combining the content and style from two characters, it is expected that even if the two characters come from two different languages (e.g., Chinese and English), we can still effectively practice font style transfer as both extractors are only responsible for extracting one single feature of either style or content. While the content extractor recognizes the content of the Chinese character (e.g., ‘人’), the style extractor only recognizes the font of the English character (e.g., Times New Roman). Thus, by combining this two information, the decoder can generate a Chinese character with an English font, achieving a universal cross-language character-font generation.

**Model architectures** Our style extractor used is a *Convolutional Neural Network* (CNN). It extracts style information from a character through its subsequent image and content vector. The image of the character is fed into 5 ResNet blocks which consist of a skipped layer of 2D convolution

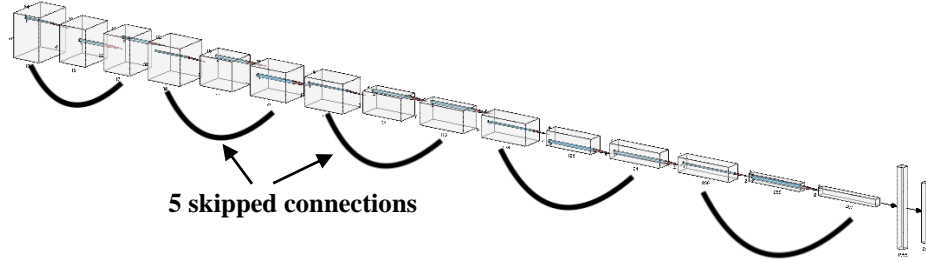


Figure 5: The model architecture of the style and content extractor

of a filter size  $1 \times 1$ , followed by a 2D convolution layer with a filter size of  $2 \times 2$ , ended with a  $2 \times 2$  max-pooling layer. After feeding through the ResNet blocks, their output is flattened and pass on to a fully-connected dense layer of 256 nodes, generating the style vector output (representing the data’s style). Throughout the model, *batch normalization* [16] and the *Leaky-ReLu* activation [17] are utilized.

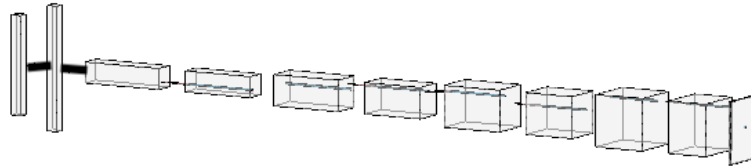


Figure 6: The model architecture of the decoder

The content extractor has the same structure as the style extractor. The only difference is that the output is passed on to a fully-connected dense layer of 512 nodes, generating the content vector output (representing the data’s content)

The decoder decodes the content vector and style vector, outputting a character with content represented by the content vector and style represented by the style vector.

The input vectors are concatenated before putting into a dense layer of 4096 nodes, which is then reshaped. The image of the character is fed into 5 ResNet blocks which consist of a skipped layer of 2D transposed convolution with a filter size  $3 \times 3$ , followed by a 2D transposed convolution layer of filter size of  $2 \times 2$ . Finally, the output has the same shape as the image input of the content and style extractors.

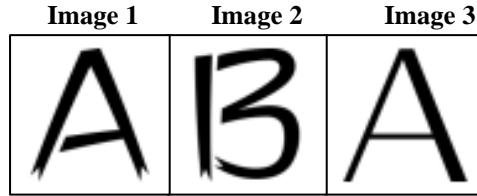


Figure 7: Demonstration of the output of the decoder

**Image 1:** Target Image

**Image 2:** Output sharing style of target image but different content

**Image 3:** Output sharing content of target image but different style

**Training** We used the *Stochastic gradient descent* (SGD) optimizer [18] to optimize our model across a computing cluster. Batches of 64 examples are used, initialized with learning rates of 0.0001 and a momentum of 0.9. The descent, where it is halved every 5 epochs in our network, is combined with a mean absolute loss a weighting of 1.0. It is crucial that the training of the two extractors’ content and style vectors, between the main and comparison character images, do not outweigh its ability to encode the image for the decoder to process.

While the pairwise comparison has a lower weighting than the auto-encoder itself, the loss weight of the style vectors’ pairwise loss should be higher than that of content vectors. This is because style vectors are more character-invariant and have more information associated with them than the content vectors. Thus, the weight of the style vectors’ pairwise loss is larger, which is 0.5, compared to the content vectors’ pairwise loss is 0.2.

**Table 2: Weights used of four loss functions**

|             | Loss function 1                       | Loss function 2  | Loss function 3                     | Loss function 4                   |
|-------------|---------------------------------------|--|-------------------------------------|-----------------------------------|
| Description | generation from main character images | generation from images with the same content but different style; and images with the same style but different content | K-L Divergence of content extractor | K-L Divergence of style extractor |
| Weight used | 0.04                                  | 0.1  | 0.01                                | 0.01                              |



## Results

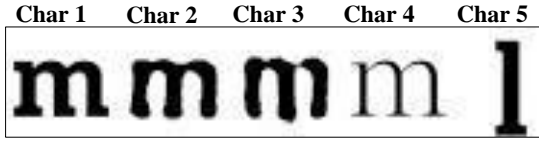


Figure 8

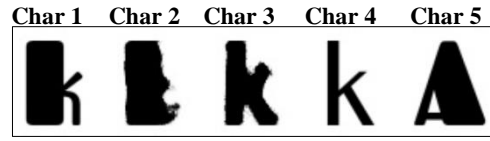


Figure 9

(Five characters in each figure, from left to right: *Char 1* - ground truth, *Char 2* - output with both ground truth's content and style, *Char 3* - formal output, *Char 4* - image to preserve content, *Char 5* - image to preserve style)

As seen from the two Figures, our network can extract thickness features of strokes from the characters fed into the style extractor. In Figure 8, *Char 5* 'l' (preserving the style) has thick strokes while *Char 4* 'm' (preserving the content) has thin strokes. This shows how our model intentionally disregarded the thickness style of *Char 4* but retained the thickness from *Char 5*. All are shown in the formal output *Char 3*, which also is a close resemblance to the ground truth *Char 1*.

In Figure 9, the model can thicken the *Char 3* 'k' yet falls short in comprehending the partial thinness presented in *Char 5* 'a' thus not implementing into *Char 3* 'k'.

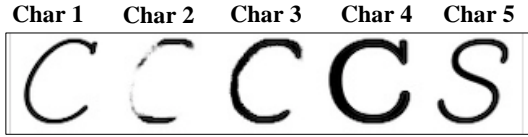


Figure 10

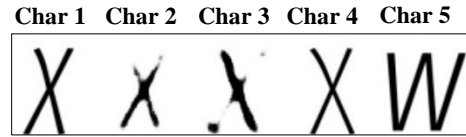


Figure 11

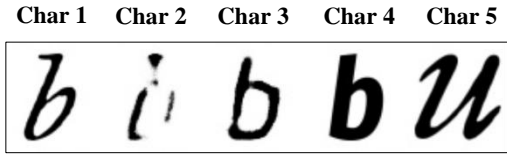


Figure 12

Second, our network is also able to extract orientation features of strokes from the character fed into the style extractor. As in Figure 10, *Char 5* 's' (preserving the style) is tilted to the right while the *Char 4* 'c' (preserving the content) is not. Take note of how our model disregards the orientation feature of *Char 4* but retain the tilting from *Char 5* as shown from the formal output *Char 3*, at the same time resembling the ground truth *Char 1*.

In Figure 11, our model can tilt the *Char 3* 'X' but shows broken strokes. In Figure 12, *Char 3* 'b' indeed tilts like 'u' but is unable to tilt to the extent of 'u' and is unlike the leftmost ground truth 'b'. The top left part of 'b' is not able to extend outwards as the top left part of 'u'.

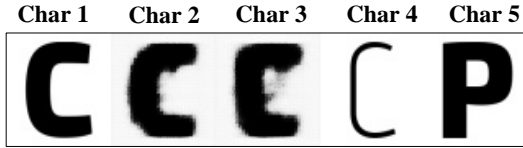


Figure 13

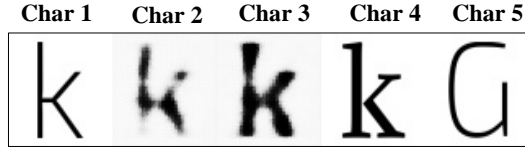


Figure 14

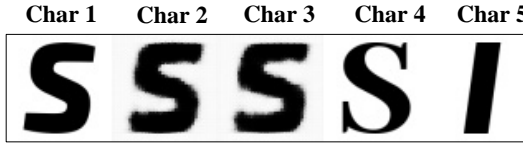


Figure 15

Third, our network is able to extract thickness features from the style character (*Char 5*) and apply the feature to the formal output (*Char 3*). In Figure 13, *Char 5* ‘P’ has thick strokes compared to *Char 4* ‘C’. We want this feature of thick strokes in *Char 3*. The model successfully retains this feature.

In Figure 14, the model is unsuccessful in extracting the thin strokes of *Char 5* ‘G’. The formal output *Char 3* has thicker strokes similar to that of *Char 4*. In Figure 15, the thick stroke in the *Char 5* ‘I’ is retained to the formal output *Char 3* ‘S’. This shows that the model is still able to extract thickness information from given character images.

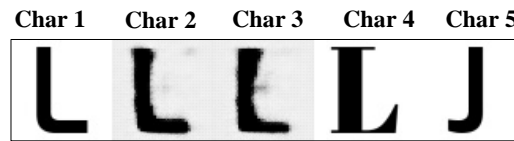


Figure 16

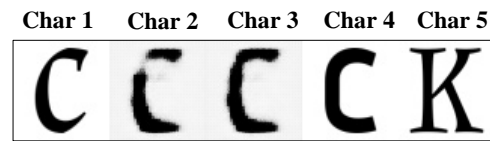


Figure 17

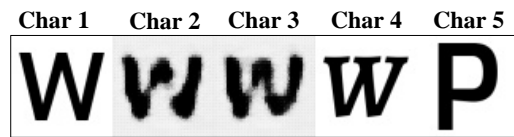


Figure 18

Forth, our model can distinguish between Serif and Sans-serif style from the given character images. In Figure 16, *Char 4* ‘L’ has a Serif feature while *Char 5* ‘J’ has a Sans-serif style. Our model successfully disregarded the Serif feature of *Char 4* and generated *Char 3* without the typical “extension” found in Serif fonts.

Yet, in Figure 17, the model is unable to generate the serif-feature of *Char 5* ‘K’ for *Char 3* ‘C’. The upper part of *Char 3* does not extend downwards to demonstrate a Serif style. Still, in Figure 18, the model again successfully extracted the style of *Char 5* ‘P’ and applied the style on *Char 3* ‘W’. *Char 3* does not show an “extension”.

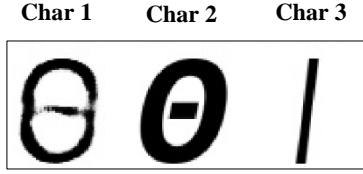


Figure 19: Greek example 1

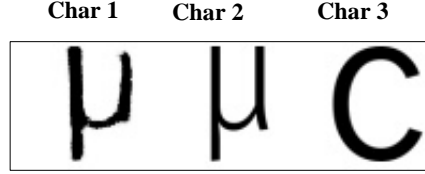


Figure 20: Greek example 2

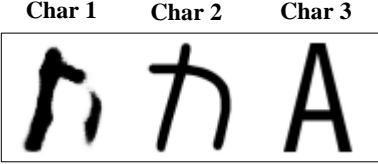


Figure 21: Japanese example

Fifth, referring to the name of our proposed network, *Universal Solution to Extension and Translation of Fonts Across Languages*, our network is well-equipped to transfer font style across different languages.

From Figure 19, it is clearly shown that multilingual character generation is possible. *Char 1* combines the style of *Char 2* and the content of *Char 3*. *Char 3* is an English ‘l’ while *Char 2* is the Greek alphabet of ‘ $\theta$ ’. Though presented with minor flaws, such as those in Figures 20 (Greek) and 21 (Japanese) where the foreign language’s style may alter the shape and blur the content, generally speaking, the generated characters all have a high resolution, with clear strokes and smooth edges comparable to the input character.

**Discussion** From the above five examples we have demonstrated our network’s ability in performing font style transfer. Yet, there are several cases where our model performs comparatively weaker at:

First, it is when the characters have an orientation that is not ‘standard’ (i.e., not at 0 degrees). On notable occasions, the orientations are not transferred although the thickness or relative location of strokes in a character is typically well-reflected. We believe the reason for this is because most fonts inputted during the training process of our network have a standard orientation of 0 degrees.

Second, it is also observed that some of the strokes in more complicated characters diffuse out into the background. Consider *Char 1* in Figure 21. We suspect it is due to the complexity of the formation of the character, where different strokes in different characters bear a greater difference in their relative location and thickness, compared to the different strokes in characters with simpler formation. This might lead to the inability of our network in fitting those characters for generic style generation. Accompanied by the observation of the blurriness of the more complicated shapes in a character, it can also be seen that characters having thinner strokes tend to lead to a blurred output. The reason might be the breaking up of thinner strokes during the generalization of the network.

Third, examining the results from a more general perspective, an important technique in calligraphy, the presentation of stroke tips (筆鋒) is missing. In Chinese calligraphy, each stroke should have a clear starting and ending point (typically, the starting point is broader, and the ending point is

narrower). Yet, our network produces no such pattern in its outputs. This might be due to the low resolution of the input being unable to present intricate features of each stroke, which makes the strokes less prominent for the network to process.

**Comparison with the GAN model** To evaluate the ability of our network, 25 Chinese characters were selected as an experiment for the *Multi-content Generative Adversarial Network* (McGan) with style vectors. The following Figure shows the generated results:

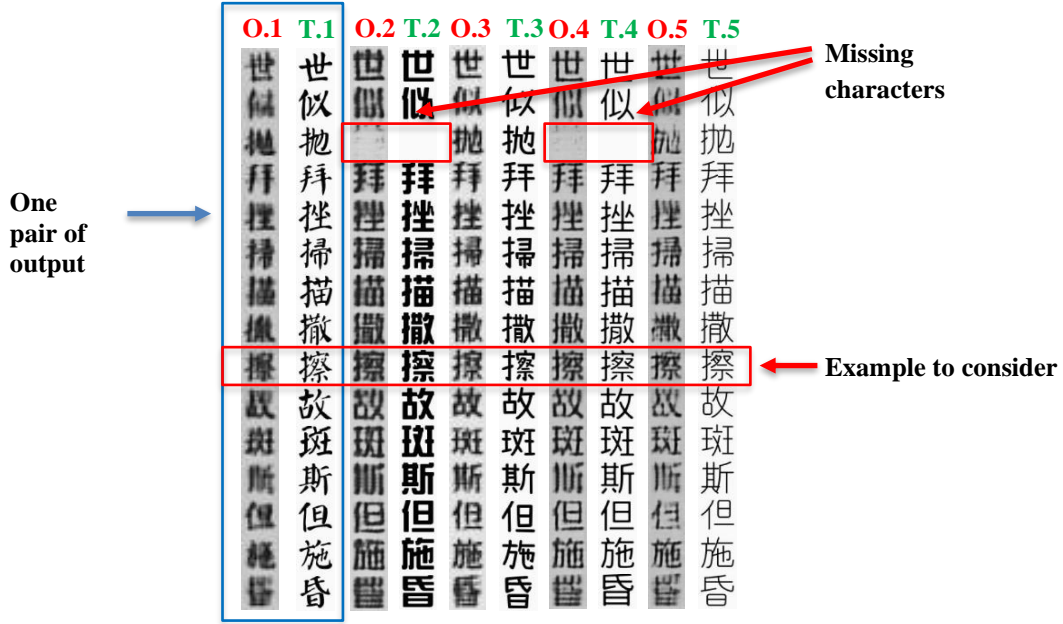
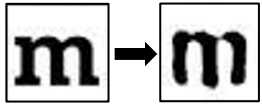
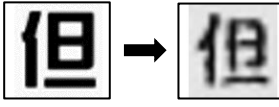
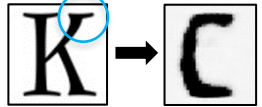
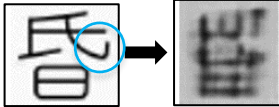

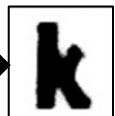



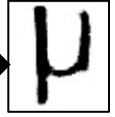


Figure 22: Pairs of output results from the GAN model (*O*: *Output*; *T*: *Target*)

From the Figure, there are several prominent issues with the McGan model. First, missing characters exist. This is because particular characters do not exist in the fonts that we collected, yet the McGan is incapable of generalizing the font style into the content. Second, in the example of “擦”, the outputs of O.2 and O.3 are blurred in comparison with the Target. A more detailed comparison is as follows:

**Table 3: comparison between our proposed model and the McGan model**

|                         | Our proposed model  | McGan model  | Comment  |
|-------------------------|---|--|--|
| Image sharpness         |  |  | Our proposed model outperforms the McGan model in output sharpness |
| Serif feature detection |  |  | Both models are unsuccessful in capturing the Serif font feature   |

|                                  | Our proposed model  | McGan model  | Comment  |
|----------------------------------|---|--|--|
| Thickness extraction             |  →  |  →  | Both captured thickness features. Yet, our model performs better in the usability of the output. |
| Font translation across language |  →  | Unable to do so  | Our model is more universal without language restrictions  |

From the above comparisons, despite that neither model is perfect, in certain aspects such as thickness extraction, the sharpness of image and universality, our proposed model performed better than the traditional McGan model.

## 5. Future Directions

First, concerning the problem of thickness, we can extract the content features of the output once again using the content extractor. This trains the generated character to retain its content information, avoiding the merging of strokes as merging of strokes changes its content.

Second, concerning the orientation problem, to make the generated results have the consistent orientation as the ground truth, we may try adding another orientation net [5] that evaluates upon the orientation of the result. This aims to transform the characters in the image to acquire the same orientation as the ground truth.

Third, to eliminate the diffusion of strokes to the background, we may add another attribute of the character array for loss comparison, which we call *total variation loss*. This forces the difference between the stroke and the background to be significant as to prevent diffusing out or being broken. The loss of two consecutive pixels can be represented as follows:

$$L(n, n + 1) = \log (1 - |P(n) - P(n + 1)|) \quad (7)$$

where  $L(n, n + 1)$  is the total variation loss of two consecutive pixels  $n$  and  $n + 1$ , where  $P(n)$  denotes the pixel value of  $n$ .

In response to the inability of our current network to present stroke tips, we may add another *convolution neural network* that distinguishes each stroke from each other, learning the basic stroke directions of each stroke, then applying the stroke tips on the output. The stroke tip loss to be minimized can be represented as

$$L_{stroke\ tip} \cdot (G(x)) = |S(z) - S(G(x))| \quad (8)$$

where  $L_{stroke\ tip} \cdot (G(x))$  is the stroke tipi loss of the product,  $G(x)$  is the output of the generator network  $G$  and input  $x$ , where  $S(z)$  is the stroke representation vector of ground truth  $z$  and  $S(G(x))$  the stroke representation of product  $G(x)$ .

## 6. Conclusion

In conclusion, we have demonstrated the potential and ability of a deep auto-encoder with multi-content and manual style loss-minimizing to *universally transfer font styles across languages* given only the style of input characters. Despite room for future improvements, this research has served as a rigorous verification of the possibility of generalizing our proposed method of style transfer.

**Broader Impact** Our model allows the combination of two characters with the content of the former and the style of the latter. This implies that *ab initio*, the number of characters our model can generate is unlimited, as long as you provide the two characters to extract their content and style from. This makes our network *universal*. It can be simplified to perform the task of generating characters in a particular written font style by any person, by simply inputting the individual's written character (into the style extractor) and a character of a standard font (into the content extractor). This enables us to immediately create a whole bank of characters with the target's font style with one-shot capture.

## References

- [1] Xu, S., Lau, F. C., Cheung, W. K., & Pan, Y. (2005). Automatic generation of artistic Chinese calligraphy. *IEEE Intelligent Systems*, 20(3), 32-39.
- [2] Xu, S., Jiang, H., Jin, T., Lau, F. C., & Pan, Y. (2009). Automatic generation of Chinese calligraphic writings with style imitation. *IEEE Intelligent Systems*, (2), 44-53.
- [3] Yijun, L., Ming-Yu, L., Xueting, L., Ming-Hsuan, Y., Jan, K. (2018). A Closed-form Solution to Photorealistic Image Stylization. *arXiv: 1802.06474 [cs.CV]*
- [4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).
- [5] Azadi, S., Fisher, M., Kim, V. G., Wang, Z., Shechtman, E., & Darrell, T. (2018). Multi-content gan for few-shot font style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7564-7573).
- [6] Kohtaro Abe, Seiichi Uchida (2019) GlyphGAN: Style-Consistent Font Generation Based on Generative Adversarial Networks. *arXiv:1905.12502*
- [7] Narusawa, A., Shimoda, W., & Yanai, K., Font Style Transfer Using Neural Style Transfer and Unsupervised Cross-domain Transfer
- [8] Grödl, M., & Resl, M., AIfont: AI-generated Typeface - Generating new Typefaces with Deep Learning

- [9] Abe, Kotaro & Iwana, Brian & Holmér, Viktor & Uchida, Seiichi. (2017). Font Creation Using Class Discriminative Deep Convolutional Generative Adversarial Networks. 10.1109/ACPR.2017.99.
- [10] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
- [11] Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576.
- [12] Sun, D., Ren, T., Li, C., Su, H., & Zhu, J. (2017). Learning to write stylized chinese characters by reading a handful of examples. arXiv preprint arXiv:1712.06424.
- [13] Fernandez, G. (2019). Font-gen: Deep Models for Inferring Alternate Language Sets from Fonts. URL: [26259019.pdf \(stanford.edu\)](https://arxiv.org/pdf/26259019.pdf)
- [14] Sun, H., Luo, Y., Lu, Z. (2018). Unsupervised Typography Transfer. arXiv:1802.02595 [cs.CV]
- [15] Lyu, P., Bai, X., Yao, C., Zhu, Z., Huang, T., & Liu, W. (2017, November). Auto-encoder guided gan for Chinese calligraphy synthesis. In 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR) (Vol. 1, pp. 1095-1100). IEEE.
- [16] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.
- [17] Xu, B., Wang, N., Chen, T., & Li, M. Empirical evaluation of rectified activations in convolutional network. arXiv 2015. arXiv preprint arXiv:1505.00853.
- [18] Saad, D. (1998). Online algorithms and stochastic approximations. Online Learning, 5, 6-3.