STAG: Smart Tools and Applications in Graphics (2020)
S. Biasotti, R. Pintus and S. Berretti (Editors)

Capturing Fonts in the Wild

Martin Kolář, Michal Hradiš and Pavel Zemčík

Brno University of Technology

Source image with unknown font *New vectorized text using imitated font*

Muļķa hipiji mēģina brīvi nogaršot celofāna žņaudzējčūsku
Muļķa hipiji mēģina brīvi nogaršot celofāna žņaudzējčūsku

Google Docs rendering with missing diacritics (above), and the same Latvian pangram using automatically generated diacritics

Figure 1: *Font capture enables editing text in photographs by capturing the original font, and extrapolating to unseen glyphs, as well as extension of existing TrueType fonts with vectorized diacritics*

Abstract

Editing text in photographs requires the ability to find the same font, which is impossible in many settings, such as historical or manually painted text. We present a method of extracting the font from a single photographed word, without relying on the retrieval of similar fonts. A deep net extracts style information and constructs the font for all characters, enabling novel applications in image editing, font creation, and the addition of language-specific characters with diacritics to existing fonts. A qualitative user study shows that this method improves convincing font capture by over 500% over prior work.

CCS Concepts

• ***Computing methodologies*** → ***Image processing;***

1. Introduction

The use of varied fonts is ubiquitous, from corporation logos, to newspapers and banknotes. Type designers define vector graphics for each character manually, and extrapolating the entire typeset given a small sample of text takes several days. This work presents a deep learning approach to capturing fonts, as well as two practical applications: editing fonts in photographs and creating diacritics for known fonts.

Digital editing and rendering allows working with lighting, textures, shapes, position, color, and numerous other factors. However, editing text in digital media requires a definition of the desired font, which may be difficult to acquire, unavailable, or non-existent. Photographs may contain typefaces which are not based on digitally defined fonts, but instead unique artistic realizations. This work presents an original approach to extrapolate other glyphs given a small sample of text in any font, enabling re-writing of text.

Another application demonstrated in this work is automatic extension of any existing typeface by characters with diacritics. Worldwide, 750 million people are native speakers of a language written in a Latin-derived alphabet with diacritics such as accents, subscripts, and superscripts [wik18]. However, out of an estimated

100 thousand digital fonts widely available, only a few hundred include these non-English characters.

Font extraction on characters of the Latin alphabet has been attempted before, either with limited applications to classical fonts [CK14], or with blurry or noisy results [Bal16, USB16], and always by using characters as input. Thanks to a novel architecture, this work creates sharp fonts extracted directly from a line of text, suitable for use in photo editing as well as vectorization. This work makes it possible to take an existing TrueType font, render new characters, convert them to vector graphics, and incorporate them in the original, thus effectively closing the loop.

2. Previous Work

Since the invention of the printing press in the fifteenth century, movable inverted glyphs of the Latin alphabet have been used to print text. Until the advent of mass digital printing, the physical type of every point size had to be designed and kept for each desired font, Figure 2. Personal computers and document editors ushered an explosion of existing fonts as well as their variety, with each glyph defined manually as a vector graphic. As in the era of metal/wood type, only those with access to these vector definitions are able to create text in a given font.

Figure 2: *Wooden type. The umlaut affects the base glyph A*

Although fonts are widely shared on the internet, and font search engines are freely available, few fonts can be acquired to perfectly match a desired input. Finding a font given an image is a challenging task, undertaken by domain experts or automated processes. Identification methods range from pixel differences on detected aligned characters [squ17] to matching manually entered detailed features [ide17] based on standard font classification techniques [CGL13], or automatically extracted attributes [OLAH14]. If these methods fail, fonts can be identified by a community of experts, such as Fontid.co. However, exotic fonts may be unknown to experts, unavailable to identification systems, or non-digitized. For example, Figure 3 shows a query text, along with nearest retrieved fonts by existing methods. This demonstrates that pixel difference is not a sufficient metric in font style matching.

Limits of finding existing fonts sparked an interest in extrapolating the entire style of a font from a single example. Font extrapolation with warp mappings dates to the nineties [TF97], inspired by the effect on shape of charge on ink particles. A manifold over fonts has allowed smooth traversal of the font space [CK14], and was

(a) *Query text from image - hand-drawn*

(b) *Nearest match by pixel difference -* ***JollyGood Sans Condensed***

(c) *Nearest match by property matching -* ***Keynote*** *(caron unavailable)*

(d) *Nearest match by expert community -* ***Krinkes***

Figure 3: *Comparison of font retrieval methods*

applied to classical typefaces to interpolate fonts [YW20]. Extrapolation of numerals on the MNIST and SVHN datasets was made possible by deep generative models, creating a latent space which allows traversal across glyphs [KMRW14].

More recently, a fully connected deep net has been used to create an embedding of 50 thousand fonts [ana17]. A feed-forward neural network has been used to generate the entire font from four characters [Bal16], see Figure 4 for results taken from the paper. In addition to limited quality, this technique suffers from requiring specific characters, which may not occur in the sample text. Variational Autoencoders have been used to generate fonts from a single example glyph [USB16], but with a small dataset of 1'839 fonts and a fully connected network, the results are still blurry. The 50k fonts dataset [ana17] has been used to train a VAE and a GAN [gan16], using the principles outlined in [RMC15]. Fonts are extrapolated from varying characters with a Multi-Content GAN [AFK*18], in color in Figure 5. The same approach has been adapted to Chinese characters by significantly extending the dataset and using One-stage Few-shot learning [YYZ*19]. However, existing methods are impractical in applications for graphics because they require segmented characters, rather than analyzing text directly in the image. Most crucially, results of all existing methods are blurry or noisy for all but the most standard fonts.

The history of Chinese printing is quite separate, with the first use of movable type dating back to the eleventh century. Since there are thousands of glyphs, rather than a few dozen, designing and managing the typeface has been a comparatively mammoth task. To make a font compatible with the Chinese standardized character set, designers need to design more than 26 000 characters, a challenging task that can take years to complete. However, since Chinese characters are composed of a core set of 270 radicals, style transfer techniques produce successful results [rew17, ZCZ17], as shown in Fig-

(a) *Input* **(b)** *Output* **(c)** *Reference*

Figure 4: *A forward neural network extrapolating "R" from "BASQ" [Bal16] on three random fonts*

Figure 5: *Multi-Content GAN results [AFK*18]. Ground truth characters are on the top and synthesized results on the bottom of each row*

ure 6. Such a style transfer network [GEB16] extrapolates content information from a given glyph and applies a style to it, rather than rendering the entire alphabet, or using one-hot encoding to render a single glyph. These findings have since been extended to two other logographic writing systems, Korean and Japanese kanji using Conditional Adversarial Networks [IZZE16, zi217]. However, these methods are not applicable to alphabetic languages, because network module encoding a character will not see sufficient variety to train effectively, since the graphemes lack repeating radicals.

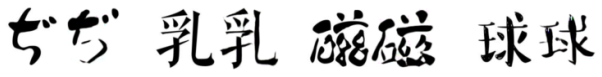


Figure 6: *Pairs of characters, where ground truth characters are on the left and synthesized results on the right [zi217]*

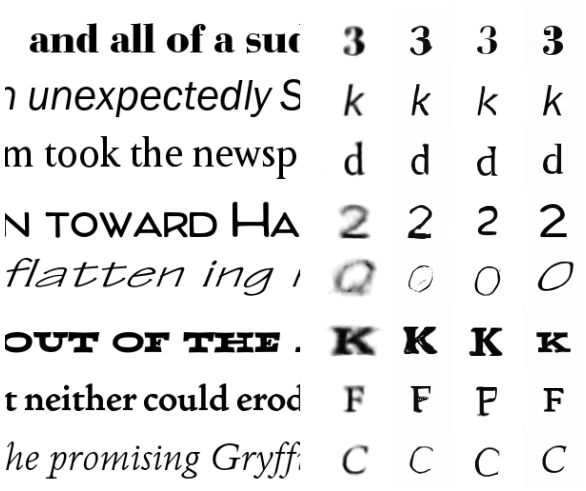
Handwriting style can be successfully imitated using a recurrent neural network [Gra13]. Such an approach may extrapolate style for calligraphic cursive fonts, or stroke-based geometric fonts.

3. Font Capture

This section describes the presented method. Our contribution lies in a improved font dataset collected by Active Learning, and a font capture network which reads text, rather than single characters.

3.1. Font Dataset

Fonts used in this paper have been acquired online, with 222 462 used out of 272 849 unique fonts, including 7 089 fonts with selected diacritical marks (an acute accent ´, circle ˚, or caron ˇ on eight characters). A font family is typically a group of related fonts which vary only in weight, orientation, width, etc., so it is desirable to include fonts with similar variations. Downloaded fonts have been filtered with a deep net trained to detect foreign non-Latin fonts, dingbats, emojis, and ornamental typefaces, which may produce unexpected characters for standard glyphs. See Figure 7 for a random sample from the cleaned dataset.



(a) *Input Style* **(b)** **(c)** **(d)** **(e)**

Figure 7: *Effects of various parts of the architecture. (b) is the forward network without a discriminator, (c) is the GAN with a discriminator that does not see the input style, (d) is the network with GAN looking at style, and (e) is the larger, final network. Note that the pre-gan results may be convincing, but blurry.*

3.2. Font Capture Network

Previous work attempts to extrapolate font style from a single capital character [USB16], from four selected capital characters [Bal16], or from all but one capital character [TF97]. However, this is impractical when font definitions are not available. Therefore, this work is the first to extrapolate a font style from an arbitrary short line of text, which can be easily extracted from photographs, as shown in Figure 1. For experiments, we chose this style source image to have resolution 64×500 which captures from two up to five words (see Figure 3a and 8).

The previous work using variational autoencoders [USB16, KMRW14] produces low quality and blurry outputs. On the other

hand various versions of conditional Generative Adversarial Networks [MO14] have demonstrated high quality results in related tasks [zi217, LBY*17].

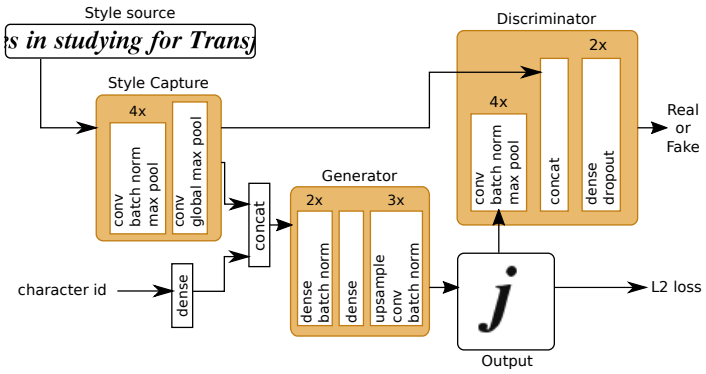


Figure 8: *Network shape*

Our Font Capture Network (Figure 8) is an extension of conditional Generative Adversarial Networks [MO14] (cGAN). The Generator takes as an input real vectors representing a font style and a requested glyph identity and renders a single 64×64 glyph image. The glyph vectors are learned by an embedding layer (look-up table) and the style vector is extracted by a Style Capture Network from a font style source image. The Discriminator outputs a binary real/fake decision for a glyph image, while making use of the extracted font style vector.

The network is trained using two loss functions: L2 loss with respect to ground truth images and an adversarial loss on the Discriminator. The pixel-wise L2 loss, which compares a generated glyph to its ground truth, ensures that a correct glyph is generated and that it has roughly correct shape. However, this results in blurry outputs as the pixel loss forces the network to produce averaged pixel values in areas which can not be reliably predicted. This blurriness is suppressed by an adversarial loss which forces the rendered glyphs to be indistinguishable from real images. We observed that making the Discriminator conditional on the style vector results in improved rendering of small details. These include defining font features such as serifs, slants, and spurs. The effect of the adversarial loss and the style-conditional adversarial loss is clearly demonstrated by Figure 7.

The Style Capture network aggregates information from a style source by four convolution layers each followed by max pooling (reducing resolution $16\times$) and a final global max-pooling operation which results in a single 512-d vector. All convolutional layers have 3×3 kernels.

The network is trained using Adam [KB14] with learning rate 0.0002 on mini-batches of 64 glyphs. The training iterates between one update of the Discriminator with cross-entropy loss and the Generator and the Style Capture networks with the pixel L2 loss and the adversarial loss from the Discriminator with real labels. For further implementation details and results, see the code made available online[†]. The networks were trained for 150k iterations and we observed reasonable outputs after 10k iterations. Thanks to the large fontbase, overfitting does not occur and we did not observe any differences between training and held-out validation fonts.

[†] included in material for review

Method	success rate	adjusted success rate
VAE [KMRW14]	4.7%	9.4%
ADV-VAE [USB16]	1.6%	3.1%
Ours	25.6%	51%

Table 1: *User Study Results. If the method produces indistinguishable outputs with respect to the ground truth, the user performs a random binary choice. This corresponds to a 100% method success rate for an ideal output, versus 50% measured in the experiment. The adjusted success rate is doubled accordingly.*

4. Evaluation

The benefits of Font Capture described above are demonstrated through a comparative user study which shows five-fold improvement over state-of-the-art. Furthermore, two applications are shown, the first autonomously creating vectorized diacritics, and the second for editing text in photographs.

4.1. User Study

A user study with 17 participants compared generated characters from state-of-the-art methods: VAE [KMRW14], ADV-VAE [USB16], and this work. The study was performed with three triplets of characters, as shown in Figure 9. Each participant received 72 rows of triplets, printed on four sheets, and was asked to identify the different triplet. If the user fails to identify the generated triplet, the methods output can be considered indistinguishable from the original font. Correct and incorrect user classifications are summed for each method, and results are presented in Table 1. The proposed method recreates fonts convincingly in 51% of cases, compared to 3% and 9% for the previous methods. This is a 566% increase in precision. According to the randomization permutation test, these results are highly significant ($p<0.0001$). Furthermore, our tests show that VAE outperforms ADV-VAE with p-value 0.059.

4.2. Generating diacritics

Out of the 26 base letters of the alphabet, 18 are base glyphs for an additional 71 diacritical letters in European languages alone [cod17], all of which are to be rendered in lower and upper case for a total of 178 characters. In addition, numerals and punctuation constitute another 40 highly used characters. Furthermore, ligatures (such as æ) and additional diacritics (such as in Vietnamese) comprise dozens of additional glyphs that a font may be required to contain.

However, rendering accents is more complex than simply placement. Diacritics may vary in appearance on different base glyphs,

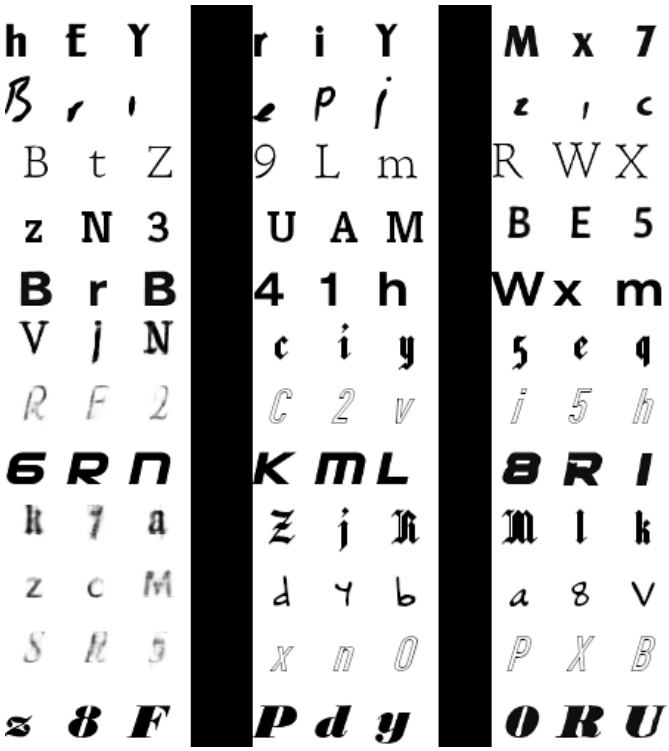


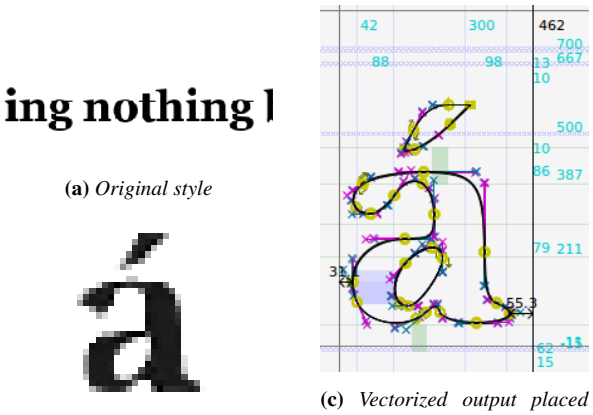
Figure 9: *User Study Setup. Each row contains three triplets, two of which are ground truth, and one is generated by one of the three methods. Their order is randomized, except for the middle triplet, which is always the ground truth.*

Figure 10: *A font which changes the standard writing of i when rendering í, and uses varying form and placement of the caron on t, d, and c*

affect the shape of the base glyph, and be placed in different locations (Figure 10 and Figure 2). Furthermore, certain diacritics may be combined, such as ā in Lithuanian. Document editors revert to similar standard fonts when diacritics are unavailable, leading to visible undesirable renderings, as shown in Figure 11.

Figure 11: *Google Docs Rendering text with diacritics*

Vectorization is performed with Potrace [Sel03], with default settings, on the thresholded outputs. Finally, the glyphs are placed into the existing font definition (.ttf), where the ascender height, x-line, descender height, and others are already defined. Kerning is copied from base glyph definition. Therefore, the entire process can be automated.



(a) *Original style*

(b) *Generated "a" with diacritic*

(c) *Vectorized output placed into the font definition of the base glyph "a"*

Figure 12: *Synthesized diacritics are placed into the font definition, thereby acquiring kerning and positioning parameters from the base glyph*

Figures 19 and 20 demonstrate the versatility of the method on sample outputs. It can be seen that the method works robustly on unseen validation fonts, and can reliably be used to generate characters with and without diacritics.

4.3. Editing text in photographs

Figure 13 to Figure 18 show the process of replacing text in an image. The original image (Fig. 13) is rectified, and the text is cropped (Fig. 14), and thresholded (Fig. 15). This is the input seen by the deep net. The network renders each desired output character (Fig. 16), which undergo manual kerning (Fig. 17) and placement into the original using standard editing tools (Fig. 18).

Methods exist that can allow text detection [VAMM*08] and rectification [TN11] to automated for any task-specific application.

Figure 13: *Original image*

Figure 18: *Generated text in image*

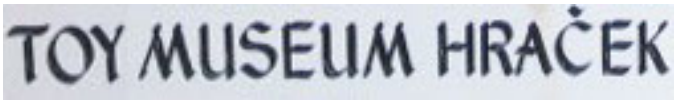


Figure 14: *Cropped image*

Figure 15: *Thresholded image*

Figure 16: *Generated characters*

Figure 17: *Text after kerning*

5. Conclusion

Fonts are present in all forms of visual media, but working with them remains possible only for those with access to the type definitions. This work widens the possibilities for tools such as Photoshop and Google image translate, where recreating text in a given font is key.

Furthermore, automatically expanding the diacritical sets for existing fonts brings all fonts to a wider audience of hundreds of millions of users whose language includes diacritics. Office suites such as Microsoft Office & Google Docs can benefit from incorporating such tools in the future.

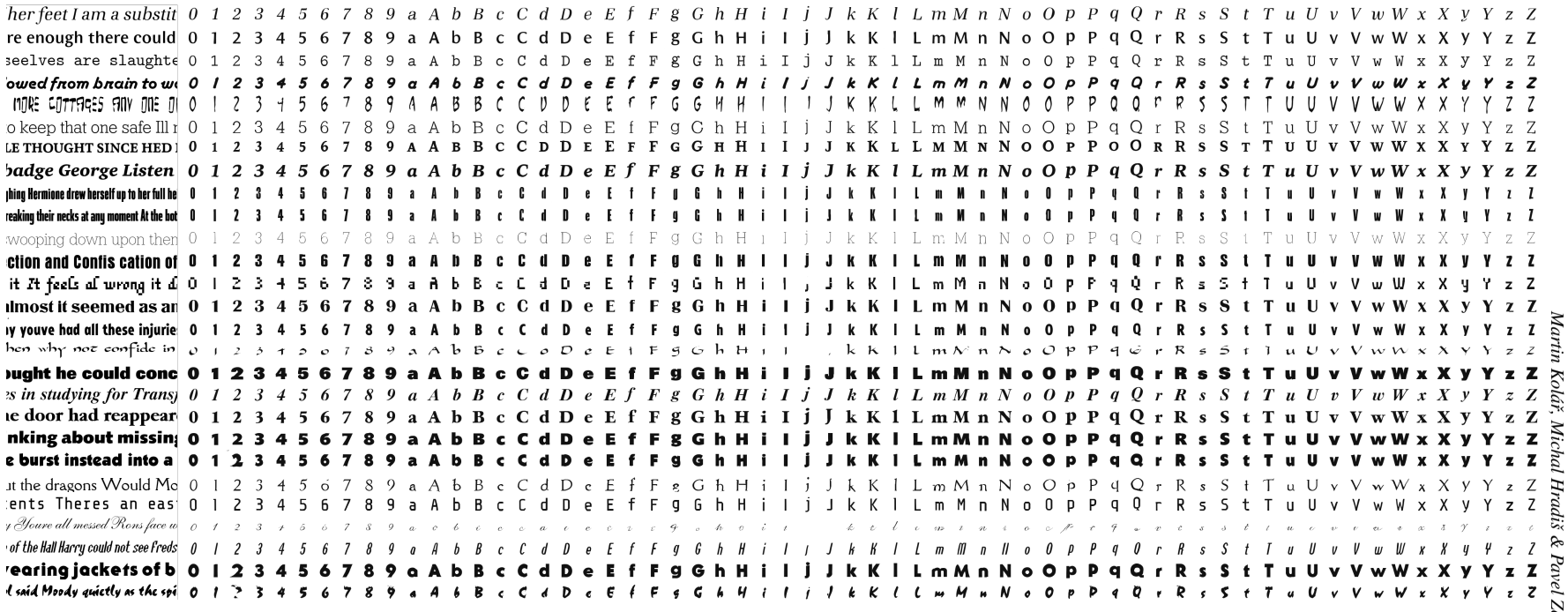
Font artists and type designers can also take advantage of Font Capture, designing a subset of the font or using existing images to generate new fonts. This functionality could be improved by outputting vector graphics, although this can already be done with Potrace [Sel03] or splines.

A limitation of this work is the lack of kerning information. Currently, kerning is being done manually, so automated letter spacing is only possible for monospaced fonts, but this can be incorporated as an additional specialized task.

Using this approach, fonts may be extended to other alphabets and non-alphabetic languages for the benefit of billions of people whose native languages are not written in Latin alphabets. Unicode defines 136900 characters [The11], all of which can be generated in any font using this approach.

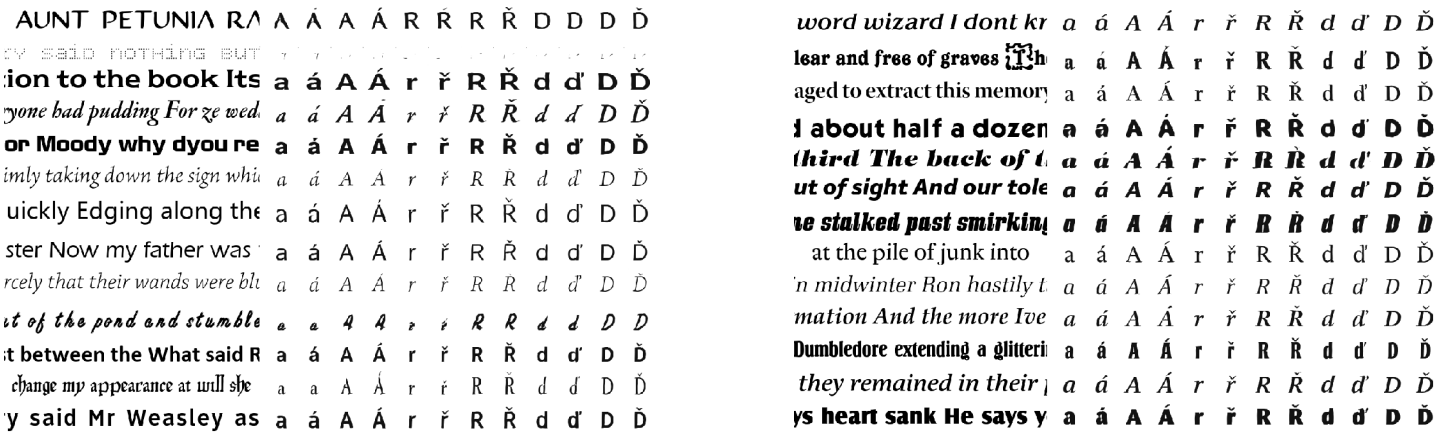
6. Acknowledgement

This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science – LQ1602.



(a) *Input sentences* **(b)** *Output glyphs*

Figure 19: *Sample Validation Fonts for Extrapolation. Note the missing j's and*



(a) *Input sentences* **(b)** *Output glyphs* **(c)** *Input sentences* **(d)** *Output glyphs*

Figure 20: *Sample Validation Fonts for Diacritics*

References

[AFK*18] AZADI S., FISHER M., KIM V. G., WANG Z., SHECHTMAN E., DARRELL T.: Multi-content gan for few-shot font style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 7564–7573. 2, 3

[ana17] Analyzing 50k fonts using deep neural networks, 2017. URL: https://erikbern.com/2016/01/21/analyzing-50k-fonts-using-deep-neural-networks.html. 2

[Bal16] BALUJA S.: Learning typographic style. *arXiv preprint arXiv:1603.04000* (2016). 2, 3

[CGL13] CHILDERS T., GRISCTI J., LEBEN L.: 25 systems for classifying typography: A study in naming frequency. *PJIM Parsons Journal for Information Mapping 5*, 1 (2013). 2

[CK14] CAMPBELL N. D., KAUTZ J.: Learning a manifold of fonts. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 91. 2

[cod17] Context of diacritics, 2017. URL: http://www.urtd.net/x/cod. 4

[gan16] A fontastic voyage: Generative fonts with adversarial networks, 2016. URL: https://multithreaded.stitchfix.com/blog/2016/02/02/a-fontastic-voyage/. 2

[GEB16] GATYS L. A., ECKER A. S., BETHGE M.: Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2414–2423. 3

[Gra13] GRAVES A.: Generating sequences with recurrent neural networks. *CoRR abs/1308.0850* (2013). URL: http://arxiv.org/abs/1308.0850, arXiv:1308.0850. 3

[ide17] Identifont, 2017. URL: http://www.identifont.com. 2

[IZZE16] ISOLA P., ZHU J.-Y., ZHOU T., EFROS A. A.: Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004* (2016). 3

[KB14] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014). URL: http://arxiv.org/abs/1412.6980, arXiv:1412.6980. 4

[KMRW14] KINGMA D. P., MOHAMED S., REZENDE D. J., WELLING M.: Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems* (2014), pp. 3581–3589. 2, 3, 4

[LBY*17] LYU P., BAI X., YAO C., ZHU Z., HUANG T., LIU W.: Auto-encoder guided gan for chinese calligraphy synthesis. *arXiv preprint arXiv:1706.08789* (2017). 4

[MO14] MIRZA M., OSINDERO S.: Conditional generative adversarial nets. *CoRR abs/1411.1784* (2014). 4

[OLAH14] O'DONOVAN P., LĪBEKS J., AGARWALA A., HERTZMANN A.: Exploratory font selection using crowdsourced attributes. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 92. 2

[rew17] Rewrite: Neural style transfer for chinese fonts, 2017. URL: https://github.com/kaonashi-tyc/Rewrite. 2

[RMC15] RADFORD A., METZ L., CHINTALA S.: Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015). 2

[Sel03] SELINGER P.: Potrace: a polygon-based tracing algorithm. *Potrace (online), http://potrace. sourceforge. net/potrace. pdf (2009-07-01)* (2003). 5, 6

[squ17] Font identifier, 2017. URL: https://www.fontsquirrel.com/matcherator. 2

[TF97] TENENBAUM J. B., FREEMAN W. T.: Separating style and content. In *Advances in neural information processing systems* (1997), pp. 662–668. 2, 3

[The11] THE UNICODE CONSORTIUM: *The Unicode Standard*. Tech. Rep. Version 10.0.0, Unicode Consortium, 2011. URL: http://www.unicode.org/versions/Unicode10.0.0/. 6

[TN11] TIAN Y., NARASIMHAN S. G.: Rectification and 3d reconstruction of curved document images. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on* (2011), IEEE, pp. 377–384. 5

[USB16] UPCHURCH P., SNAVELY N., BALA K.: From a to z: supervised transfer of style and content using deep neural network generators. *arXiv preprint arXiv:1603.02003* (2016). 2, 3, 4

[VAMM*08] VON AHN L., MAURER B., MCMILLEN C., ABRAHAM D., BLUM M.: recaptcha: Human-based character recognition via web security measures. *Science 321*, 5895 (2008), 1465–1468. 5

[wik18] List of languages by number of native speakers, 2018. URL: https://en.wikipedia.org/wiki/List_of_languages_by_number_of_native_speakers. 1

[YW20] YIZHI WANG* YUE GAO* Z. L.: Attribute2font: Creating fonts you want from attributes. *ACM Trans. Graph.* (2020). 2

[YYZ*19] YUE G., YUAN G., ZHOUHUI L., YINGMIN T., JIANGUO X.: Artistic glyph image synthesis via one-stage few-shot learning. *ACM Trans. Graph. 38*, 6 (2019). URL: http://doi.acm.org/10.1145/3355089.3356574. 2

[ZCZ17] ZHANG Y., CAI W., ZHANG Y.: Separating style and content for generalized style transfer. *arXiv preprint arXiv:1711.06454* (2017). 2

[zi217] zi2zi: Master chinese calligraphy with conditional adversarial networks, 2017. URL: https://github.com/kaonashi-tyc/zi2zi. 3, 4