Few-Shot Image Synthesis using Noise-Based Deep Conditional Generative Adversarial Nets

Finlyson Mwadambo Msiska*, Ammar Ul Hassan*, Jaeyoung Choi**, Jaewon Yoo***

Abstract

In recent years research on automatic font generation with machine learning mainly focus on using transformation-based methods, in comparison, generative model-based methods of font generation have received less attention. Transformation-based methods learn a mapping of the transformations from an existing input to a target. This makes them ambiguous because in some cases a single input reference may correspond to multiple possible outputs. In this work, we focus on font generation using the generative model-based methods which learn the buildup of the characters from noise-to-image. We propose a novel way to train a conditional generative deep neural model so that we can achieve font style control on the generated font images. Our research demonstrates how to generate new font images conditioned on both character class labels and character style labels when using the generative model-based methods. We achieve this by introducing a modified generator network which is given inputs noise, character class, and style, which help us to calculate losses separately for the character class labels and character style labels. We show that adding the character style vector on top of the character class vector separately gives the model rich information about the font and enables us to explicitly specify not only the character class but also the character style that we want the model to generate.

Keywords : font| font styles| computer vision| image generation| generative adversarial networks

I. INTRODUCTION

Automated font generation with machine learning is a lucrative research field in the font design world recently. This is mostly because font design is a resource-consuming task and designers can spend considerable time creating new fonts. Modern font generation solutions use machine learning because of their end-to-end nature with less human involvement which saves a lot of time and reduces labor needed to design new fonts.

This ranges from designing regular, handwritten, or artistic styled fonts using architecture-variant models which can help generate quality fonts with these distinctive features. Earlier work in font generation introduced the use of generative adversarial networks (GAN) [1]. GANs are a popular machine learning framework used for data generation tasks.

Generally, machine learning font generation can be loosely categorized into two methods, those requiring an existing image as reference called transformation methods and those that just need random noise vectors and label vectors information called just generative methods. This research work studies the generative methods and proposes a model that generates fonts guided with the font character label and style label vectors. Since fonts in deep learning are represented as images in which the collective pixels in the image graphically signify a particular character and its font style.

In this paper, we introduce a combination of three vectors, a noise vector, a character embedding

*This work was partially supported by Institute of Information & communications Technology Planning and Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2016-0-00166, and partially supported by the Soongsil University Research Fund of 2019.

* Student Member, School of Computer Science and Engineering, Soongsil University, Graduate Student
** Member, School of Computer Science and Engineering, Soongsil University, Professor
***Member, Department of Small Business and Entrepreneurship, Soongsil University, Associate Professor

Manuscript : 2020. 10. 12
Revised : 2021. 01. 24
Confirmation of Publication: 2021. 02. 02
Corresponding Author : Jaeyoung Choi,
e-mail : choi@ssu.ac.kr

vector, and a style embedding vector to train a deep generative model to create new font images. The noise vector is used to predict and generate the image pixels that the character and style embedding specify.

II. RELATED WORKS

Font generation using GANs [1] has been an interesting area of research of late. This is because GANs are robust and have helped in solving challenges like image generation, text-to-image translation [2], and image-to-image translation [3]. This is because GANs have two networks, the generator and the discriminator which train simultaneously through an adversarial process. The training involves estimating new samples and also determining that samples are real or generated samples via this adversarial process in which both objectives are being pitted against each other.

Generative adversarial networks revolutionized unsupervised learning in various domains of machine learning. Regardless of their success, GANs are difficult to train because of the instability during training which is an ongoing challenge in computer vision in order to generate diverse and high-quality images. Nevertheless, there are some fair proposed guidelines for stable training of GANs [4].

1. Transformation-based methods

Various images are distinctive visually but share key structural characteristic similarities which can be used to group images together into visual domains which are basically the categories the images fall in. Apart from this, images in these domains can also be grouped according to their specific unique imprint which is basically called style. StarGAN [5,6] is a recent transformation-based method using image-to-image translation to provide a solution that transforms images considering the diverse styles in each domain in turn providing the diversity of generated images and providing the scalability over multiple domains.

Font generation methods using these transformation-based methods mostly use image-to-image translation in which they transform an existing font image from one domain to another. Transformation-based image generation is also a style transfer problem [7] that aims to create a new image by combining the content of one image with the style of another. Style transfer has been widely applied to many modern image processing problems to achieve great performance in image transformation and previously style transfer was recently used to generate fonts in neural font style transfer [8].

In transformation-based font generation, we can learn the process of transformation from the source font style to the target font style given pairs of source and target inputs [9]. This approach is basically approximating between font styles how we can take the style of one font character and transfer it to another font character. This is limited because it involves the generation of one font style at a time and can fail when there's a large number of different fonts involved. This problem can be solved by image-to-image translation techniques introduced in Pix2Pix [3], which learns the mapping from the input image to the output image. The Pix2Pix model had a one-to-many relationship problem that could cause a character to appear in multiple fonts. Zi2Zi [10] adopted the Pix2Pix architecture and added category embedding to improve the architecture and learn multiple font styles at the same time. Zi2Zi solved this problem through category embedding by concatenating a non-trainable Gaussian noise to the character embedding as the style embedding before it goes into the decoder. The decoder in Zi2Zi takes both the character and style embeddings as it goes through the process of upsampling to generate the target image.

Image style transfer is also evidently used in DCFont [11] as one part of the two major components in a system which given only a few characters can automatically generate an entire font library with realistic-looking synthetic results. In DCFont, the proposed model is given a small number of characters as input and the font feature reconstruction network estimates the deep font features of the remaining characters from the deep feature space. The result of the font feature reconstruction network is the estimated style representation of the target character and this is fed to the font style transfer network. The font style transfer network converts characters in the reference font to the corresponding output style by using the style vector from the font feature reconstruction as style. This approach fails when the source font image has a very different shape

compared to the target font image and makes it harder to accomplish image to image translation.

According to SCFont [12], these recent end-to-end approaches like transformation-based deep-learning methods often obtain synthesis results without correct structures and artifacts because they lose some information in between. This is the main reason we elected to use generative methods which learn the buildup knowledge of the structure of the characters from the ground truth.

2. Generative model-based methods

Generative model-based approaches are useful in such cases because they are trained to learn the buildup of all the characters under different fonts and learn the process of how to reproduce these designed characters and generate new fonts from this learned knowledge. Generative model-based methods emulate the vanilla GANs approach and generate new fonts from only random noise by capturing the data distribution and learning how to model the high dimension distribution of the font data.

In recent font generation literature, previous research has demonstrated that font generation can also be accomplished using generative model-based methods. These methods are different from the transformation-based methods because they generate new characters from only noise and conditioned data.

Recently GlyphGAN [13] proposed a new architecture which is a generative model-based approach to font generation. These generative model-based methods train the model to learn each font character's design principle using generative adversarial learning and come up with new characters by sampling data from the estimated manifold that the fonts compose in the learned image space. In the GlyphGAN architecture, the generator is given an input vector that consists of the style vector and character class vector. The character class vector is a one-hot encoded vector based on the classes of characters of the training data and the style vector is a uniform random vector with arbitrary information. The discriminator used the Wasserstein distance [14] to find the distance between its inputs which consist of the training examples and the generated samples. The goal in GlyphGAN was to generate an infinite variety of fonts with control on the character and style independently.

Although GlyphGAN succeeds in explicitly controlling the class of the generated font character, GlyphGAN fails to explicitly control the style of the generated font characters. In GlyphGAN explicit style control is not performed because they used a uniform random vector as the style which is not supervised information. In this paper, we approach the font generation problem similar to GlyphGAN, a generative model-based method, and solve their proposed model's problem by proposing a model that can explicitly control the style of the generated fonts.

III. PROPOSED METHOD

In GAN the generator part of the model computes this function G(z) with z being the input noise variable, the discriminator computes the function D(x) with the input x, the real image training examples. The discriminator also computes the function D(G(z)) having the generator output G(z) as input. The goal of the discriminator network D is to predict a single scalar which determines whether the input image is real or fake by maximizing the probability of correctly classifying which images are real and which ones are fake, the generator network G meanwhile tries to succeed in minimizing its failure to fool the discriminator. This pioneering GAN framework has no control over the modes of data being generated.

This problem is solved by conditional GANs (cGANs) [15] which condition the model by introducing an additional input vector y to the GAN model on both networks G and D with y serving as the character class label for the training images examples and also for the fake examples. In cGANs the input to the generator is z and y, the generator computes the function G(z, y) and the inputs to the discriminator are x, y, and G(z, y), the discriminator computes the functions D(x, y) for real examples and D(G(z, y),y) for generated examples.

In order to improve the performance of GANs more, deep convolutional GAN (DCGAN) [16] scaled up GANs further by successfully introducing

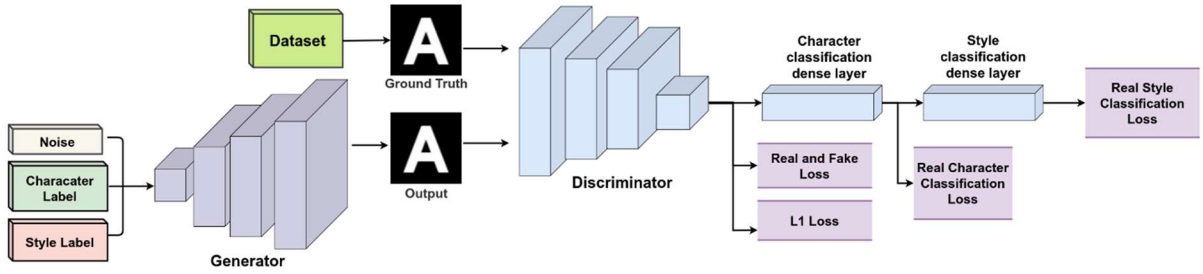


Fig. 1. The architecture of the proposed model.

the use of CNN [17−19] in GAN which makes it possible to train the discriminative and generative networks with deep convolutional layers and our model utilizes both.

Our approach solves the explicit style control problem introduced in GlyphGAN by providing our model with a separate style vector which is additional supervised information. We achieve this by specifying the generator' s input vector into three parts, the noise vector z, character label vector y, and style label vector s.

This type of conditioned input helps us calculate character class label and style label losses separately and distinctly learn each of these conditions. This bolsters our model's precision capability to intentionally specify the style of our generated font samples after training. Our model is reliably capable to perform explicit style control when generating samples from only noise.

We use cGAN loss with y and s as our conditions for characters and style classification loss, respectively,

$$L_{cGAN}(G,D) = \mathbb{E}_{x,y,s}[logD(x,y,s)] + \mathbb{E}_{z,y,s}\left[\log\left(1 - D\left(G(z,y,s)\right)\right)\right].$$

To further improve the performance of the model we also add L1 loss to the generator so that the generated samples gravitate towards clearer output,

$$L_{L1(G)} = \mathbb{E}_{x,z,y,s}[\| x - G(z,y,s) \|_1].$$

Finally, our final objective is a combination of these two functions,

$$G^* = \arg\min_G \max_D L_{cGAN}(G,D) + \lambda L_{L1}(G).$$

In our network architecture, as shown in Fig. 1, the generator gets an input of 3 vectors, the noise vector, character class vector, and character style vector. The noise vector is a 100−dimensional vector that is sampled from a normal distribution. The character class and character style vectors are one−hot encoded vectors according to the number of characters and styles, respectively i.e., the character vector is a 26−dimensional vector (26 alphabet letters) and the style vector is a 204−dimensional vector (200 training styles and 4 finetuning styles). The discriminator gets two inputs, one input is the generated fake images from the generator, the other input is the real data which are the fonts images in the dataset. Both of these inputs are already concatenated with the one−hot encoded character class and character style vectors before being fed into the discriminator. We use batch normalization in both discriminator and generator networks [20]. In the generator network, each de−convolutional layer is followed by a ReLU activation function and its final layer has a sigmoid. On the other hand, each of the discriminator network's convolutional layer is followed by a leaky ReLU activation function and after the series of convolutions, we have a sigmoid activation function. The ReLU and the leaky ReLU activation functions achieve good performance in the generator and the discriminator, respectively, and this addresses the vanishing gradient problems assuring stable training of the deep neural network [4, 15]. We also introduce two new dense layers after the flattened layer in the discriminator, one used for character classification and the other for style classification. This allows us to calculate losses separately for the character class labels and character style labels and with this, we can perform explicit style control for the generated characters.

IV. Experiments

We created our own alphabet dataset based on tools shared by IBM [21] which involves collecting a

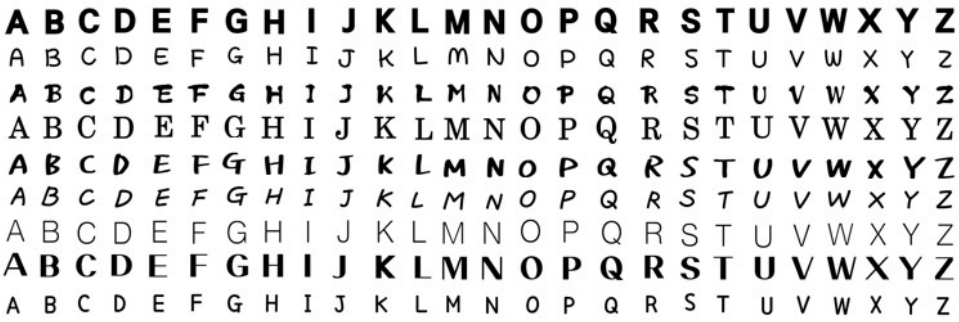


Fig. 2. Example of the diverse training samples from the dataset

large enough number of font files online and generating numerous images from A–Z in each respective font file. Our generated dataset consists of 200 different fonts used during training and another 4 different fonts used for finetuning in the selected finetuning characters, a total of 204 font files. Each style has its label e.g., 'Style 1' is labeled as a value '1' and represented by the one–hot vector for this value. This applies to all the font styles up to the 204th style and can be expanded with regard to the length of the dataset depending on the number of font files that make up the dataset used for the model.

We used TensorFlow to implement our model and trained it on an NVIDIA 2080ti GPU using Adam optimizer with a learning rate set to 0.0002 while using batch size as 64 with the input and output character images used are all of size 64 x 64 x 1. In our experiments, we do three tests to evaluate our model's performance. We test how effective the model is on controlling font style, generating new font style, and few–shot font generation.

At test time we generate samples after training the model. We feed the trained model a noise vector, character label, and style label in order to generate specified characters from A–Z in the provided style showing explicit style control. Fig. 3. shows the results of controlling style with a supervised style vector and selecting varying specified styles we want the model to generate.

In order to test our model's robustness, we introduce the model to an incomplete set of characters in a new font style. The aim is to retrain and finetune the model on these new characters. To do this we conducted 2 finetuning tests with variations in the number of characters used to finetune. We selected disjointedly 20 and 5 finetuning characters to use in separate tests. We used the structural similarity index measure (SSIM) to shortlist the finetuning characters used [22].

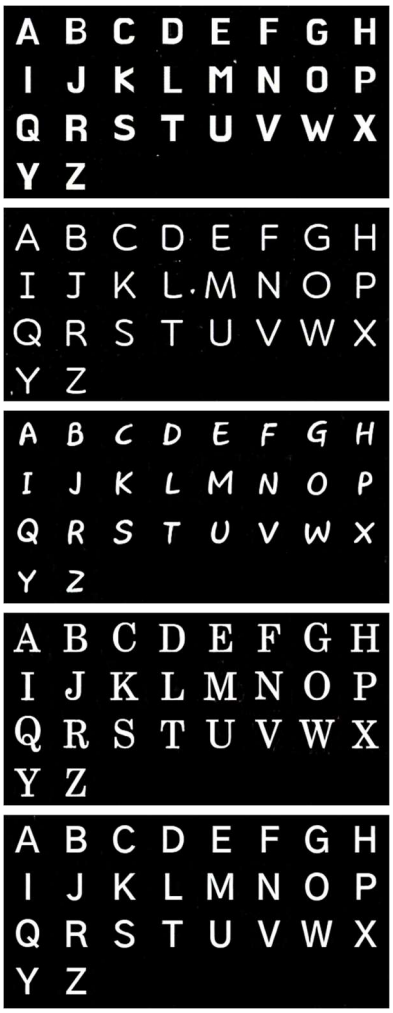
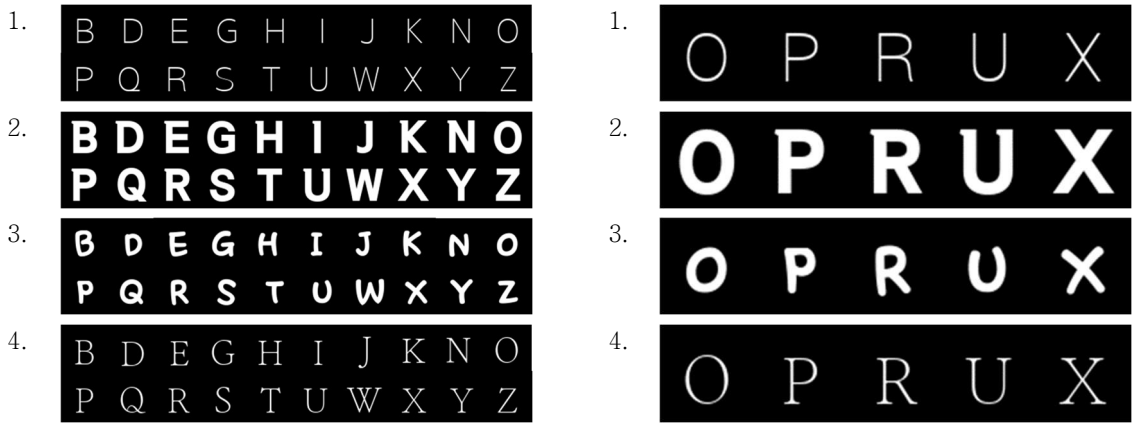


Fig. 3. Characters A–Z in different font styles generated from noise with style and character labels after training.



(a) SSIM selected 20 finetuning characters (b) SSIM selected 5 finetuning characters

Fig. 4. Characters are used to finetune the trained model.

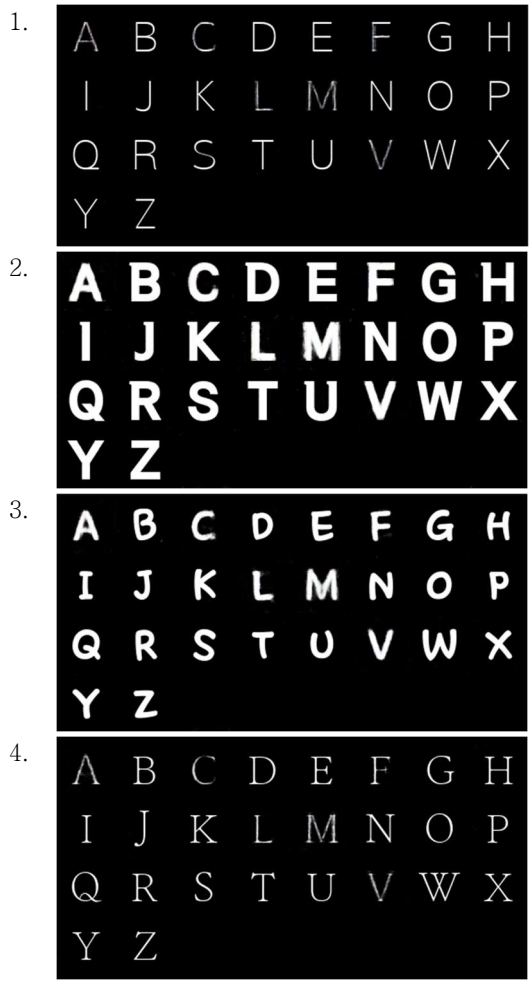


Fig. 5. Generated results from 20 characters used in finetuning.

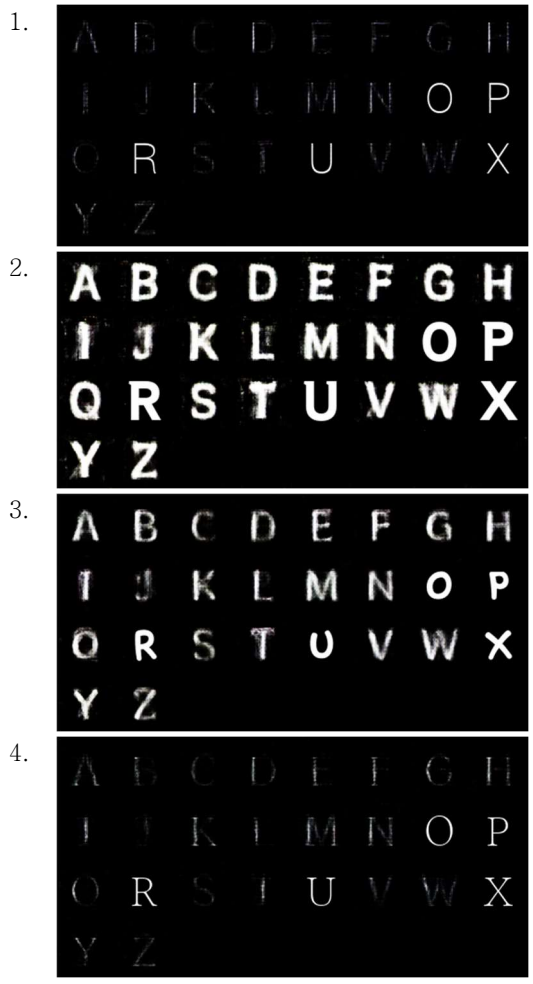


Fig. 6. Generated results from 5 characters used in finetuning.

SSIM is a simple image classifier that can tell how similar two images are by focusing on the structural comparison based on the image's pixel density values similarity. This measure is not based on exact differences between pixels and it enables us to find the images that are representative of the whole dataset. Fig. 4. shows the selected characters which are used for finetuning. These characters have the highest SSIM score compared to each of the other representative of the entire 26 characters of the English alphabet.

This section explains the few-shot font generation. Since we finetune on only a few characters, we want to find out how the other unseen characters look like in the finetuning styles. Fig. 5 and Fig. 6 show the test results of the seen and unseen characters after finetuning the trained model. These characters are in the font styles used for finetuning to test if our model can learn some extra new font styles. Ideally, we want the font designer to design only a few font characters from the alphabet and the rest complete set should be generated by the model.

V. Discussion

These results show that our model learns to generate all characters from noise in respective font styles during training while being able to learn both the character labels and style labels simultaneously.

Fig. 7 below shows the graphical representations of the character classification loss, *y,* and style classification loss, *s* that is optimized during training classification loss during training.

As the number of the training steps progresses the character classification loss *discriminator_loss_real_y* significantly decreases from 0.4 to somewhere near 0 and as the number of the training steps progresses the style classification loss *discriminator_loss_real_s* significantly decreases from 5 to somewhere near 0.

The results also show that thin fonts as demonstrated in Fig. 6. become blurry when using fewer finetuning characters while bold and thick fonts generate promising and close to the best results. The few-shot font generation results in this work look promising and substantiate the need to explore this research area while also showing potential for better and improved results for future works.

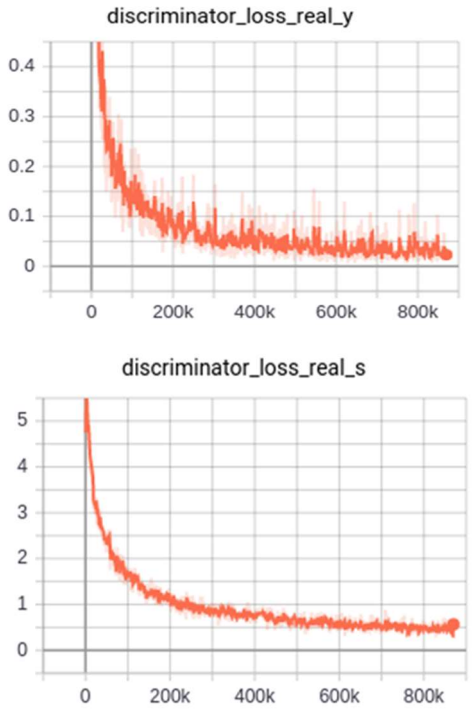


Fig. 7. Character classification loss and style

VI. Conclusion

In this paper, the objective was to create a font generation model using only the noise, character, and style vectors that can also perform a few-shot font generation. This generative model-based approach attempted to solve the problem that exists in transformation-based methods in where a single input reference can be mapped to multiple possible outputs. This model accomplishes this by learning the buildup of the characters from noise-to-image while providing explicit font style control of the generated fonts by intentionally specifying both the character and style in our generated samples. This contribution helps the generative-model to learn these specific additional informative attributes of the fonts that we generate which significantly impacts the loss functions in our model.

In our future work, we will explore a new way to improve the results of this approach with an emphasis on the few-shot font generation process and also diversifying the dataset to work with Chinese, Japanese, and Korean (CJK) character datasets.

REFERENCES

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial networks," *NIPS, Proc. of Neural Information Processing Systems*, pp. 2672-2680, 2014

[2] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," *arXiv preprint* 1605.05396, 2016

[3] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image translation with conditional adversarial networks," *CVPR, Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017

[4] J. Brownlee, "Tips for Training Stable Generative Adversarial Networks, https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/(accessed July 24, 2020)

[5] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation," *CVPR, Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018

[6] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha, "StarGAN v2: Diverse image synthesis for multiple domains," *arXiv preprint* arXiv:1912.01865, 2019

[7] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," *CVPR, Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414-2423, 2016

[8] G. Atarsaikhan, B. K. Iwana, A. Narusawa, K. Yanai, and S. Uchida, "Neural font style transfer," *ICDAR, Proc. of 14th International Conference on Document Analysis and Recognition*, pp. 51-56, 2017

[9] Y. Tian, "Rewrite: Neural style transfer for Chinese fonts," https://github.com/kaonashityc/Rewrite(accessed April 25, 2020)

[10] Y. Tian, "Zi2Zi: Master Chinese calligraphy with conditional adversarial networks," https://github.com/kaonashi-tyc/zi2zi (accessed April 25, 2020)

[11] Y. Jiang, Z. Lian, and Y. Tang, and J. Xiao, "DCFont: An end-to-end deep Chinese font generation system," *SIGGRAPH Asia, Proc. of SIGGRAPH Asia Technical Briefs*, 2017

[12] Y. Jiang, Z. Lian, Y. Tang, and J. Xiao, "SCFont: Structure-guided Chinese font generation via deep stacked networks," *AAAI, Proc. of 33rd AAAI Conference on Artificial Intelligence*, 2019

[13] H. Hayashi, K. Abe, and S. Uchida, "GlyphGAN: Style-consistent font generation based on generative adversarial networks," *arXiv preprint* arXiv:1905.12502, 2019

[14] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," *Machine Learning, Proc. of 34th International Conference on Machine Learning*, pp. 214-223, 2017

[15] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint* arXiv:1511.06434, 2015

[16] M. Mirza, and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint* arXiv:1411.1784, 2014

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *NIPS, Proc. of Neural Information Processing Systems*, pp. 1097-1105, 2012

[18] S. Noh, "Classification of Clothing Using Googlenet Deep Learning and IoT based on Artificial Intelligence," *Smart Media Journal*, vol.9, no.3, pp. 41-45, 2020

[19] Y. Kim, and J. Kim, "A Study on the Performance of Enhanced Deep Fully Convolutional Neural Network Algorithm for Image Object Segmentation in Autonomous Driving Environment," *Smart Media Journal*, vol.9, no.4, pp. 9-16, 2020

[20] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating deep network training by reducing internal covariate shift," *ICML, Proc. of International Conference on Machine Learning*, 2015

[21] IBM; "Handwritten Korean Character Recognition with TensorFlow and Android," https://github.com/IBM/tensorflow-hangul-recognition (accessed July 27, 2020)

[22] J. Nilsson, T. Akenine-Moller, "Understanding SSIM," *arXiv preprint* arXiv:2006.13846, 20

Authors

Finlyson Mwadambo Msiska

He received his B.S. degree in Information and Communications Technology from Daeyang University, Malawi in 2018. Currently, he is a graduate student at Soongsil University. His research interests include Font Systems, Deep Learning, and System Software.

Ammar Ul Hassan

He received his B.S. degree in Department of Software engineering, from International Islamic University Islamabad, Pakistan in 2013. He then received his M.S. degree in School of Computer Science and Engineering from Soongsil University, Seoul, South Korea in 2018. He is currently taking his Ph.D. degree in Department of computer science from Soongsil University Seoul, South Korea. His current research interests are Deep learning, Computer vision, Generative models, making font environment for new fonts in Linux Operating System.

Jaeyoung Choi

He received his B.S. degree in Control and Instrumentational Engineering from Seoul National University, Seoul, Korea, in 1984, the M.S. degree in Electrical Engineering, University of Southern California in 1986, and the Ph.D. degree in Electrical Engineering from Cornell University in 1991. He is currently a professor of School of Computer Science and Engineering at Soongsil University, Seoul, Korea. His current research interests include Korean Typography, Distributed Computing, and HPC.

Jaewon Yoo

He received his B.S. degree in Business Administration from Hankuk University of Foreign Studies, Seoul, Korea, in 1996, the M.S. degree in Marketing, Hanyang University in 2000, and the Ph.D. degree in Marketing from Oklahoma State University in 2011. He is currently an associate professor of the department of Small Business and Entrepreneurship at Soongsil University, Seoul, Korea. His research and teaching interests focus on service marketing strategies, salesforce management, marketing research, customer orientation, and retailing.