KIPS Trans. Softw. and Data Eng.
Vol.11, No.12 pp.499~508
ISSN: 2287-5905 (Print), ISSN: 2734-0503 (Online)

https://doi.org/10.3745/KTSDE.2022.11.12.499

CKFont2: An Improved Few-Shot Hangul Font Generation Model Based on Hangul Composability

Jangkyoung Park† · Ammar Ul Hassan†† · Jaeyoung Choi†††

ABSTRACT

A lot of research has been carried out on the Hangeul generation model using deep learning, and recently, research is being carried out how to minimize the number of characters input to generate one set of Hangul (Few-Shot Learning). In this paper, we propose a CKFont2 model using only 14 letters by analyzing and improving the CKFont (hereafter CKFont1) model using 28 letters. The CKFont2 model improves the performance of the CKFont1 model as a model that generates all Hangul using only 14 characters including 24 components (14 consonants and 10 vowels), where the CKFont1 model generates all Hangul by extracting 51 Hangul components from 28 characters. It uses the minimum number of characters for currently known models. From the basic consonants/vowels of Hangul, 27 components such as 5 double consonants, 11/11 compound consonants/vowels respectively are learned by deep learning and generated, and the generated 27 components are combined with 24 basic consonants/vowels. All Hangul characters are automatically generated from the combined 51 components. The superiority of the performance was verified by comparative analysis with results of the zi2zi, CKFont1, and MX-Font model. It is an efficient and effective model that has a simple structure and saves time and resources, and can be extended to Chinese, Thai, and Japanese.

Keywords : Deep Learning, Few-Shot, Initial/Middle/Final Components, CKFont, Korean 14 Characters

CKFont2: 한글 구성요소를 이용한 개선된 퓨샷 한글 폰트 생성 모델

박 장 경† · Ammar Ul Hassan†† · 최 재 영†††

요 약

딥러닝을 이용한 한글 생성 모델에 대한 연구가 많이 진행되었으며, 최근에는 한글 1벌을 생성하기 위하여 입력되는 글자 수를 얼마나 최소화할 수 있는지(Few-Shot Learning)에 대하여 연구되고 있다. 본 논문은 28개 글자를 사용하는 CKFont (이하 CKFont1) 모델을 분석하고 개선하여 14개 글자만을 사용하는 CKFont2 모델을 제안한다. CKFont2 모델은 28글자로 51개 한글 구성요소를 추출하여 모든 한글을 생성하는 CKFont1 모델을, 24개의 구성요소(자음 14개와 모음 10개)를 포함한 14개의 글자만을 이용하여 모든 한글을 생성하는 모델로 성능을 개선하였으며, 이는 현재 알려진 모델로서는 최소한의 글자를 사용한다. 한글의 기본 자/모음으로부터 쌍자음(5), 복자음(11)/복모음(11) 등 27개를 딥러닝으로 학습하여 생성하고, 생성된 27개 구성요소를 24개의 기본 자/모음과 합한 51개 구성요소로부터 모든 한글을 자동 생성한다. zi2zi, CKFont1, MX-Font 모델 생성 결과와 비교 분석하여 성능의 우수성을 입증하였으며, 구조가 간결하고 시간과 자원이 절약되는 효율적인 모델로 한자나 태국어, 일본어에도 확장 적용이 가능하다.

키워드 : 딥러닝, 퓨샷학습, 초성/중성/종성, CKFont, 한글 14글자

1. 서 론

오늘날 딥러닝을 이용하여 새로운 폰트를 생성하는 것이 매우 효율적임은 이미 널리 알려진 사실이다. 한글이나 한자의 경우 수백자에서 수십자로 감소된 입력 글자로 수만자의 다른 글자를 같은 스타일로 단시간에 생성할 수 있다[1-2].

그러나 여전히 많은 입력 글자를 필요로 하여 사용하기가 쉽지 않다. 최근에는 수요자가 기존의 글자체(Printed Font)가 아닌 자신이나 기업만의 고유한 특징을 나타내는 글자체를 새롭게 디자인하여 사용하려는 요구가 증가하고 있다[3]. 이러한 경우에 절대적으로 필요한 것이 실용성 및 편의성과 관련된 입력 글자 숫자의 최소화이다[4-6].

입력글자의 숫자가 많으면 그만큼 실용성과 편이성이 떨어

† 비 회 원 : 숭실대학교 컴퓨터학부 박사과정
†† 준 회 원 : 숭실대학교 컴퓨터학부 박사과정
††† 종신회원 : 숭실대학교 컴퓨터학부 교수
Manuscript Received : April 21, 2022
First Revision : June 17, 2022
Accepted : June 26, 2022
* Corresponding Author : Jaeyoung Choi(choi@ssu.ac.kr)

짐을 의미하며, 입력 글자를 줄인다는 것은 그만큼 실용성과 편이성이 증대된다. 그러나 무조건 글자 숫자를 줄일 수 없는 이유는 적은 숫자의 글자만으로는 고품질의 결과물을 생성하기 어렵기 때문이다. 생성된 글자의 품질 저하 문제를 해결할 수 없으면 입력글자 수의 감소 의미가 상실된다. 다시 말하면 글자 수를 줄이면 품질이 저하되는 문제에 직면하게 된다. 이러한 딜레마를 해결하기 위하여 많은 연구가 진행되고 있다[4-8]. 한글 생성과 관련하여, 잘 알려진 퓨샷 학습(Few-shot Font Generation, FFG) 모델로는 28자를 사용하는 네이버의 DMFont[4], 이를 개선한 LFFont[5], 그리고 다시 이를 개선한 MXFont[6]가 있다. 또한 글자의 골격(skeleton)을 이용하는 SKFont[7] 모델이 있으며, CKFont 모델(이하 CKFont1)[8]은 글자의 구성요소를 분리하여 스타일 변환 후 재합성하는 방법으로 글자를 생성할 수 있으며 한글과 한자 등 구성요소로 분리되는 글자에 적용이 가능하다.

본 논문에서는 28개의 글자를 사용하는 CKFont1 모델을 개선하여 14개 자음과 10개 모음을 포함하는 14 글자만으로 우수한 품질의 한글 1벌을 생성하는 최소글자 모델 CKFont2를 제안한다.

한글은 기본적으로 14개의 자음과 10개의 모음으로 이루어진 글자이며, 모든 한글은 19개의 초성과 21개의 중성, 그리고 27개의 종성 등 67개 (19+21+27) 구성요소가 초/중/종성 순서대로 조합되어 모두 11,172자(19x21x28, 종성이 없는 경우 포함)의 조합형 형태의 글자를 가진다. 67개의 구성요소 중에서 초성과 중성이 중복된 16개 구성요소를 제외하면 51개 구성요소를 가진다. 이 중 5개 쌍자음, 11개의 복자음, 11개의 복모음은 기본 자/모음의 결합으로 생성될 수 있으므로, 이를 다시 제외하면 14개 자음과 10개 모음인 24개의 구성요소가 된다. 본 논문은 이러한 조합형 한글의 특성을 이용하여 24개 구성요소를 포함하는 14글자로부터 51개 구성요소를 획득하여 모든 한글을 생성할 수 있으며, 이미지에서 이미지로 변환하는 I2I (Image-to-image translation) 방법의 cGAN (conditional Generative Adversarial Networks)[11]을 기반으로 한글 자동 생성 모델인 CKFont2를 제안한다.

CKFont1 모델을 개선하여 본 논문에서 제안하는 CKFont2 모델의 특성과 기여하는 바는 다음과 같다.

- ◦ CKFont1 모델 보다 적은 글자 사용 (28자 => 14자)
- ◦ 글자 수 감소로 인한 품질 저하 방지를 위해
  - 스타일 인코더의 깊이(layer) 증가(6 -> 9)
  - 아웃풋 레이어 Loss 대신 Feature Loss 사용
- ◦ 자원과 시간이 절약되는 효율적인 모델
  - 스타일 레이블 대신 스타일 인코더를 사용하여 미세 조정(finetuning)이 필요 없는 구조
  - 글자 수의 감소와 미세조정 불필요로 모델의 실용성과 편의성 증대
- ◦ 비교 대상 모델 대비 고품질 결과물 생성

2. 관련 연구

딥러닝을 이용한 전통적인 폰트 생성 방법은 GAN을 이용하여 여러 폰트 스타일로 이미지를 변환하는 I2I 방법이다[12]. 소스 도메인(Domain)에서 서로 다른 도메인으로 매핑(mapping) 하는 것을 학습하여, 원본 이미지 콘텐츠와 스타일 정보를 얻어 서로 다른 스타일의 콘텐츠 이미지로 생성하는 방법이다. 이 방법은 정확한 콘텐츠와 스타일 정보를 얻고, 학습하는 동안 정보가 약화되거나 손실되지 않도록 지속하는 것이 고품질의 이미지를 생성하는 중요한 열쇠이다. 퓨샷 학습은 최소한의 글자에서 이러한 콘텐츠와 스타일 정보를 분리하기 위해 다음의 2가지 방법이 사용되고 있다. (1) 글자 전체에서 콘텐츠와 스타일의 정보를 각각 얻는 전역적(global)인 표현 방법과 (2) 구성요소 기반의 지역적인(local) 스타일 정보를 글자의 구성요소에서 얻는 두 가지 방법이다. (1)은 글자 전체의 이미지를 학습하여 글자의 내용과 스타일 정보를 얻는 것인데, 영어와 달리 한글이나 한자처럼 글자가 복잡한 경우에는 정확한 스타일 정보를 얻기가 용이하지 않다. 따라서 (2) 방법으로 복잡한 글자의 구성요소를 분리하여 스타일 정보를 얻는 방법이 보다 우수한 성능을 나타내고 있다[4-7].

(1)의 방법을 사용하는 Rewrite[13] 및 zi2zi[14]와 같은 모델들은 폰트 스타일 정보를 얻기 위해 수천 개의 문자 쌍으로 소스 스타일을 대상 스타일에 매핑하는 방법을 학습하여 한자를 생성한다. 이를 기반으로 CNN(Convolutional Neural Network)을 이용하는 DCFont[15], 소스 글리프에서 골격/획을 추출하여 대상 스타일로 변환하는 SCFont[16] 등 한자 폰트 생성 모델이 개발되었다. 이러한 방법은 학습 중에 관찰된(seen) 폰트에 대해서는 고품질 폰트 이미지를 생성하지만 학습 중에 관찰되지 않은 처음 보는(unseen) 폰트 이미지를 생성할 때는 품질이 크게 저하되며 이를 극복하기 위해 추가적인 미세 조정 단계가 필요하고, 최적의 결과를 얻기까지는 많은 시간이 소요된다.

(2)의 방법을 사용한 DM-Font[4]는 한글의 구성요소 정보를 한 번에 추출하고 전부를 메모리에 저장하여 필요시 매번 찾아서 사용하므로 구성요소를 사용하여 글자 수를 줄이는 데는 성공하였으나 시간이 많이 소요되고 메모리에 저장된 한글만 가능하다.

이를 개선하여 한자가 가능한 LF-Font[5]는 구성요소별 스타일 인코더와 콘텐츠 인코더를 사용하여 미세조정 없이 한글은 물론 한자 생성도 가능하지만, 구성요소가 없는 부분을 인수분해 형식(factorization)으로 추정하여 사용함으로써 평균정보를 사용한다.

다시 이를 개선하여 다국어용으로 개발된 MX-Font[6]는 각 참조 이미지에 대해 다중 인코더를 사용하여 콘텐츠와 스타일 간의 정보를 분리하여 언어 간 작업이 가능하다. DM- Font, LF-Font 보다 성능이 우수하고 고품질의 결과물을 생성하지만 처음 보는 글자가 있으면 몇 번의 추가적인 학습이 필요하다.

CKFont1[8]은 소스 글자를 대상 글자의 분리된 구성요소와 짝을 짓고 대상 글자의 분리된 구성요소에서 스타일 정보

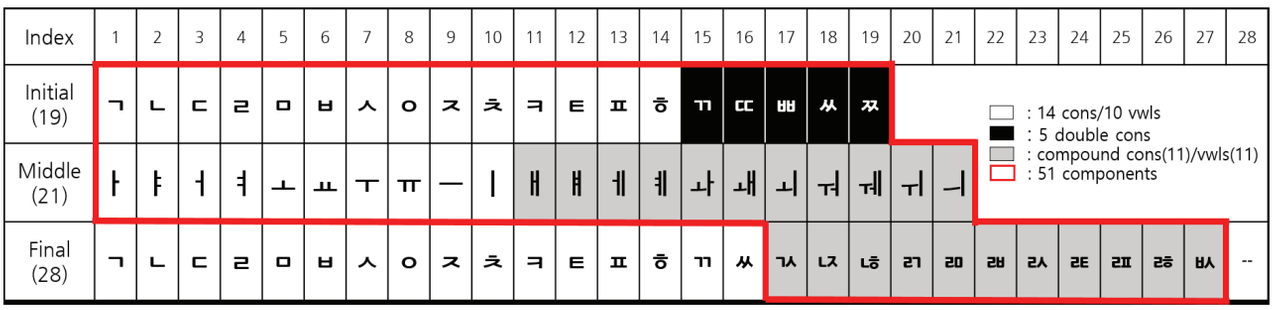

Fig. 1. Hangul's Initial, Middle and Final Components and Consonants/Double-Consonants/Compound-Consonants, Vowels/Double Vowels and 51 Components

를 획득하여 pix2pix로 결과물을 생성한다. 따라서 구성요소로 분리하고 조합하는 학습을 통하여 처음 보는 모든 글자를 생성하기 때문에 한글은 물론 한자나 태국어도 확장 적용이 가능하며 짧은 시간에 고품질의 결과물을 생성한다.

3. 한글의 구조적 특성

3.1 한글의 구성요소

한글은 기본자음 14자와 기본모음 10자의 24자를 사용하여 총 11,172자를 만들 수 있다. 모든 글자는 [초성 + 중성 + (종성)]의 순서대로 조합된 형태를 가지며, 초성에 사용되는 자음은 모두 19개의 자음으로 14개 기본 자음과 5개의 쌍자음이 사용되며, 중성은 모두 21개의 모음으로 10개 기본모음과 11개의 복모음이 사용된다. 종성에 사용되는 27개 자음은 14개 기본 자음과 2개의 쌍자음, 11개의 복자음이 사용된다.

Fig. 1에서 보는 바와 같이 51개 구성요소를 살펴보면 기본 자음과 모음 외에 5개 쌍자음과 11개 복자음, 11개 복모음은 모두 기본 자음과 모음의 조합으로 이루어진 구성요소임을 알 수 있다. 다시 말하면 14개 기본 자음과 10개 모음의 조합으로 나머지 27개의 구성요소를 생성할 수 있다. 즉, Fig. 2에서 보는 바와 같이 쌍자음은 자음의 중복으로 생성되며, 복자음과 복모음은 기본 자/모음의 조합으로 만들어진다.

따라서 모든 한글의 구성요소는 초성과 중성, 종성의 합인 67개이지만 종성에 중복된 16개 자음을 제외하면 51개가 되며, 다시 27개의 쌍자음과 복자/모음이 기본 자음과 모음의 조합으로 구성될 수 있으므로 이를 제외하면 최종적으로 24개의 기본 구성요소만 남는다.

즉 한글은 기본 자/모음 24개로 모든 글자를 생성할 수 있다. 24개의 자/모음 구성요소로 나머지 27개 쌍자음과 복자/모음 구성요소도 딥러닝을 이용하여 생성할 수 있다. 생성된

ㄲ = {ㄱ + ㄱ} 과 = {ㅗ + ㅏ}

ㄺ = {ㄹ + ㄱ} 새 = {ㅅ + ㅐ + ㅣ}

Fig. 2. Examples of Double Consonants and Compound Consonants/ Vowels Made by Combining Basic Consonant and Vowels

값 = {ㄱ(initial), ㅏ(middle), ㅄ(final)} ➡ ㅄ = {ㅂ + ㅅ}

웰 = {ㅇ(initial), ㅞ(middle), ㄹ(final)} ➡ ㅞ = {ㅜ + ㅓ + ㅣ}

Fig. 3. Hangul is Decomposed or Combined into Components, and Double/Compound Consonants and Vowels can be Decomposed or Combined into Basic Consoants/Vowels Again

가	나	더	려	모	부	쇼
야	져	쵸	켜	튜	프	히

Fig. 4. Example of 14 Characters Including Both Basic Consonants and Vowels

27개 구성요소와 24개 구성요소를 합하여 이를 다시 입력으로 사용하면 모든 한글을 생성할 수 있다. Fig. 3에서 보듯이 한글은 구성요소로 분해되거나 조합되며, 쌍/복자음과 복모음은 다시 기본 자/모음으로 분해되거나 조합될 수 있다.

결국 한글의 모든 글자는 24개 구성요소만 가지고 생성할 수 있다. Fig. 4는 초성에 있는 14개 기본자음 구성요소와 중성의 10개 기본모음을 모두 포함하는 14 글자의 샘플 셋(set)이며, 자/모음을 모두 포함하는 여러 종류의 샘플을 만들 수 있다.

3.2 결합규칙(Combining Rule)

한글의 자음과 모음은 유니코드 U+1100~U+11FF, U+A960~U+A97F, U+D7B0~U+D7FF에 배당되어 있으며 한글을 조합형으로 구현할 수 있도록 초·중·종성을 일일이 배당하였다. 여기에는 옛한글 낱자들도 포함되어 있어, ㆆ과 같은 옛한글도 옛한글 전용 폰트만 있으면 문제없이 사용할 수 있다. 글자는 가나다순으로 U+AC00~U+D7A3에 11,172자가 배당되어 있다[9]. 이러한 유니코드를 십진수로 환산하면 초·중·종성 51개 구성요소는 12,593~12,643이 되며, 11,172개의 글자는 44,032~55,203이 된다. 예를 들면, 유니코드 'ㄱ' 은 '0x3131'(16진수 3131)이며, 십진수 숫자로 나타내면 12,593이고 이는 'ㄱ'의 십진수 유니코드가 된다.

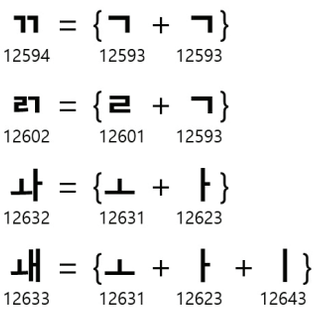

Fig. 5. Example of Creating Double Consonants and Compound Consonants/Vowels Using Unicode

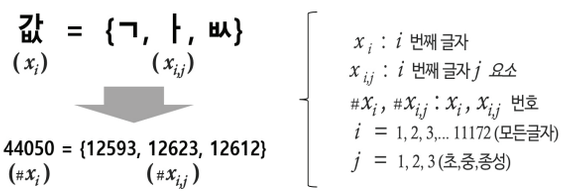

Fig. 6. Example of Creating a Character Using Order

같은 방법으로 글자 '가'의 유니코드는 '0xAC00'으로 십진수 44,032를 '가'의 십진수 유니코드로 사용할 수 있다. 다시 말하면 한글의 자모나 글자를 알면 십진수 유니코드를 알 수 있고, 반대로 십진수 유니코드를 알 면 글자를 알 수 있다.

또한 Fig. 5에서 보는 바와 같이 쌍자음과 복자/모음은 기본 자음과 모음의 합으로 나타낼 수 있으며, 이를 십진수 유니코드로 대체할 수 있다. 이를 컴퓨터가 인식할 수 있도록 결합 규칙을 만들면 Fig. 5의 예시에서 보는 바와 같이 십진수 유니코드의 조합으로 표현할 수 있다.

같은 방법으로 글자는 초 · 중 · 종성의 합으로 표시할 수 있다. Fig. 6에서와 같이 글자(x_i)는 초 · 중 · 종성 구성요소($x_{i,j}$)를 가지며, 글자의 값(#x_i)과 요소의 값(#$x_{i,j}$)으로 표시할 수 있다.

한글 24개 기본 자/모음으로부터 27개의 쌍자음, 복자음/복모음을 순서대로 조합하며, Table 1은 16개의 쌍자음과 복자음을 표시하였으며, Table 2는 11개 복모음을 나타내었다.

이렇게 만들어진 모든 자음과 모든 모음을 조합하여 하나하나의 글자를 표시할 수 있으며, 같은 방법으로 초성과 중성, 그리고 종성을 순서대로 조합하여 모든 글자를 나타낼 수 있다. 한글 표준코드체계(KS X 1001) 11,172자 중 일부 글자에 대하여 십진수 유니코드를 이용한 결합규칙을 Table 3에 예시하였다. 글자를 사용하지 않고 글자를 생성하는 것은 딥러닝 분야에서 의미 있는 시도이며 입력데이터의 감소는 물론 컴퓨터 파워의 절약과 속도를 증진시킨다.

각 구성요소의 십진수 유니코드를 그대로 사용하여 모든 글자를 십진수 유니코드의 조합으로 표시할 수 있으며, 본 논문의 공개 프로그램(generate-combine-rule-imgs.py)을 이용하여 모든 한글 11,172자에 대한 고유한 결합 규칙을 자동으로 생성할 수 있다.

Table 1. Double/Compound Consonants Combining Rule

idx	consnt (unicode)		combining rule	dbl/cmpnd consnt (unicode)	
1	ㄱ	12593	12593 + 12593 → 12594	12594	ㄲ
2	ㄴ	12396	12599 + 12599 → 12600	12600	ㄸ
3	ㄷ	12599	12610 + 12610 → 12611	12611	ㅃ
4	ㄹ	12601	12613 + 12613 → 12614	12614	ㅆ
5	ㅁ	12609	12616 +12616 → 12617	12617	ㅉ
6	ㅂ	12610	12593 + 12613 → 12595	12595	ㄳ
7	ㅅ	12613	12396 + 12616 → 12597	12597	ㄵ
8	ㅇ	12615	12396 + 12622 → 12598	12598	ㄶ
9	ㅈ	12616	12601 + 12593 → 12602	12602	ㄺ
10	ㅊ	12618	12601 + 12609 → 12603	12603	ㄻ
11	ㅋ	12619	12601 + 12610 → 12604	12604	ㄼ
12	ㅌ	12620	12601 + 12613 → 12605	12605	ㄽ
13	ㅍ	12621	12601 + 12620 → 12606	12606	ㄾ
14	ㅎ	12622	12601 + 12621 → 12607	12607	ㄿ
			12601 + 12622 → 12608	12608	ㅀ
			12610 + 12613 → 12612	12612	

Table 2. Compound Vowels Combining Rule

idx	vowels (unicode)		combining rule	cmp vowels (unicode)	
1	ㅏ	12523	12523 + 12643 → 12624	12624	ㅐ
2	ㅑ	12625	12625 + 12643 → 12626	12626	ㅒ
3	ㅓ	12627	12627 + 12643 → 12628	12628	ㅔ
4	ㅕ	12629	12629 + 12643 → 12630	12630	ㅖ
5	ㅗ	12631	12631 + 12523 → 12632	12632	ㅘ
6	ㅛ	12635	12631 + 12523 + 12643 → 12633	12633	ㅙ
7	ㅜ	12636	12631 + 12643 → 12634	12634	ㅚ
8	ㅠ	12640	12636 + 12627 → 12637	12637	ㅝ
9	ㅡ	12641	12636 + 12627 + 12643 → 12638	12638	ㅞ
10	ㅣ	12643	12636 + 12643 → 12639	12639	ㅟ
			12641 + 12643 → 12642	12642	ㅢ

Table 3. Example of Characters Combining Rule

char unicode (#x_i)	char (x_i)	initial ($x_{i,1}$)	mdl (x_i)	final ($x_{i,3}$)	combining rule		
					initial (#$x_{i,1}$)	mdl (#$x_{i,2}$)	final (#$x_{i,3}$)
44032	가	ㄱ	ㅏ	-	12593	12623	-
44033	각	ㄱ	ㅏ	ㄱ	12593	12623	12593
44036	간	ㄱ	ㅏ	ㄴ	12593	12623	12596
44039	갇	ㄱ	ㅏ	ㄷ	12593	12623	12599
44040	갈	ㄱ	ㅏ	ㄹ	12593	12623	12601
44041	갉	ㄱ	ㅏ	ㄺ	12593	12623	12602
44602	갊	ㄱ	ㅏ	ㄻ	12593	12623	12603
44048	감	ㄱ	ㅏ	ㅁ	12593	12623	12609
44049	갑	ㄱ	ㅏ	ㅂ	12593	12623	12610
44050	값	ㄱ	ㅏ	ㅄ	12593	12623	12612

4. CKFont2 모델 구조

4.1 개 념

CKFont2 모델은 한글이 기본적인 자음 14자와 모음 10자로 이루어져 있는 사실을 실제적으로 컴퓨터가 이를 증명해 보이는 모델이다. 기본 자/모음에서 27개 요소(Learnable Components)를 생성하면 모두 51개 구성요소를 얻을 수 있고, 이를 이용하여 모든 글자를 생성할 수 있다. CKFont1은 51개 구성요소와 이 구성요소를 모두 포함하는 28개 글자를 입력하여 모든 한글을 생성하는 모델이다. 이를 활용하여 CKFont2는 Fig. 7에서 보듯이 사전에 정의된 임의의 14개 글자를 입력으로 받고 그로부터 24개 구성요소를 얻어서 결합규칙을 학습한 Model $\boldsymbol{f}$를 이용하여 27개의 구성요소를 생성한다. 최종적으로 이를 결합한 51개 요소를 Model $\boldsymbol{g}$에 입력하여 모든 글자를 생성하는 구조이다.

Fig. 7에서 보는 바와 같이 새로운 모델 CKFont2는 14글자에서 얻어진 24개의 기본 자/모음(x)만을 사용하며 먼저 Model $\boldsymbol{f}$를 이용하여 27개 구성요소(y, Learnable Comporents)를 사전에 정의된 결합 규칙(Combining Rule)에 따라 생성하고($f(x) \rightarrow y$), 이를 기본 24개 자/모음과 합하여 51개 구성요소($x+y$)를 Model $\boldsymbol{g}$에 입력한다. Model $\boldsymbol{g}$는 입력된 51개 한글 구성요소를 이용하여 모든 글자를 생성한다 ($g(x+y) = g[x+f(x)] \rightarrow z$). Model $\boldsymbol{f}$는 I2I 방법을 이용한 cGAN 모델을 수정하였으며, Model $\boldsymbol{g}$는 28글자와 51개 구성요소를 사용하여 모든 한글을 생성하는 CKFont1 모델을 개선하였다.

4.2 구 조

앞에서 설명한 개념을 적용하여 만든 CKFont2 모델의 구조는 Fig. 8과 같다. Fig. 8A는 14글자로부터 자/모음 24개를 입력받아 27개 구성요소를 생성하고 (Fig. 8B)는 이를 24개 구성요소와 합하여 모델로 입력 (51개 구성요소)되어 모든 글자를 생성한다. Fig. 8C는 이때 사용되는 생성기(Generator, G)로 인코더(Encoder, En)에서 구성요소(x)와 대상 요소 label을 입력받아 convolution, instance normalization, relu 함수를 거치며 content layer는 6 계층(layer), style layer는 9 계층을 지나면서 다운 샘플링된다. 디코더(Decorder, De)는 이를 입력받아 구성요소 결합 규칙이 적용되고, 다시 업샘플링되어 대상 스타일별 구성요소(y)를 생성한다. 인코더(En) 계층과 디코더(De) 계층은 서로 연결되어 있으며(Skip Connection), 생성된 결과물 x와 y는 Fig. 8D의 판별기(Discriminator, D)에 입력된다. 판별기는 소스 이미

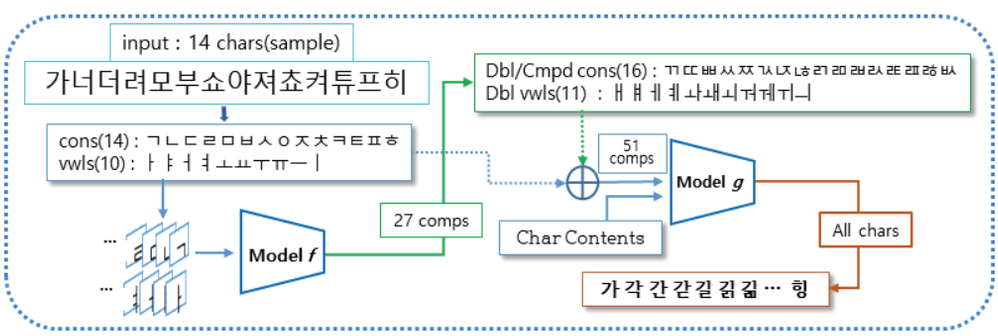

Fig. 7. Concept Diagram of CKFont2 Model

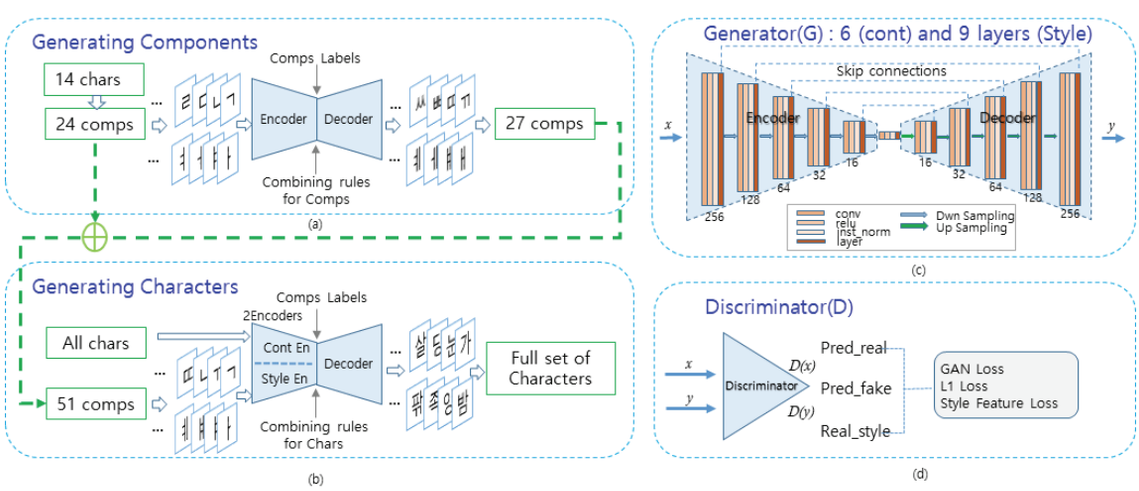

Fig. 8. Overview of CKFont2 Architecture. Figure (a) and (b) is the Connected Diagram that Generates the Components and Characters Respectively. (c) and (d) is the Diagram of the Generator and the Discriminator, Respectively

지(x)와 G에서 생성되는 결과물(y) 이미지를 비교하여 GAN 손실과 이진분류 (Binary Classification) 값 (D(y))을 계산하고, 스타일이 대상 스타일과 같은지를 판별(D(x))한다.

CKFont1은 고정 스타일 레이블(각 스타일에 대한 하나의 핫 벡터(onehot vector))를 사용하여 디코더에 스타일을 삽입한다. 고정 스타일 레이블을 사용할 때의 단점 중 하나는 학습 폰트 스타일만 학습한다는 것이다. 따라서 추론하는 동안 unseen 폰트 스타일에 대해 네트워크는 새로운 폰트 스타일로 미세조정 되어야 한다. 이러한 문제를 극복하기 위해 CKFONT2는 문자의 구성요소 이미지를 가져와서 폰트 이미지를 생성하기 위해 디코더에 입력되는 스타일 특징을 추출하는 별도의 스타일 인코더를 사용하며 따라서 미세조정을 하지 않아도 된다(code참조:http://github.com /xjnpark/ CKFont2)

4.3 손실함수

모델의 손실함수(Loss Function, L)는 Equation (1)과 같이 Adversarial 손실 (L_{ADV}), 스타일 손실(Ls), L1 손실 (L_{L1})의 합으로 표시된다. Equation (1)에서 λ_s와 λ_{L1}은 학습할 때 스타일과 L1 손실에 대한 Hyper parameter로 각 손실에 대한 가중치 역할을 한다.

$$L = arg\ min_G\ max_D\ L_{ADV}(G,D) + \lambda_s L_S(G,D) + \lambda_{L1} L_{L1}(G) \quad (1)$$

Equation (1)의 각 손실함수는 다음과 같다.

Adversarial 손실 (L_{ADV}) : cGAN의 L_{ADV}는 min-max Loss로 알려진 Equation (2)와 같이 표시되며[8], G는 최소화(Generator Loss)하고 D는 최대화(Discriminator Loss)한다. 판별기 D는 입력받은 가짜 이미지($G(x)$)가 진짜(True:1)인지 가짜(False:0)인지를 판별하여 최대화($D(y)$≒1)한다. 생성기 G는 가짜 이미지를 생성하고($G(x)$) D가 진짜로 판별하도록($D(G(x))$≒1) G(x)가 최소가 되도록($G(x)$ = 0)한다.

$$min_G\ max_D\ L_{ADV}(G,D) = E_y\ [log\ D(y)] + E_x\ [log(1-D(G(x)))] \quad (2)$$

Style 분류 손실 (L_S) : 일대다의 스타일 변환된 이미지를 생성하기 위해서 D는 진짜와 가짜를 판별하고 스타일이 대상 스타일과 같은지를 판별($D(y_S)$)한다. 이때 사용된 이미지는 생성된 마지막 이미지 이전 단계의 이미지(feature)를 사용하였다. 이미지의 폰트 스타일을 유지하기 위하여 D는 폰트 스타일을 예측(0~1)하고, G로 피드백하여 손실을 줄이고 G가 올바른 스타일의 폰트를 생성하도록 한다. 이는 Adversarial 손실과 같은 개념으로 $D(y_S)$를 최대화하고 $G(x_S)$를 최소화하는 Equation (3)과 같다.

$$min_G\ max_D\ L_S(G,D) = E_y\ [log\ D(y_S)] + E_x\ [log\ (1-D(G(x_S)))] \quad (3)$$

L1 손실(L_{L1}) : L_{L1}은 G가 가짜 이미지를 생성하고($G(x)$) 대상 이미지(Y)와 픽셀별로 비교한 MAE (Mean Absolute Error)를 감소시켜 두 이미지가 같게 되도록 하며 Equation (4)와 같이 나타낼 수 있다.

$$L_{L1} = E_{x,y}\ [\ //\ Y - G(x)\ //\ ] \quad (4)$$

5. 실험 및 결과

5.1 데이터 준비

구성요소 데이터를 준비하기 위하여 '네이버 나눔 손글씨'에서 77개 스타일을 내려 받았으며 그 중 60개를 학습에 사용하고 17개는 처음 보는 스타일로 남겨두어 새로운 글자를 생성할 때 사용하였다[10]. 14개 글자에서 얻은 24개 자/모음 샘플 이미지는 Fig. 9와 같으며, 이미지의 정확한 식별을 위하여 파일 이름의 첫번째 숫자는 폰트 스타일을 표시하고 뒤에 나오는 숫자는 글자의 십진수 유니코드를 사용하였다.

얻어진 24개 기본 자/모음으로 결합규칙을 적용하여 Model ***f***에 입력하는 이미지 샘플은 Fig. 10 예시와 같으며, 최종 생성된 27개 구성요소의 결과 이미지 샘플은 Fig. 11과 같다.

생성된 27개 구성요소와 기본 24개 구성요소를 합하여 초/중/종성으로 글자를 생성하는 학습을 위하여 유니코드 기반 상용 한글 2000자(한글 표준코드체계(KS X 1001))에 대하여 학습하였으며 NVIDIA GTX-2080Ti GPU 12GB-Memory Size를 사용하여 약 70시간이 소요(40,413,716 파라미터) 되었다.

Fig. 9. 24 Components Images

Fig. 10. Input Dataset Sample with Combining Rule Applied

Fig. 11. Generated 27 Components Images

Fig. 12. Unseen Style Results Sample

5.2 생성 결과

Fig. 12는 CKFont2 모델로 처음보는 손글씨체 17개 스타일에 대하여 글자를 생성한 결과 샘플이다. 미세조정을 비롯한 어떤 추가적인 학습 없이 생성된 고품질의 결과물이다.

Fig. 13은 모델별로 생성된 이미지 비교를 위하여 17개의 동일한 폰트 스타일로 생성된 이미지 샘플이다.

6. 평 가

모델 성능을 평가하기 위해 zi2zi, CKFont1, MX-Font 모델을 같은 폰트 스타일과 컨텐트를 이용하여 CKFont2 모델과 동일 조건으로 처음 보는 17 스타일로 256자를 생성하였다.

Fig. 13은 정성적으로 결과물의 품질을 비교할 수 있는 모델별 결과물의 비교 예시이다. 시각적으로는 CKFont1, CKFont2 및 MX-Font의 결과물이 유사하며 구체적인 비교를 위해서 정량적으로 비교하였다.

정량적인 비교를 위하여 모델별 L1 및 L2 손실, 유사도(Structural Similarity Index, SSIM)와 FID(Frechet Inception Distance) 점수[17]를 포함한 모든 산출된 데이터 값을 Table 4에 표시하였다. zi2zi와 CKFont1 모델은 미세조정이 필요한 모델이며 MX-Font 모델은 미세조정을 사용하지 않는 모델이나, CKFont2 모델은 모두 가능한 모델로서 미세조정 유무에 따른 손실 값을 모두 표시하였다.

Table 4에서 보는 바와 같이 L1, L2 손실 값은 미세조정 없이 생성된 CKFont2 모델(0.2644, 0.2652)이 MX-Font 모델(1.0153, 1.0135) 대비 크게 낮게 나타났으며, 유사도는 CKFont2 (0,8387) 모델이 MX-Font 모델(0.7122) 보다 높게 나타나 CKFont2 모델이 월등히 우수함을 보여주고 있다. Fig. 14와 Fig. 15는 CKFont2 모델과 MX-Font 모델의 L1, L2 손실비교와 유사도를 각각 비교하여 그래프로 나타낸 것이다(CKFont2 모델 : 실선, MX-Font 모델 : 점선).

또한 Table 4에서 보는 바와 같이 CKFont2 모델의 유사도(0.8387)는 CKFont1의 유사도(0.8562)에 비해 낮았다.

Fig. 13. Generated Output Comparison Sample by Different Model

CKFont2의 결과는 미세조정을 하지 않고 나온 결과임을 고려할 때 우수한 값이며, 미세조정을 했을 때 CKFont2 모델의 유사도(0.8562)는 CKFont1의 유사도 값과 같은 수준임을 알 수 있다.

Table 4. Values of Loss / SSIM / FID

INDEX		WITH finetuning			WITHOUT finetuning		
		L1 Loss	L2 Loss	SSIM	L1 Loss	L2 Loss	SSIM
Zi2zi (FID:127)	mean	0.2916	0.3081	0.8478			
	var	0.0005	0.0016	0.0052			
CKFont1 (FID : 71)	mean	0.2859	0.2814	0.8562			
	var	0.0004	0.0005	0.0006			
CKFont2 (FID : 31)	mean	0.2542	0.2540	0.8562	0.2644	0.2652	0.8387
	var	0.0003	0.0003	0.0006	0.0001	0.0001	0.0007
MXFont (FID : 36)	mean				1.0153	1.0135	0.7122
	var				0.0002	5.9743	0.0012

CKFont1 모델과 CKFont2 모델의 손실 값은 미세조정 유무에 상관없이 CKFont2 모델이 모두 낮게 나타나고 있다. 이는 CKFont2 모델의 성능이 CKFont1 모델보다 우수함을 보여주는 것으로 이를 L1, L2 Loss 비교 그래프로 Fig. 16에 나타내었다(CKFont1 모델 : 실선, CKFont2 모델 : 점선).

아울러 CKFont2 모델의 미세조정 유무에 따른 손실 값의 변화를 Fig. 17에 나타내었다. 미세조정을 하지 않은(점선) 결과가 예상대로 미세조정을 한 결과(실선) 보다 조금 높게 나타났지만, 그 차이는 감내할 만큼 미세하다. 이는 미세조정 없는 CKFont2 모델 성능의 우수함을 보여준다.

실제 이미지의 특징 벡터와 가짜 이미지의 특징 벡터 (생성기에서 생성) 사이의 거리를 계산하는 성능 지수인 FID 점수는 점수가 낮을수록 생성기에 의해 생성된 이미지의 품질이 실제와 더 비슷하고 더 우수하다는 것을 나타낸다. Fig. 18에서와 같이 CKFont2가 31점('★')으로 가장 우수하며, 그 다음은 MX-Font로 36점('•'), CKFont1 71점('⭘'), zi2zi 127점 ('/')순으로 나타났다. -

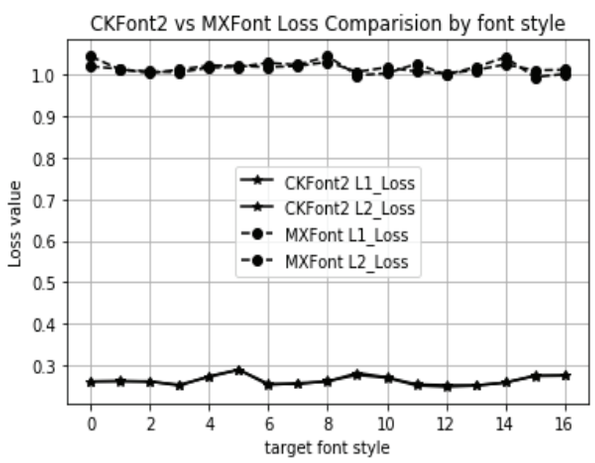

Fig. 14. CKFont2 vs MX-Font L1/L2 Loss Comparision by 17 Font Style

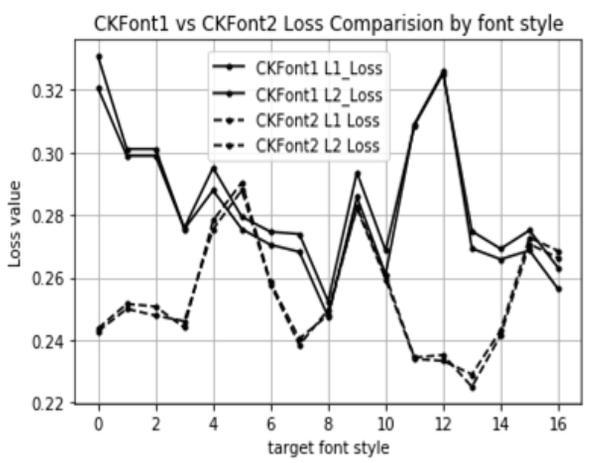

Fig. 16. CKFont1 vs CKFont2 Loss Comparision by 17 Font Style

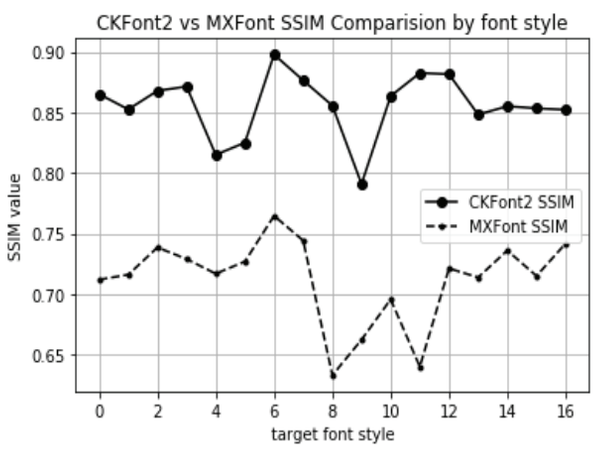

Fig. 15. CKFont2 vs MX-Font SSIM Comparision by 17 Font Style

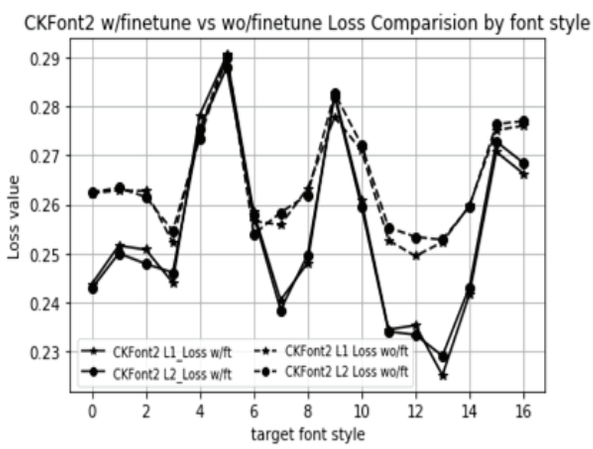

Fig. 17. CKFont2 Comparision with and Without Finetuning

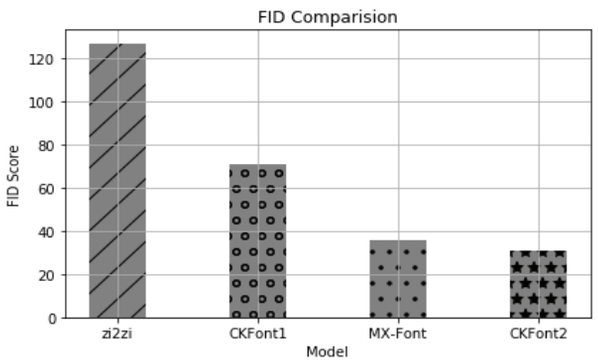

Fig. 18. FID Score Comparision by Models

Fig. 19. Example of Handwriting Hangul Characters Outputs

Fig. 20. Example of Chinese Characters Outputs (3components)

결론적으로 CKFont2 모델이 MX-Font 모델 보다 고품질의 이미지를 생성하였다. CKFont2 모델은 CKFont1 모델보다 더 적은 글자를 사용하고 finetuning이 필요치 않은 모델로 개선하여 성능이 대폭 개선되었음을 알 수 있다.

또한 CKFont2 모델도 CKFont1 모델처럼 한글 손글씨체는 물론 구성요소로 분리될 수 있는 한자와 태국어, 일본어 등에도 적용이 가능하다. Fig. 19는 한글 손글씨체를 생성한 예시이며, Fig. 20은 3개부수로 분리되는 한자에 적용한 결과이다.

한글의 결과물은 DM-Font, MX-Font 모델에서 제시한 결과물보다 고품질이며, 한자의 이미지는 RD-GAN[18]에서 구성요소를 이미지로 분리한 결과보다 우수한 품질임을 정성적인 평가로 알 수 있다.

7. 결 론

폰트 생성 모델은 다양한 폰트를 손쉽게 생성할 수 있는 방법을 제시하며, 보다 효과적인 모델 개발을 위해 많은 논문이 발표되었고 여전히 활발한 연구가 진행 중이다.

본 논문은 최소한의 글자로부터 폰트 고유의 스타일 정보를 추출하여 고품질의 전체 폰트를 생성하면서, 시간과 자원을 절약할 수 있는 간결한 모델을 생성하는 것이 연구의 최종 목표이다. 특별히 한글의 구조적 특성을 분석하고 한글의 조합성에 근거하여 24개의 기본 구성요소를 포함하는, 최소 14글자만으로 모든 나머지 글자를 생성할 수 있는 모델을 제안하였다.

CKFont2 모델은 CKFont1 모델을 개선한 것으로 계층을 깊이하고 Feature Loss를 사용하였으며 스타일 인코더를 사용하여 미세조정이 필요 없는, 시간과 자원이 절약되는 효율적인 모델이다. 아울러 생성된 결과를 zi2zi, CKFont1, MX-Font 모델의 결과와 비교 평가하여 CKFont2 모델의 우수한 성능을 확인하였다. 본 모델은 특별히 한글 생성에 유용하지만 한글의 손글씨는 물론 한자의 부수를 구성요소로 시험적용 하였을 때도 훌륭한 결과를 보여주었으며, 이는 구성요소를 이용한 다양한 변환 모델에도 적용 가능함을 의미한다. 향후 구성요소 개념을 적용하는 연구를 계속 진행할 계획이며 아울러 공개된 코드로 다양한 많은 시도가 있기를 기대한다.

References

[1] S. Weidman, “Deep learning from scratch,” O'Reilly Media, Inc. 2019.

[2] D. Foster, “Generative deep learning,” O'Reilly Media, Inc. 2020.

[3] Z. Huang and L. Li. “Dynamic design of text and exploration of layout space,” *7th International Conference on Arts, Design and Contemporary Education (ICADCE 2021)*, Atlantis Press, 2021.

[4] J. Cha, S. Chun, G. Lee, B. Lee, S. Kim, and H. Lee, “Few-shot compositional font generation with dual memory,” *ECCV (European Conference on Computer Vision)*, 2020.

[5] S. Park, S. Chun, J. Cha, B. Lee, and H. Shim. "Few-shot font generation with localized style representations and factorization," *AAAI (Association for the Advancement of Artificial Intelligence)*, 2021.
[6] S. Park, S. Chun, J. Cha, B. Lee, and H. Shim, "Multiple heads are better than one: Few- shot font generation with multiple localized experts," *ArXiv abs/2104.00887*, 2021.
[7] D. H. Ko, A. U. Hassan, J. Suk, and J. Choi, "SKFont: Skeleton-driven Korean font generator with conditional deep adversarial networks," *International Journal on Document Analysis and Recognition (IJDAR)*, Vol.24, No.4, pp.325-337, 2021.
[8] J. Park, A. U. Hassan, and J. Choi, "Few-Shot Korean font generation based on hangul composability," *KIPS Transactions on Software and Data Engineering*, Vol.10, No.11, pp.473-482, 2021.
[9] Unicode [Internet], https://namu.wiki/w/unicode.
[10] Naver Nanum Fonts [Internet], https://Hangul.naver.com/2011/font.
[11] Ian Goodfellow et al., "Generative adversarial networks," In *Advances in Neural Information Processing Systems*, *ArXiv Preprint arXiv: 1406.2661*, 2014.
[12] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
[13] Rewrite [Internet], https://github.com/kaonashi-tyc/Rewrite.
[14] Y. Tian, "zi2zi: Master chinese calligraphy with conditional adversarial networks," [Internet] https:// github.com/kaonashi-tyc/zi2zi, Mar. 22. 2021.
[15] Y. Jiang, Z. Lian, Y. Tang, and J. Xiao, "DCFont: An end-to-end deep chinese font generation system," SIGGRAPH Asia 2017, Technical Briefs, 2017.
[16] Y. Jiang, Z. Lian, Y. Tang, and J. Xiao, "SCFont: Structure guided Chinese font generation via deep stacked networks," 2019.
[17] G. Parmar, R. Zhang, and J. Y. Zhu, "On buggy resizing libraries and surprising subtleties in FID calculation," 2021, [Internet] https://github.com/ bioinfjku/TTUR, Sep. 2021.
[18] Y. Huang, M. He, L. Jin, and Y. Wang, "RD-GAN: few/zero-shot Chinese character style transfer via radical decomposition and rendering," *European Conference on Computer Vision*, Springer, Cham, 2020.

박 장 경

https://orcid.org/0000-0001-6446-4590
e-mail : xjnpark@soongsil.ac.kr
1981년 공군사관학교 항공공학(학사)
1987년 미국 해군대학원 OR/SA(석사)
2019~현 재 숭실대학교 컴퓨터학부 박사과정
관심분야 : 딥러닝, 자동 폰트 생성, 최적화

Ammar Ul Hassan

https://orcid.org/0000-0001-6744-507X
e-mail : ammar.instantsoft@gmail.com
2013년 International Islamic University Islamabad, Pakistan 컴퓨터공학부(학사)
2018년 숭실대학교 컴퓨터공학과(석사)
2018년~현 재 숭실대학교 컴퓨터학부 박사과정
관심분야 : 딥러닝, 자동 폰트 생성, 폰트 데이터셋

최 재 영

https://orcid.org/0000-0002-7321-9682
e-mail : choi@ssu.ac.kr
1984년 서울대학교 제어계측공학과(학사)
1986년 미국 남가주대학교 전기공학과 (컴퓨터공학)(석사)
1991년 미국 코넬대학교 전기공학부 (컴퓨터공학)(박사)
1992년~1994년 미국 국립오크리지연구소 연구원
1994년~1995년 미국 테네시 주립대학교 연구교수
1995년~현 재 숭실대학교 컴퓨터학부 교수
관심분야 : 디지털 타이포그라피, 시스템소프트웨어, 병렬/분산처리, 고성능컴퓨팅