文字種とスタイルを考慮した文字生成による書家俵越山の模倣

桑田　若菜† 　三林　亮太†† 　谷　雅徳†††,†††† 　大島　裕明†,††

† 兵庫県立大学社会情報科学部　〒 651–2197 兵庫県神戸市西区学園西町 8–2–1
†† 兵庫県立大学大学院情報科学研究科　〒 651–2197 兵庫県神戸市西区学園西町 8–2–1
††† 関西大学社会学部　〒 564–8680 大阪府吹田市山手町 3–3–35
†††† アカデミックビジョン　〒 606–8203 京都府京都市左京区田中関田町
E-mail: †fa19f028@guh.u-hyogo.ac.jp, ††threeforest8@gmail.com, †††ohshima@ai.u-hyogo.ac.jp

あらまし　本研究では深層学習を用いて，書家俵越山の作品を模倣した文字の生成と分析を行う．文字は，同じ人が同じ文字を書いたとしても，書く文章やタイミングによって微妙なゆれが生じる．書家俵越山の文字にもその傾向があり，実際に俵越山作品を見てみると同じ文字に対してバリエーションのある書き方をしていることがわかる．そこで本研究では，文字のゆれを考慮した文字生成を行うために，文字の文字種とスタイルを抽出してベクトルで表現し，任意の組み合わせで再構築して生成を行うモデルを提案する．本研究では文字の画像をグリフと呼ぶ．今回扱う生成モデルでは 2 つのグリフを入力とする．1 つ目のグリフからはスタイルを抽出し，2 つ目のグリフからは文字種を抽出する．それぞれのグリフから抽出したスタイルと文字種を組み合わせたグリフを出力する問題に取り組む．文字種とスタイルを抽出してベクトルで表現することで，スタイルをゆらすことや，文字種をゆらすことが可能になり，同じ文字に対して様々なバリエーションをもった文字の生成が可能であると考えた．実際に今回提案したモデルにより学習した生成器を用いて，ゆれの表現に取り組んだ．そして，ベクトルの操作や入力グリフの変更によって，生成グリフのゆれの表現が可能であることを示した．

キーワード　文字画像生成，書道作品，深層学習

1 はじめに

近年，深層学習を用いて芸術作品を生成する技術が発展している．例えば，17 世紀オランダの画家であるレンブラントをコンピュータで再現する「The Next Rembrandt [1]」というプロジェクトがある．このプロジェクトは，コンピュータに過去のレンブラントの作品を学習させ，レンブラントの作風でありながらも，レンブラントが書いたことがない新たな作品を生み出すプロジェクトである．他にも，膨大なデータで事前学習された，入力されたテキストから画像を生成する Text to Image モデルの「Stable Diffusion [2]」が 2022 年にオープンソースとして公開された．このモデル以前にも Text to Image モデルは存在していたが，オープンソースとしての公開はされていなかったため，一部の人しか利用することができなかった．しかし，Stable Diffusion が公開されて以降，このモデルを用いた画像生成サービスが多くリリースされた．それにより，深層学習モデルの学習を行う環境を構築することが難しい一般の人であっても，各サービスを介して手軽に深層学習による画像生成を行うことが容易となった．

深層学習による画像生成が身近なものとなっていく中で，深層学習を用いて芸術作品を生成するという取り組みは，今後ますます発展していくと考えられる．

図 1　書家俵越山が書いた様々な「あ」

そこで，本研究では深層学習による芸術作品を生成する研究の一環として，書家俵越山 [3]を模倣した書道作品の生成に取り組む．書道作品は，毛筆と墨を用いて紙に文字を書くことで制作される．毛筆は，力の入れ方によって線に強弱をつけることができる．線の強弱や筆の運び方によって，さまざまな表現を行うことができ，その書家ならではの文字の書体や，書風を表現することができる．そして，同じ人が同じ文字を書いたとしても，書く文章やタイミングによって微妙なゆれが生じる．例えば，図 1 のように書家俵越山の文字を見ると，同じ文字であっても様々な書き方をしていることがわかる．

そこで本研究では，文字のゆれを考慮した文字生成を行うために，文字の文字種とスタイルを抽出し，任意の組み合わせで再構築して生成を行うモデルを提案する．文字種とスタイルを抽出してベクトルで表現することで，スタイルをゆらすことや，文字種をゆらすことが可能になり，同じ文字であっても様々なバリエーションをもった文字の生成が可能であると考えた．

1：https://www.nextrembrandt.com
2：https://github.com/CompVis/stable-diffusion
3：http://etsuzan.jp，https://twitter.com/etsuzan_tawara

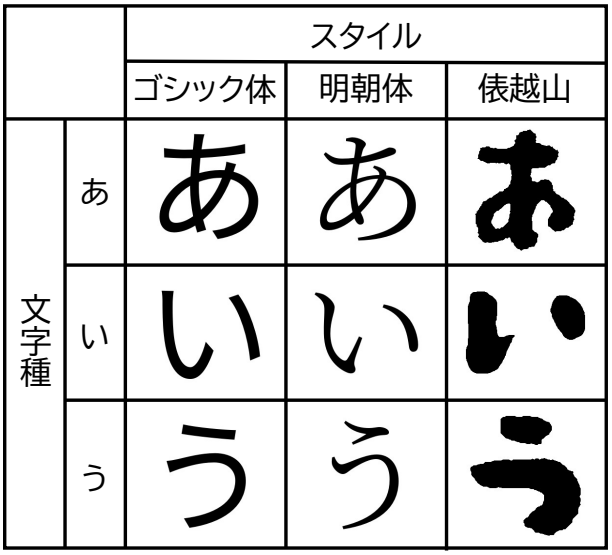

図 2　文字種とスタイルの各組み合わせにおけるグリフの例

2　用語の定義

本節では本論文で使用する用語を定義する．

まず，文字の種類を**文字種**と呼ぶこととする．具体的には「あ」や「い」といったクラス分類のことを指す．手書きの文字と印刷された文字では，同じ文字であったとしても，線の太さや角度，とめやはらいなど，さまざまな部分で形が違う．しかし，これらの文字が持つ骨組みはある程度共通しており，多少文字の形が変わっていても同じ文字であることを認識できる．

次に，文字種を実際に具現化する際の一定のデザインのことを**スタイル**と呼ぶこととする．具体的にはゴシック体や明朝体などが挙げられる．何らかの文字種に対し，どう具現化して骨組みに肉付けするかという体系的な情報がスタイルには含まれる．その中に文字種，つまり文字の骨組みの情報は含まれない．例えば，ゴシック体の「あ」とゴシック体の「い」は同じスタイルである．しかし，文字種は違うものである．

そして，文字種にスタイルを合わせ，実際にイメージとして具現化されたものを**グリフ**と呼ぶこととする．スタイルがゴシック体で文字種が「あ」であれば，図 2 の左上のようなグリフとなる．文字種もスタイルも抽象的な概念であり，ある文字種が具現化され表記された際には，何らかのスタイルを有することになる．文字種もスタイルもそれ単体では具現化することができず，文字種とスタイルが合わさりグリフになることで初めて具現化される．本研究ではこのグリフの生成を行う．

3　関 連 研 究

3.1　Image-to-Image translation

Image-to-Image translation（I2I）とは，ソースドメインとターゲットドメインの間のマッピング関数を学習するタスクである．ソース画像の内容を保持しながら，同時にターゲットドメインのスタイルを融合させる．

I2I 手法の一つである pix2pix [4] では，GAN [2] をベースに構築され，画像の内容を維持したままスタイルのみを変換した学習を行うことができる．また，pix2pix では U-Net [12] が用いられており，エンコーダ側の特徴マップをデコーダ側の特徴マップに結合するスキップ接続が行われている．

I2I 手法の一つである MUNIT [3] では，Content Encoder と Style Encoder を用いて，入力画像からコンテンツの特徴量とスタイルの特徴量をそれぞれ抽出する．FUNIT [8] も同様に入力画像からコンテンツの特徴量とスタイルの特徴量をそれぞれ抽出する．FUNIT では同じクラスから複数枚の画像をそれぞれ特徴量抽出し，その結果を平均したものをスタイルの特徴量としている．

本研究ではコンテンツを文字種とし，文字種エンコーダとスタイルエンコーダを用いて文字種とスタイルの特徴量をそれぞれ抽出する．

3.2　GAN を用いたフォント生成

フォント生成の分野では，フォントをデザインする作業をいかにして自動化するかという問題に取り組まれてきた．この問題に対して GAN という画像生成技術を利用したフォント生成の研究が多く行われている．

初めて中国語フォント生成に GAN を用いた手法として，Yuchen が提案した zi2zi [15] がある．zi2zi は pix2pix [4] ベースのフォント生成モデルである．zi2zi では，pix2pix をベースに，ACGAN [10] と DTN [13] の仕組みを取り入れることで，文字種とスタイルを抽出するような学習を行う．zi2zi ではスタイルの学習に埋め込み表現を用いている．

DCFont [5] では，zi2zi の構成をもとに，フォント特徴再構成ネットワークを追加している．フォント特徴再構成ネットワークでは未見の文字の特徴を推定し，Generator を介して与えられた内容の文字を合成する．SCFont [6] では，まず入力文字の骨格をターゲット文字の骨格に変換し，その骨格に対し GAN モデルによってスタイルを付与する．ZiGAN [16] では，ペアではない多数のグリフを利用し，異なる書体の文字の特徴をヒルベルト空間に写像し，特徴分布を整列させることで，文字種情報を保持するとともにスタイル変換を行っている．SE-GAN [18] では筆文字フォントに焦点を当てたフォント生成を行っている．一般的なフォントは，基本的な構造や配置に共通点があるが，筆文字フォントは大きく歪んでいることが多い．

本論文では zi2zi の構成をもとに，文字種だけでなくスタイルもエンコーディングできるようなモデルを作成した．

3.3　Few-shot フォント生成

Few-shot フォント生成タスクでは，非常に少ない数のスタイルを参照するだけで新しいフォントライブラリを生成することを目的としている．Few-shot フォント生成では，与えられたグリフから文字種とスタイルを切り離すことがよく行われている．文字種とスタイルの抽出の方法として，グローバル特徴表現とコンポーネントに基づいた特徴表現の 2 種類に分けることができる．

グローバル特徴表現では，文字種グリフとスタイルグリフか

ら，それぞれ文字種とスタイルに関連するベクトルを抽出する．EMD [20] や DG-Font [17] では，文字種ベクトルとスタイルベクトルを抽出し，2 つのベクトルを合わせることで新たなグリフを合成する．MLFont [1] では，文字種グリフ 1 枚を文字種エンコーダに入力して得られた特徴量と，スタイルグリフ複数枚をスタイルエンコーダに入力して得られた特徴量を合わせてデコーダに入力する．

コンポーネントに基づいた特徴表現では，文字をコンポーネント単位やストローク単位で表現することがよく行われている．コンポーネントとは，例えば「叶」という字であれば「口」と「十」で分けることができる．この「口」と「十」をそれぞれコンポーネントと呼ぶ．ストロークとは，文字を書く際の 1 画 1 画のことである．FsFont [14] では，参照スタイル文字からコンポーネント単位でスタイルを抽出し，文字種を参照スタイルで構成している．StrokeGAN [19] では，中国語の漢字で用いられるストロークは全部で 32 種類あるとし，任意の漢字に対して何画目がどのストロークかを表すストロークエンコーディングを行っている．CG-GAN [7] では，コンポーネント単位でスタイルと文字種を切り離すように生成器に成約をかけている．LF-Font [11] ではコンポーネント単位のスタイル表現を学習する．XMP-Font [9] では，ストロークレベルで文字を分割することで，入力をストロークラベルにすることができ，入力フォントに縛られない生成を行う事ができる．

本研究ではグローバル特徴表現を用いて，文字種グリフとスタイルグリフから，それぞれ文字種とスタイルに関連するベクトルを抽出する．

4 書家俵越山を模倣した文字生成における問題定義と使用するデータ

本節では，本研究における問題定義と使用するデータについて述べる．

4.1 問題定義

文字種集合 C は n 文字の文字種 $c_i (i = 1, 2, 3, \cdots, n)$ で構成されるものとする．

$$C = \{c_1, c_2, c_3, \cdots, c_n\} \tag{1}$$

例えば,「あ」と「い」の 2 種類の文字種で構成される文字種集合 $C = \{c_1, c_2\}$ であれば，c_1 は「あ」，c_2 は「い」となる．文字種集合 C のうち，ある一つの文字種を $c_\bullet$ と表すこととする．今回は 6,527 種類の文字種を使用する．

スタイル集合 S は m 種類のスタイル $s_j (j = 1, 2, 3, \cdots, m)$ で構成されるものとする．

$$S = \{s_1, s_2, s_3, \cdots, s_m\} \tag{2}$$

例えば，ゴシック体と明朝体の 2 種類のスタイルで構成されるスタイル集合 $S = \{s_1, s_2\}$ であれば，s_1 はゴシック体，s_2 は明朝体となる．スタイル集合 S のうち，ある一つのスタイルを $s_\bullet$ と表すこととする．今回は 20 種類のスタイルを使用する．

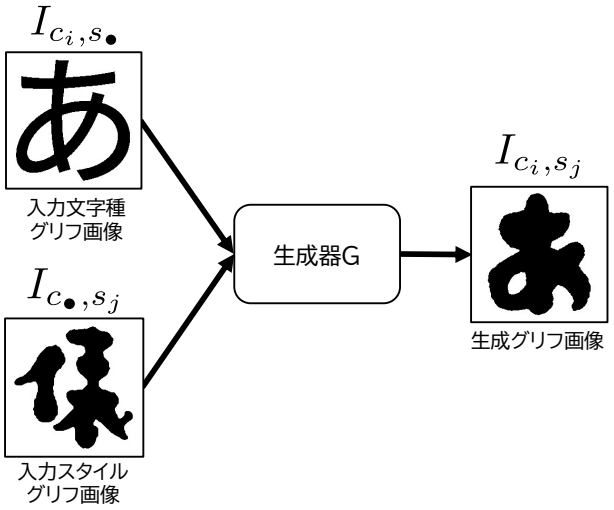

図 3 入 出 力

文字種 c_i とスタイル s_j を持つグリフ画像を I_{c_i,s_j} と表すこととする．同じ文字種 c_i を持ち，異なるスタイル $s_1, s_2, s_3, \cdots, s_m$ をそれぞれ持つグリフの集合を R_{c_i} とする．

$$R_{c_i} = \{I_{c_i,s_1}, I_{c_i,s_2}, I_{c_i,s_3}, \cdots, I_{c_i,s_m}\} \tag{3}$$

同じスタイル s_j を持ち，文字種集合 C の文字種をそれぞれ持つグリフの集合を R_{s_j} とする．

$$R_{s_j} = \{I_{c_1,s_j}, I_{c_2,s_j}, I_{c_3,s_j}, \cdots, I_{c_n,s_j}\} \tag{4}$$

本研究では，文字種 c_i と任意のスタイル $s_\bullet$ を持つグリフ画像 $I_{c_i,s_\bullet}$ と，スタイル s_i と任意の文字種 $c_\bullet$ を持つグリフ画像 $I_{c_\bullet,s_j}$ を入力としたときに，文字種 c_i とスタイル s_j を持つグリフ画像 I_{c_i,s_j} を出力する生成器 $\mathbf{G}$ を作る問題に取り組む．

$$I_{c_i,s_j} = \mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j}) \tag{5}$$

入力文字種グリフ画像 $I_{c_i,s_\bullet}$，入力スタイルグリフ画像 $I_{c_\bullet,s_j}$，出力グリフ画像 I_{c_i,s_j} は全て 256×256 ピクセルの 2 値化画像とする．入出力のイメージを図 3 に示す．例えば，文字種「あ」を持ったグリフ画像と，スタイル「俵越山」を持ったグリフ画像を入力すると，文字種「あ」とスタイル「俵越山」を持つグリフ画像が出力されるということを行う．

4.2 俵越山文字データ

本研究を行うにあたり本人から許諾を得た上で，書家俵越山の書道作品を利用した．書家俵越山が自身の Twitter アカウント [4] に投稿した作品から取得した 100 件，書家俵越山から直接提供を受けた 6 件，書家俵越山が手掛けた向日町警察署カレンダーから 13 件の合計 119 件の作品を利用した．図 4 に作品の一例を示す．これら 119 件の作品に対して 1 文字ずつ，最小のバウンディングボックスで切り抜き，合計 1,358 枚のグリフ画像を取得した．そして，長辺が 236 ピクセルになるよう拡大縮小し，長辺に対しては両端に 10 ピクセルずつ余白を取り，

4：https://twitter.com/etsuzan_tawara

図 4　俵越山作品

短辺に対しては 256 ピクセルになるように余白を取ることで，256 × 256 ピクセルの画像にした．その後，グレースケール化を行った上で各ピクセルの値が 128 未満であれば 0，128 以上であれば 1 になるよう 2 値化を行った．

1,358 枚のグリフ画像うち，向日町警察署カレンダーから取得したグリフ画像の中で，向日町警察署カレンダー以外のグリフ画像に含まれていない文字種を持つグリフ画像 72 枚を評価用に使用するため省き，残りの 1,286 枚を学習用データとした．また，評価用に省いた 72 枚から文字種の重複をなくした 54 枚を評価用データとし，残りの 18 枚は学習にも評価にも使用しないこととした．

4.3　日本語フォントデータ

Google Fonts [5] にて取得できる日本語フォントから，19 フォントを取得した．使用するフォントを表 1 に示す．フォントの太さは全て Regular とした．19 種類の各フォントについて，ひらがな，カタカナ，JIS 第一水準漢字，JIS 第二水準漢字の計 6,523 文字を使用した．こちらも俵越山文字データと同様に，1 文字に対して最小のバウンディングボックスで切り抜き，長辺が 236 ピクセルになるよう拡大縮小した．そして，長辺に対しては両端に 10 ピクセルずつ余白を取り，短辺に対しては 256 ピクセルになるように余白を取ることで，256 × 256 ピクセルの画像にした．その後，グレースケール化を行った上で各ピクセルの値が 128 未満であれば 0，128 以上であれば 1 になるよう 2 値化を行った．

4.4　俵越山文字データと日本語フォントデータからなるデータセット

俵越山文字データの学習用データ 1,286 枚と，日本語フォントデータから 19 スタイル各 6,523 枚からそれぞれランダムに 1,286 枚ずつ選んだ 24,434 枚を合わせた，25,720 枚のグリフ画像をターゲットグリフとする．

25,720 枚のグリフ画像は全てターゲットグリフ画像として用いるとともに，それぞれのグリフ画像 I_{c_i,s_j} が持つ文字種 c_i とスタイル s_j に応じて，入力文字種グリフ集合 R_{c_i} と入力スタイルグリフ集合 R_{s_j} にも追加する．

各ターゲットグリフ画像 1 枚につき，入力文字種グリフ画像と入力スタイルグリフ画像の 2 枚の入力画像を用いる．入力文字種グリフ画像は，ターゲットグリフ画像 I_{c_i,s_j} と同じ文字種 c_i の入力文字種グリフ集合 R_{c_i} からランダムに 1 枚選んだグリフ画像 $I_{c_i,s_\bullet}$ を用いる．入力スタイルグリフ画像は，ターゲットグリフ I_{c_i,s_j} と同じスタイル s_j の入力スタイルグリフ集合 R_{s_j} からランダムに 1 枚選んだグリフ画像 $I_{c_\bullet,s_j}$ を用いる．各ターゲットグリフに対して，入力文字種グリフ画像 $I_{c_i,s_\bullet}$，入力スタイルグリフ画像 $I_{c_\bullet,s_j}$，ターゲットグリフ画像 I_{c_i,s_j} の

表 1　使用したフォント一覧

フォント名	サンプル
DelaGothicOne-Regular	あアa亜永
DotGothic16-Regular	あアa亜永
HachiMaruPop-Regular	あアa亜永
HinaMincho-Regular	あアa亜永
KaiseiDecol-Regular	あアa亜永
KleeOne-Regular	あアa亜永
MochiyPopOne-Regular	あアa亜永
NewTegomin-Regular	あアa亜永
PottaOne-Regular	あアa亜永
ReggaeOne-Regular	あアa亜永
RocknRollOne-Regular	あアa亜永
ShipporiMincho-Regular	あアa亜永
SourceHanSansJP-Regular	あアa亜永
SourceHanSerifJP-Regular	あアa亜永
Stick-Regular	あアa亜永
YujiSyuku-Regular	あアa亜永
YuseiMagic-Regular	あアa亜永
ZenKurenaido-Regular	あアa亜永
ZenMaruGothic-Regular	あアa亜永

5：https://fonts.google.com

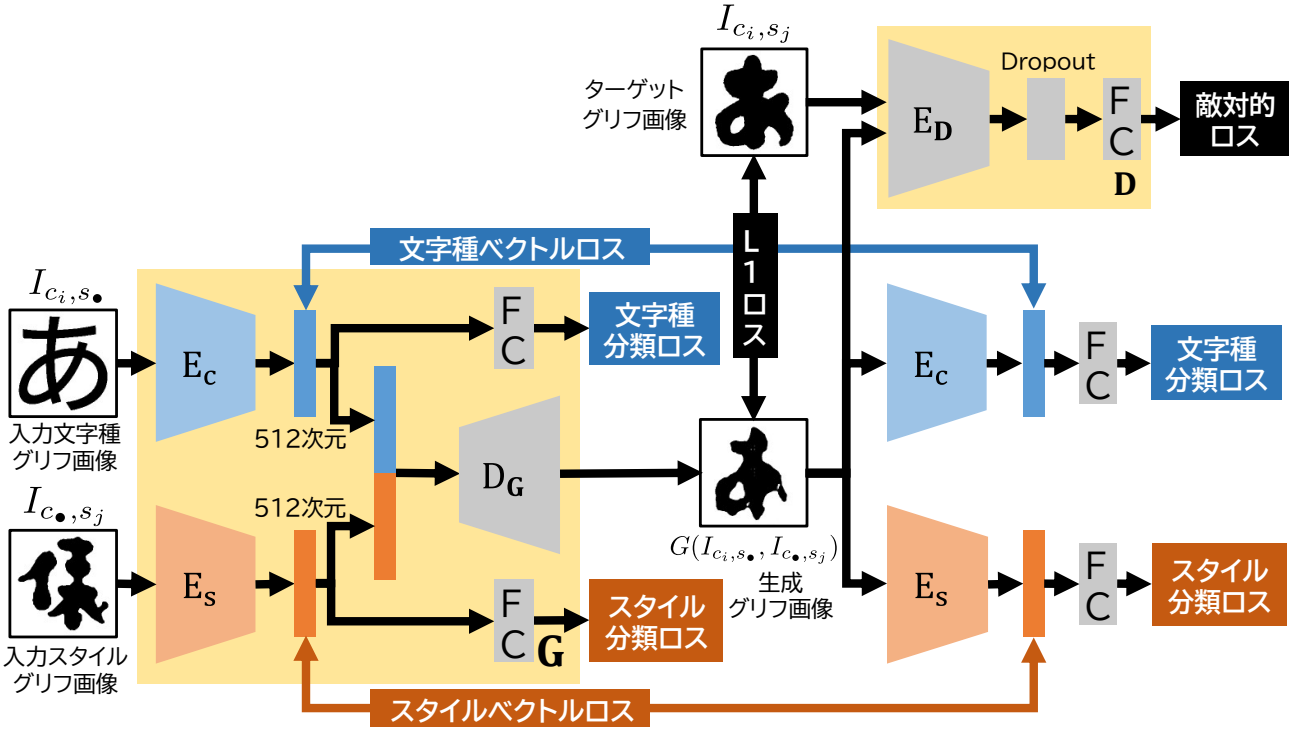

図 5　モデルの構成図

3 枚を 1 組とするデータセットを作り，9:1 の割合でランダムに訓練データと検証データに分割し，学習に用いた．

5　文字種とスタイルを考慮したグリフ生成モデル

5.1　モデルの概要

今回提案するモデルは，生成器 **G** と識別器 **D** で構成する．モデル構成を図 5 に示す．

生成器 **G** は，文字種エンコーダ $\mathrm{E_c}$，スタイルエンコーダ $\mathrm{E_s}$，デコーダ $\mathrm{D_G}$，2 つの FC 層からなる．識別器 **D** は，エンコーダ $\mathrm{E_D}$ と FC 層からなる．

生成器 **G** では，生成したい文字種 c_i を持つグリフ画像 $I_{c_i,s_\bullet}$ と，生成したいスタイル s_j を持つグリフ画像 $I_{c_\bullet,s_j}$ を，生成器 **G** に入力すると，文字種 c_i とスタイル s_j を持つグリフ画像 I_{c_i,s_j} が生成されることを目的とする．

識別器 **D** は，生成器 **G** が生成したグリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ と，ターゲットグリフ画像 I_{c_i,s_j} を分類することを目的とする．

5.2　生　成　器

生成したい文字種 c_i を持つグリフ画像 $I_{c_i,s_\bullet}$ と，生成したいスタイル s_j を持つグリフ画像 $I_{c_\bullet,s_j}$ が，生成器 **G** に与えられたとき，生成される画像は文字種 c_i とスタイル s_j を保持している必要がある．そこで，文字種エンコーダ $\mathrm{E_c}$ を用いて文字種グリフ画像 $I_{c_i,s_\bullet}$ から文字種ベクトル $\mathrm{E_c}(I_{c_i,s_\bullet})$ を抽出し，スタイルエンコーダ $\mathrm{E_s}$ を用いてスタイルグリフ画像 $I_{c_\bullet,s_j}$ からスタイルベクトル $\mathrm{E_s}(I_{c_\bullet,s_j})$ を抽出する．

抽出した文字種ベクトル $\mathrm{E_c}(I_{c_i,s_\bullet})$ を FC 層に入力し，文字種数 n と同じ次元数の n 次元ベクトルを出力する．同様にスタイルベクトル $\mathrm{E_s}(I_{c_\bullet,s_j})$ を FC 層に入力し，スタイル数 m と同じ次元数の m 次元ベクトルを出力する．

そして，文字種ベクトル $\mathrm{E_c}(I_{c_i,s_\bullet})$ とスタイルベクトル $\mathrm{E_s}(I_{c_\bullet,s_j})$ を，結合してデコーダ $\mathrm{D_G}$ に入力する．デコーダ $\mathrm{D_G}$ から文字種 c_i とスタイル s_j を持つグリフ画像 I_{c_i,s_j} が生成されるような学習を行う．

今回，生成器 **G** には U-Net を用いる．文字種エンコーダ $\mathrm{E_c}$ とスタイルエンコーダ $\mathrm{E_s}$ はそれぞれ 8 層の畳み込み層で構成する．各層の間には，活性化関数 Leaky ReLU を用いる．各層の出力に対し，ミニバッチごとに正規化を行う．文字種エンコーダ $\mathrm{E_c}$ とスタイルエンコーダ $\mathrm{E_s}$ は，それぞれ 256×256 ピクセルの画像を入力として受け取り，512 次元ベクトルを出力する．出力したベクトルは，2 つの FC 層とデコーダ $\mathrm{D_G}$ に入力する．FC 層には線形層を用いる．文字種エンコーダ $\mathrm{E_c}$ から出力された 512 次元ベクトルは FC 層に入力され，6,523 次元ベクトルを出力する．スタイルエンコーダ $\mathrm{E_c}$ から出力された 512 次元ベクトルは FC 層に入力され，19 次元ベクトルを出力する．2 つの 512 次元ベクトルを結合した 1,024 次元ベクトルをデコーダ $\mathrm{D_G}$ に入力する．

デコーダ $\mathrm{D_G}$ は 8 層の逆畳み込み層で構成する．各層の前には，活性化関数 ReLU を用いる．各層の出力に対し，ミニバッチごとに正規化を行う．文字種エンコーダ $\mathrm{E_c}$ の各層での出力を，デコーダ $\mathrm{D_G}$ の対応する層に渡し，前の層からの出力と結合する．デコーダ $\mathrm{D_G}$ は 256×256 ピクセルの画像を出力する．

5.3　識　別　器

識別器 **D** では，生成器 **G** から出力されたグリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ またはターゲットグリフ画像 I_{c_i,s_j} を入力とする．入力された画像が生成器 **G** から出力されたグリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ であれば 0，ターゲットグリフ画像 I_{c_i,s_j} であ

れば 1 が出力されることを目的とする．

今回，識別器 $\mathbf{D}$ のエンコーダ $\mathrm{E}_{\mathbf{D}}$ は 4 層の畳み込み層で構成する．各層の前には活性化関数 Leaky ReLU を用いる．各層の出力に対し，ミニバッチごとに正規化を行う．Dropout 層では Dropout 率を 0.5 に設定した．FC 層には線形層を用いる．エンコーダ $\mathrm{E}_{\mathbf{D}}$ からの出力を Dropout 層に入力する．Dropout 層からの出力を FC 層に入力し，1 次元ベクトルを出力する．

5.4 学　　習

入力文字種グリフ画像 $I_{c_i,s_\bullet}$ と入力スタイルグリフ画像 $I_{c_\bullet,s_j}$ から，ターゲットグリフ画像 I_{c_i,s_j} が生成されるようにモデルの学習を行う．入力文字種グリフ画像 $I_{c_i,s_\bullet}$ は，ターゲットグリフ画像 I_{c_i,s_j} と同じ文字種 c_i を持つグリフである必要がある．入力スタイルグリフ画像 $I_{c_\bullet,s_j}$ は，ターゲットグリフ画像 I_{c_i,s_j} と同じスタイル s_j を持つグリフである必要がある．

ピクセルレベルでの学習には，生成器 $\mathbf{G}$ から生成されたグリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ とターゲットグリフ画像 I_{c_i,s_j} との間の平均絶対誤差を用いる．

$$\mathcal{L}_{L1} = \|I_{c_i,s_j} - \mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})\|_1 \tag{6}$$

生成グリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ を文字種エンコーダ $\mathrm{E_c}$ とスタイルエンコーダ $\mathrm{E_s}$ にそれぞれ入力する．入力文字種グリフ画像 $I_{c_i,s_\bullet}$ を文字種エンコーダ $\mathrm{E_c}$ に入力して得られた文字種ベクトル $\mathrm{E_c}(I_{c_i,s_\bullet})$ と，生成グリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ を文字種エンコーダ $\mathrm{E_c}$ に入力して得られた文字種ベクトル $\mathrm{E_c}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j}))$ から，平均二乗誤差によって文字種ベクトルロスを計算する．

$$\mathcal{L}_{Cconst} = \|\mathrm{E_c}(I_{c_i,s_\bullet}) - \mathrm{E_c}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j}))\|_2^2 \tag{7}$$

スタイルについても文字種と同様のことを行い，入力スタイルグリフ画像 $I_{c_\bullet,s_j}$ をスタイルエンコーダ $\mathrm{E_s}$ に入力して得られたスタイルベクトル $\mathrm{E_s}(I_{c_\bullet,s_j})$ と，生成グリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ をスタイルエンコーダ $\mathrm{E_s}$ に入力して得られたスタイルベクトル $\mathrm{E_s}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j}))$ から，平均二乗誤差によってスタイルベクトルロスを計算する．

$$\mathcal{L}_{Sconst} = \|\mathrm{E_s}(I_{c_\bullet,s_j}) - \mathrm{E_s}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j}))\|_2^2 \tag{8}$$

入力文字種グリフ画像 $I_{c_i,s_\bullet}$ を文字種エンコーダ $\mathrm{E_c}$ に入力して得られた文字種ベクトル $\mathrm{E_c}(I_{c_i,s_\bullet})$ を，文字種数 n と同じ n 次元のベクトルを出力する全結合層 FC_n に入力する．出力したベクトルに softmax 関数を適用したものと，正解ラベル c_i を n 次元で One-Hot エンコーディングしたベクトル t_{c_i} から，Cross Entropy loss を用いて入力文字種分類ロスを計算する．以下，損失関数内において Cross Entropy loss を CE と表記する．

$$\mathcal{L}_{C_{input}} = \mathrm{CE}(\mathrm{softmax}(\mathrm{FC}_n(\mathrm{E_c}(I_{c_i,s_\bullet}))), t_{c_i}) \tag{9}$$

入力スタイルグリフ画像 $I_{c_\bullet,s_j}$ をスタイルエンコーダ $\mathrm{E_s}$ に入力して得られたスタイルベクトル $\mathrm{E_s}(I_{c_\bullet,s_j})$ を，スタイル数 m と同じ m 次元のベクトルを出力する全結合層 FC_m に入力する．出力したベクトルに softmax 関数を適用したものと，正解ラベル s_j を m 次元で One-Hot エンコーディングしたベクトル t_{s_j} から，Cross Entropy loss を用いて入力スタイル分類ロスを計算する．

$$\mathcal{L}_{S_{input}} = \mathrm{CE}(\mathrm{softmax}(\mathrm{FC}_m(\mathrm{E_s}(I_{c_\bullet,s_j}))), t_{s_j}) \tag{10}$$

生成グリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ を文字種エンコーダ $\mathrm{E_c}$ に入力して得られた文字種ベクトル $\mathrm{E_c}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j}))$ を，文字種数 n と同じ n 次元のベクトルを出力する全結合層 FC_n に入力する．出力したベクトルに softmax 関数を適用したものと，正解ラベル c_i を n 次元で One-Hot エンコーディングしたベクトル t_{c_i} から，Cross Entropy loss を用いて入力文字種分類ロスを計算する．

$$\mathcal{L}_{C_{fake}} = \mathrm{CE}(\mathrm{softmax}(\mathrm{FC}_n(\mathrm{E_c}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})))), t_{c_i}) \tag{11}$$

生成グリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ をスタイルエンコーダ $\mathrm{E_s}$ に入力して得られたスタイルベクトル $\mathrm{E_s}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j}))$ を，スタイル数 m と同じ m 次元のベクトルを出力する全結合層 FC_m に入力する．出力したベクトルに softmax 関数を適用したものと，正解ラベル s_j を m 次元で One-Hot エンコーディングしたベクトル t_{s_j} から，Cross Entropy loss を用いて入力スタイル分類ロスを計算する．

$$\mathcal{L}_{S_{fake}} = \mathrm{CE}(\mathrm{softmax}(\mathrm{FC}_m(\mathrm{E_s}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})))), t_{s_j}) \tag{12}$$

生成器 $\mathbf{G}$ から生成されたグリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ のみを識別機 D に入力する．生成グリフ画像かターゲットグリフ画像かの 2 値分類を行い，Binary Cross Entropy loss を用いて生成器敵対的ロスを計算する．生成グリフ画像が 1 となるように学習を行う．以下，損失関数内において Binary Cross Entropy loss を BCE と表記する．

$$\mathcal{L}_{G_{GAN}} = \mathrm{BCE}(\mathrm{sigmoid}(\mathbf{D}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j}))), 1) \tag{13}$$

上記の損失関数を組み合わせた損失関数 $\mathcal{L}_G$ を用いて生成器 $\mathbf{G}$ の学習を行う．

$$\begin{aligned}\mathcal{L}_G = \mathcal{L}_{L1} + \mathcal{L}_{Cconst} + \mathcal{L}_{Sconst} + \mathcal{L}_{C_{input}} + \mathcal{L}_{S_{input}} \\ + \mathcal{L}_{C_{fake}} + \mathcal{L}_{S_{fake}} + \mathcal{L}_{G_{GAN}}\end{aligned} \tag{14}$$

生成器 $\mathbf{G}$ から生成されたグリフ画像 $\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j})$ と，ターゲットグリフ画像 I_{c_i,s_j} を識別機 D に入力する．生成グリフ画像かターゲットグリフ画像かの 2 値分類を行い，Binary Cross Entropy loss（BCE）を用いて識別器敵対的ロスを計算する．生成グリフ画像であれば 0，ターゲットグリフ画像であれば 1 になるように識別器 $\mathbf{D}$ の学習を行う．

$$\begin{aligned}\mathcal{L}_D = \mathrm{BCE}(\mathrm{sigmoid}(\mathbf{D}(I_{c_i,s_j})), 1) \\ + \mathrm{BCE}(\mathrm{sigmoid}(\mathbf{D}(\mathbf{G}(I_{c_i,s_\bullet}, I_{c_\bullet,s_j}))), 0)\end{aligned} \tag{15}$$

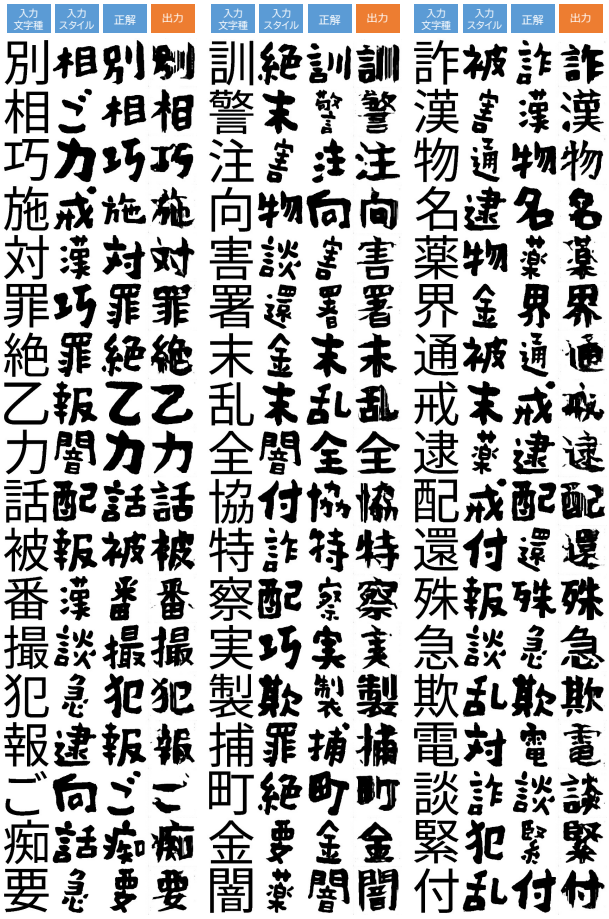

図 6　生成結果

6　実　　験

6.1　生成結果

評価用データ 54 枚をターゲットグリフとした．入力文字種グリフには，ターゲットグリフと同じ文字種を持つグリフを SourceHanSansJP-Regular から用いた．入力スタイルグリフには，評価データからターゲットグリフを抜いた 53 枚のうちランダムに 1 枚を用いた．生成結果を図 6 にて示す．

6.2　評　　価

評価用データ 54 枚を正解データとし，zi2zi 及び提案モデルの推論を行った．zi2zi が生成したグリフ画像 54 枚と提案モデルが生成したグリフ画像 54 枚の合計 108 枚を，どちらの手法で生成したかわからないようにシャッフルした．生成したグリフ画像 1 枚ずつに対して，2 つの指標にて評価を行った．1 つ目は「生成画像が俵越山文字をどのくらい模倣できているか」である．書家俵越山本人が，1 が「できていない」，5 が「できている」として 5 段階で評価を行った．2 つ目は「生成画像が文字として成立しているか」である．こちらは本人ではない人が，1 が「成立していない」，5 が「成立している」として 5 段階で評価を行った．評価の際は，どちらも参考として正解データを見せた．

1 つ目の「生成画像が俵越山文字をどのくらい模倣できているか」に対する評価結果を図 7 に示す．zi2zi が平均 3.59，提案手法が平均 2.17 であった．

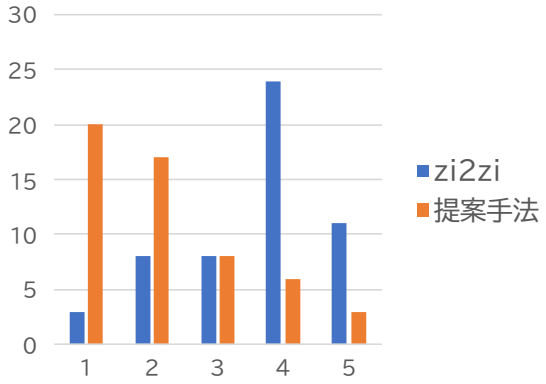

図 7　生成画像が俵越山文字をどのくらい模倣できているか

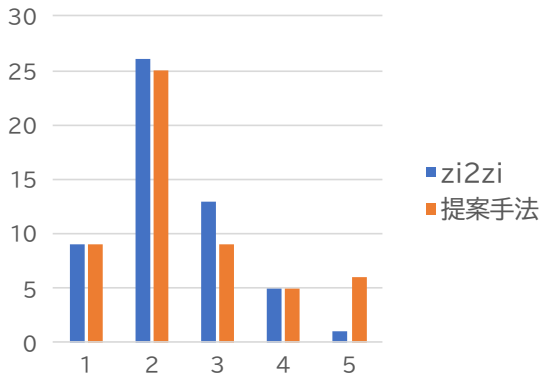

図 8　生成画像が文字として成立しているか

2 つ目の「生成画像が文字として成立しているか」に対する評価結果を図 8 に示す．zi2zi が平均 2.31，提案手法が平均 2.51 であった．

6.3　学習済み生成器を用いたゆれの表現

学習済み生成器を用いて，グリフのゆれの表現を行った．

まず，抽出したベクトルを用いてグリフのゆれを表現した．俵越山文字とゴシック体フォントの SourceHanSansJP-Regular から文字種「相」を持つグリフを用いて，文字種エンコーダ E_c とスタイルエンコーダ E_s にそれぞれ入力し，文字種ベクトルとスタイルベクトルを抽出した．俵越山のスタイルベクトルと，ゴシック体のスタイルベクトルを，割合を変えて足し合わせ，デコーダ D_G に入力し，グリフ生成を行った．この際，文字種ベクトルには俵越山の文字種ベクトルを用いた．生成結果を図 9 に示す．俵越山からゴシック体へ，スタイルベクトルを 0.25 ずつ割合を変えながら生成を行った．少しずつ線の太さが変わっていき，まっすぐな線に変化していることがわかる．文字種ベクトルだけでなく，スタイルベクトルも他のベクトルと混ぜ合わせることで，グリフを少しずつゆらすことができた．

次に，入力グリフ自体を変更することによってグリフのゆれを表現した．入力スタイルグリフは固定し，入力文字種グリフのスタイルを変えて生成を行った結果と，入力文字種グリフは固定し，入力スタイルグリフの文字種を変えて生成を行った結果を図 10 に示す．入力グリフの組み合わせによっても，グリ

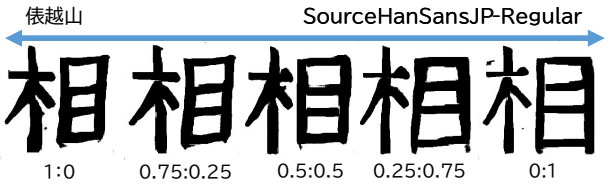

図 9　ベクトルを用いたゆれの表現

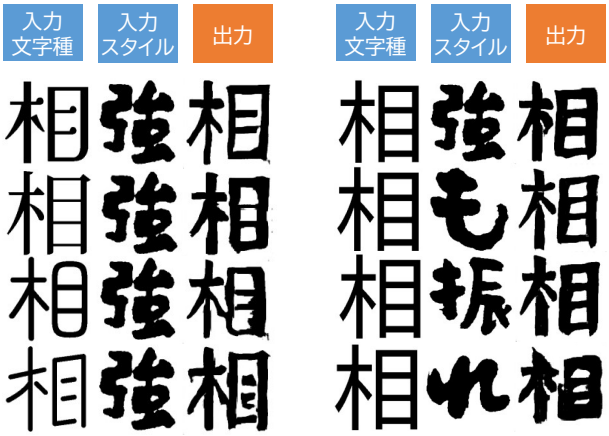

図 10　グリフの変更によるゆれの表現

フをゆらすことができた．

7　まとめと今後の課題

本論文では，書家俵越山の文字を生成するために，文字種とスタイルを抽出する手法を提案した．文字は書く文章やタイミングによって微妙なゆれが生じるものであり，実際に書家俵越山の文字にもその傾向がある．そこで本研究では，文字のゆれを考慮した文字生成を行うために，文字の文字種とスタイルを抽出してベクトルで表現し，再構築して生成を行うモデルを提案した．そして今回学習した生成器を用いて，グリフのゆれの表現が可能であることを示した．

課題としては，他のスタイルと比較すると俵越山スタイルの生成が安定していないという点や，学習したスタイルに引っ張られて違うスタイルが生成されてしまうという点がある．これらは学習データの少なさが影響していると考えられるため，今後は学習データの少ないスタイルや未知のスタイルであっても，対応できるような構造を検討したいと考えている．

謝　　辞

本研究は JSPS 科研費 JP21H03775，JP22H03905 の助成を受けたものです．ここに記して謝意を表します．

文　　献

[1] Xu Chen, Lei Wu, Minggang He, Lei Meng, and Xiangxu Meng. MLFont: Few-shot chinese font generation via deep meta-learning. In *Proceedings of ICMR'21*, pp. 37–45, 2021.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of NIPS'14*, pp. 2672–2680, 2014.

[3] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *Proceedings of ECCV'18*, pp. 179–196, 2018.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of CVPR'17*, pp. 1125–1134, 2017.

[5] Yue Jiang, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. DCFont: An end-to-end deep chinese font generation system. In *Proceedings of SIGGRAPH'17*, pp. 1–4, 2017.

[6] Yue Jiang, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. SCFont: Structure-guided chinese font generation via deep stacked networks. In *Proceedings of AAAI'19*, pp. 4015–4022, 2019.

[7] Yuxin Kong, Canjie Luo, Weihong Ma, Qiyuan Zhu, Shenggao Zhu, Nicholas Yuan, and Lianwen Jin. Look closer to supervise better: One-shot font generation via component-based discriminator. In *Proceedings of CVPR'22*, pp. 13482–13491, 2022.

[8] Ming-Yu Liu, Xun Huang, Arun Mallya, Tero Karras, Timo Aila, Jaakko Lehtinen, and Jan Kautz. Few-shot unsupervised image-to-image translation. In *Proceedings of ICCV'19*, pp. 10551–10560, 2019.

[9] Wei Liu, Fangyue Liu, Fei Ding, Qian He, and Zili Yi. XMP-Font: Self-supervised cross-modality pre-training for few-shot font generation. In *Proceedings of CVPR'22*, pp. 7905–7914, 2022.

[10] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In *Proceedings of ICML'17*, pp. 2642–2651, 2017.

[11] Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. Few-shot font generation with localized style representations and factorization. In *Proceedings of AAAI'21*, pp. 2393–2402, 2021.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of MICCAI'15*, pp. 234–241, 2015.

[13] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. In *Proceedings of ICLR'17*, pp. 1–15, 2017.

[14] Licheng Tang, Yiyang Cai, Jiaming Liu, Zhibin Hong, Mingming Gong, Minhu Fan, Junyu Han, Jingtuo Liu, Errui Ding, and Jingdong Wang. Few-shot font generation by learning fine-grained local styles. In *Proceedings of CVPR'22*, pp. 7895–7904, 2022.

[15] Yuchen Tian. zi2zi: Master chinese calligraphy with conditional adversarial networks. `https://github.com/kaonashi-tyc/zi2zi`, 2017. 2022 年 12 月 15 日閲覧.

[16] Qi Wen, Shuang Li, Bingfeng Han, and Yi Yuan. ZiGAN: Fine-grained chinese calligraphy font generation via a few-shot style transfer approach. In *Proceedings of MM'21*, pp. 621–629, 2021.

[17] Yangchen Xie, Xinyuan Chen, Li Sun, and Yue Lu. DG-Font: Deformable generative networks for unsupervised font generation. In *Proceedings of CVPR'21*, pp. 5126–5136, 2021.

[18] Shaozu Yuan, Ruixue Liu, Meng Chen, Baoyang Chen, Zhijie Qiu, and Xiaodong He. SE-GAN: Skeleton enhanced gan-based model for brush handwriting font generation. In *Proceedings of ICME'22*, pp. 1–6, 2022.

[19] Jinshan Zeng, Qi Chen, Yunxin Liu, Mingwen Wang, and Yuan Yao. StrokeGAN: Reducing mode collapse in chinese font generation via stroke encoding. In *Proceedings of AAAI'21*, pp. 3270–3277, 2021.

[20] Yexun Zhang, Ya Zhang, and Wenbin Cai. Separating style and content for generalized style transfer. In *Proceedings of CVPR'18*, pp. 8447–8455, 2018.