Vol.2019-CG-173 No.9
2019/3/5

生成モデルと印象評価を用いた文字フォント推薦システム

藤田 裕樹[1,a)]　謝 浩然[1,b)]　宮田 一乘[1,c)]

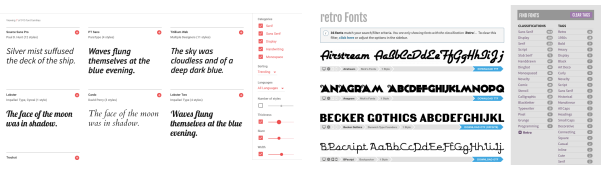
概要：現在世の中には多くの文字フォントが存在しており，それらの多くを利用することができるが，現状として所望のフォントを取得することは困難である．本研究ではこのような問題を解決するために，画像から得られる特徴量と人が文字画像から感じる印象とを結びつけることを試みた．画像特徴量を取得するために生成モデルの一つである Variational Autoencoder を使用し，その多次元潜在空間における印象の分布データを取得するために，生成された文字フォント画像からどのような印象を受けるのかというアンケート調査を行なった．以上の調査で得られた分布データに対して，Manifold Learning を用いて人が操作しやすい二次元まで次元削減を行ない，その二次元上の分布密度を各印象ごとにヒートマップ として表示した．そのヒートマップ上をマウスで移動することで，対応する文字フォント画像を連続的に変更しながら表示するユーザーインターフェースを開発した．

Generative Model Based Font Recommendation System Using Perception Classification

Abstract: Recently, there are numerous character fonts on the Internet for public usage. However, it is still difficult and challenging to generate the fonts meeting the user preferences. To solve this issue, we propose the font recommendation system to visualize the perceptual adjustment in the latent space of a generative model. In this paper, we adopt the variational autoencoder (VAE) network for font generation. Then, we conduct a perceptual study on the generated fonts from the multi-dimensional latent space of the generative model. After we obtained the distribution data of specific preferences, we utilize a manifold learning approach to visualize the font distribution. As a case study of our proposed method, we develop a user interface for generated font selection in the designated user preference using the representation of heat map.

1. Introduction

現在世の中には多くの文字フォントが存在している．特にアルファベットに関しては，それらの多くが無料でネット上に公開されており，我々は多くのフォントを使用することができる．フォントは主にプレゼンテーション資料や，広告のポスターなどの作成に使用される．Google Fonts[1] や Font Squirrel[2] は，代表的な文字フォント提供サイトである．しかしながら，現状としてそれらのサイトから所望のフォントを検索することは困難である．**図 1** にそれらのサイトの UI を示す．



Google Fonts　　Font Squirrel

図 1　Example of Web Sites Providing Character Fonts

Google Fonts では “Thickness”，“Slant”，“Width” のパラメータを操作することによってフォント検索を可能にしているが，決して使い勝手が良いとは言えない．なぜならば人がフォントから受ける印象やイメージというものは，それらの画像特徴量のように単純に要素分解できるものではないからである．例えばお祭りのポスターを制作しており，“楽しそう”なフォントが欲しいとする．このときどのような画像特徴量を操作すれば良いのだろうか．この解を

[1] 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa 923–1292, Japan
a) yfujita@jaist.ac.jp
b) xie@jaist.ac.jp
c) miyata@jaist.ac.jp

導くことは非常に困難である．

一方で Font Squirrel では各フォントに対してクラス分けを行うと同時に，多くのタグによるグループ分けが行われている．しかしながらこのサイトに関しても，フォントの分類がタグを付ける人の独断に依るものであるという点，タグの種類が多くてイメージを持ちにくいという点，さらにそれらが連続的な存在ではないため，少しずつ変化をつけることが不可能であるという点などにおいて問題がある．

本研究ではこのような問題を解決するために，フォントの画像特徴量と人がフォント画像から感じる印象とを結びつけることを試みた．画像特徴量を取得するために生成モデルの潜在空間を使用する．本研究における生成モデルを使用する利点として，潜在空間が連続的な空間であり，似た特徴を持つ画像が空間内の近い位置に分布するという性質が挙げられる．この性質を利用して，人がフォント画像からどのような印象を持つかについての分布を考える．

まず我々が作成した文字フォント画像データセットを用いて生成モデルを学習させる．学習の結果として得られた多次元空間の潜在変数を変更し，その変数に対する出力画像を表示するユーザーインターフェースを開発した．

文字フォント画像の印象調査を行うため，提案したユーザーインターフェースを用いてアンケート調査を実施した．本研究では表示された出力画像が “POP”, “Formal”, “Casual” の三種類のいずれかに当てはまると感じた時，その潜在変数データを保存するという形式で調査を実施した．この三種類はなるべくイメージを持ちやすく，なおかつそれぞれが異なる場面で必要とされることを考えた上で決定した．

以上の調査で得られた多次元の潜在空間上の分布データに対して Manifold learning を用い，ユーザーが操作しやすい二次元まで次元削減を行なった．その二次元上の点の分布密度を，印象調査を行なった三種類の各分類に対するヒートマップ として表示し，そのヒートマップ上をマウスで移動することで，対応する文字フォント画像を連続的に変更しながら表示するユーザーインターフェースを作成した．以上のように本研究では生成モデルを用いて，画像特徴量と人が文字画像から感じる印象とを結びつけたフォント推薦システムを提案する．

2. Related Works

2.1 Generative Model and Character Fonts

Paul ら [3] は生成モデルによって作成された潜在空間を用いて，入力された一つの文字フォント画像と似たスタイルを持つ文字フォントセットを作成している．文字フォントセットとは，大文字の “A”から “Z”，小文字の “a”から “z”，数字の “0”から “9”を含む 62 種類の画像を指す．他にも Samaneh ら [4] の発表した Multi-Content GAN では，“Glyph Network”と “Ornamentation Network” という生成モデルを用いた二つのネットワークによる End-to-End のシステムによって，少ない入力文字フォント画像からそれらの特徴を持つような 26 個の大文字アルファベット画像を生成する．この研究の特徴として，二つの役割が異なるネットワークを使用することによって，文字フォントの形だけでなく，その入力フォント画像が持つ色や装飾の特徴に対しても再現を行うという点がある．

2.2 Character Fonts and Perception

文字フォントと人が受ける印象との関係性を考慮したものとして，Choi ら [5] の研究がある．この研究では入力された画像に対して，その画像から受けるイメージと一致するような文字フォントを提示する．warm-cool と hard-soft という二つの軸を持つ二次元意味空間において，あらかじめ多数の文字フォントがマッピングされており，入力画像がその空間上においてどの文字フォントと対応するか示すものとなっている．

2.3 Distribution of Character Fonts

Campbell ら [6] は文字フォントの分布を二次元の Manifold で表現した．ユーザーはマウス操作によってこの二次元上を移動することが可能であり，連続的にフォントを変形させることができる．Guo ら [7] は漢字の文字フォントに対して，それらの分布を二次元の Manifold で表現した．文字の形や骨格の特徴量を使用しながら生成モデルを学習させることによって，入力に用いられた既存の文字と異なる新しいフォントを生成することが可能になっている．

2.4 The Position of This Paper

本研究では，これまでに紹介した論文を参考にした上で，それぞれの要素を踏まえた論文という立ち位置を取りたいと考えている．

2.1 節の研究は，生成モデルと文字フォントを組み合わせたものであるが，文字画像を生成するという点に注目しており，人の画像から受ける印象に関しては一切考慮していない．そこで本研究では生成モデルの潜在空間における印象の分布を考えることによってこの点に対応する．

2.2 節の研究は，文字フォントと印象評価との関係性に注目しているが，意味空間へのマッピングにおいてアンケート調査による側面が強く，文字画像としての特徴量を取得することをあまり重視していない．そこで本研究では潜在空間へのマッピングに生成モデルを用いることによって，直接的な文字画像の特徴量を考慮する．

2.3 節の研究は，文字フォントの分布を表示し，その分布からフォント画像を生成・検索することが可能であるが，これらの研究も画像から受ける印象を一切考慮していない．そこで本研究ではこれらの分布表示に加えて，印象調

査を組み合わせることを考える．

3. Preliminary Research

本章では予備実験として，生成モデルの潜在空間内における生成画像の印象分布データを取得する．まず生成モデルを学習させるためのデータセットを作成し，本手法で用いる生成モデルの説明を行う．そしてデータ取得のために実施したアンケート調査のユーザーインターフェースを紹介し，実際の調査における流れと結果を述べる．

3.1 Dataset

印象調査の実験を行うためにデータセットを作成する．本研究では Google Fonts から全てのフォントデータに対して一括ダウンロードを行った．全部で 2244 個の ttf ファイル（TrueType Font）を取得できたが，その内 75 個はデータに問題があったため除外した．こうして取得した合計 2169 個のデータによるデータセットを用いて実験を進めていく．次に機械学習を使用するため，これらのデータ整備を行う．

データ整備の目標としては有名な手書き数字データセット “MNIST”[8] 同様に，28*28 ピクセルの正方形画像を作成することとする．本研究では画像から得られる特徴量を考えるが，画像にフォント以外の余白部分が存在するとそれ自体に意味を持ってしまうため，なるべく全てのデータに対して同等の手段をもって余白消去，センタリングを行う．

まずは先ほど作成されたデータセットの ttf ファイルをグレースケールの png 画像ファイルに変換する．最終的に 28*28 ピクセルに戻す予定ではあるが，最初は十分大きいサイズとして 256*256 ピクセルで作成した．

次に文字のセンタリングを行うために，文字の範囲のみを取得した長方形画像を生成する．アルゴリズムとしては，画像に対して一番端から順番に行と列を走査し，各ピクセルごとに全ての画素値が 255，すなわち白色である行と列を削除することによって実現する．ここで上下左右の四方向の端から一つずつ走査するが，その方向で一度全てが白色であるわけではない行（列）を見つけた時点で，その方向からの走査を終了する．このアルゴリズムによって “i”などといった，間に空白の列が含まれてしまう文字に対して誤作動を起こさないようにしている．

そしてこのようにして得られた長方形の画像をうまく正方形へと変形する．長方形の短い方の辺に対して，左右交互に白色のみの行（列）を追加することによって実現する．正方形画像を生成することが完了したら，その画像にバイリニア補間を用いて 28*28 ピクセルに変換する．

図 2 にこのデータ整備の流れを示す．**図 3** は生成された “A”のデータセットの一部である．

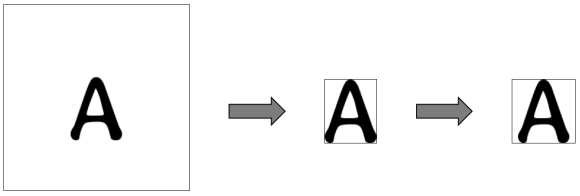


図 2 Data Cleansing Process

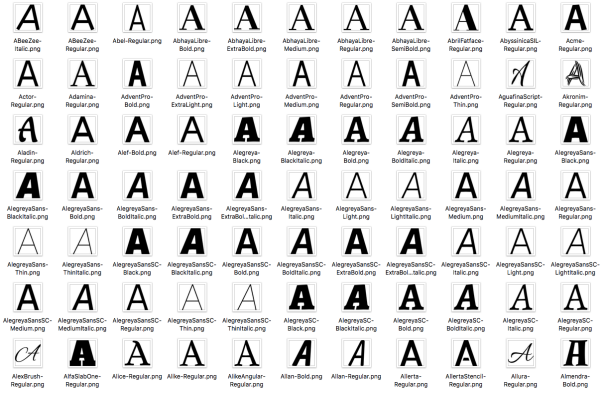


図 3 Font Examples

3.2 VAE Model

本研究では生成モデルの一つである Variational Autoencoder（VAE）[9], [10] を使用した．**図 4** に使用した VAE のモデルを示す．このモデルは Python のライブラリである Keras[11] を用いて作成した．括弧で示された数字はそれらのデータ配列のサイズを示し，Input から Output の方向へと計算は進む．それぞれの色分けされた層に関しても以下，簡単な説明を行う．

Convolutional 2 次元の畳み込み層
Flatten 入力に対して平滑化を行い，1 次元に変形
Dense 全結合層
Lambda 指定された計算を実行，ここでは平均ベクトルと分散ベクトルの計算

実行環境は以下の通りである．

PC ubuntu 16.04 LTS
CPU Intel Core i7-7700 CPU @ 3.60GHz * 8
GPU GeForce GTX 1060 3GB/PCIe/SSE2
Language Python 3.5
CUDA CUDA 8.0
CuDNN CuDNN 6.0
Library Keras-gpu 2.1.6, numpy

学習時のエポック数は 50 であり，潜在変数は 5 次元としている．次元数は大きい方が質の高い潜在空間を生成することが可能となるが，印象調査のデータ分布の取得が困難になる．さらに文字画像の特徴量として重要な要素となるものが，“線の太さ”，“線の丸さ”，“全体の横幅”，“斜め具合”，“かすれ具合” という 5 種類程度であると仮定した

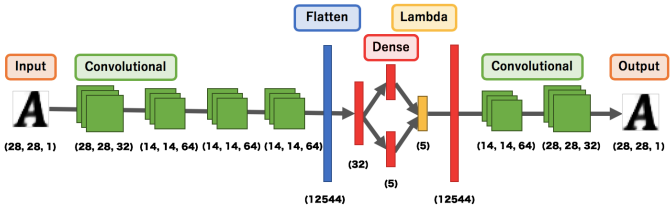


図 4　Variational Autoencoder Model

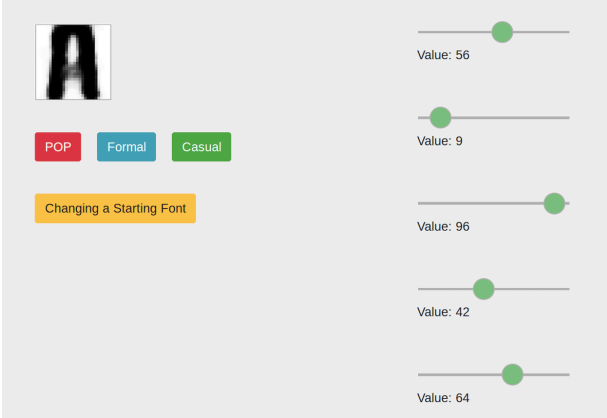


図 5　User interface of Survey Form

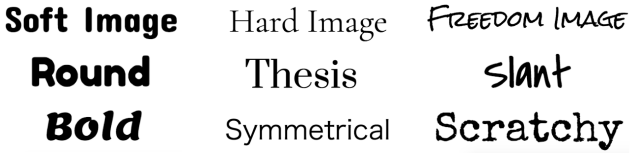
上で次元数を決定した．

3.3　User Interface of Perception Survey

生成された VAE モデルにおける潜在空間と，ユーザーが画像から受ける印象の情報とを結びつけるためにアンケート調査を行う．この調査では実験参加者が自由に潜在空間内を移動することが可能であり，なおかつリアルタイムでその潜在空間中の座標における生成画像を表示するようなアプリケーションを使用する．**図 5** にアンケートサイトの具体的なユーザーインターフェースを示す．

右半分に存在する五つのスライダーは，学習させた VAE モデルにおける潜在変数の値を示す．実験参加者はこのスライダーを操作することによって，潜在変数を操作することが可能となる．五次元の各潜在変数を 0 から 99 までの値で操作できるようにしている．これは VAE が正規分布に基づくことから，パーセント点関数を用いて 5%から 95%までの範囲について 100 等分することで対応させている．

そしてその潜在変数を VAE によってデコードした結果が左上に表示されている画像である．このようにして実験参加者は多くのフォント画像を生成することが可能となる．左側にある “POP”, “Formal”, “Casual” の三つのボタンは，それぞれフォントに対するクラスタリングを行うためのものである．それらのボタンを押すことによって，その時表示されている画像の潜在変数の値を保存する．さらにその下にある “Changing a Starting Font” ボタンは，五つの潜在変数をランダムに変更するものである．ランダムな数値の取得に関しては JavaScript の “Math.random” 関数を使用している．



（1）POP　（2）Formal　（3）Casual

図 6　Feature Examples

表 1　Details of Results

POP	Formal	Casual	Total
273	311	298	884

3.4　Survey Flow

今回のアンケート調査では表示されている文字画像を “POP”, “Formal”, “Casual” の三種類に分類することを目的としている．調査を始める前にこれらの特徴の説明として**図 6** を提示した．このような特徴を示した上で，実験参加者にはまずランダムに文字を表示する “Changing a Starting Font” ボタンによって文字画像を生成してもらい，それが “POP”, “Formal”, “Casual” のいずれかに分類できると感じた場合，そのボタンをクリックしてもらうことによって特徴量を保存する．そしてそのようにして特徴量の保存が完了したか，または三種類いずれにも分類できないと感じた場合，もう一度 “Changing a Starting Font” ボタンをクリックして次の文字画像を表示してもらう．この作業を 5 分間続けてもらうことによってデータを収集した．

大学院の学生 17 名に対してこのアンケート調査を行い，合計 884 個の特徴点を取得した．取得した点の種類の内訳を**表 1** に示す．

4.　Experiments

4.1　t-SNE

アンケート調査によって五次元の潜在空間内における各特徴の分布を取得することができた．本研究では最終的にユーザーに提示するアプリケーションを作成したいので，これを二次元まで次元削減する．現在次元削減の手法は様々なものが存在するが，本研究では複数の手法を試した結果，最も特徴ごとの分類に成功した次元削減手法の “t-SNE” を採用する．

t-SNE（t-distributed Stochastic Neighbor Embedding）[12] とは manifold learning の一種であり，高次元のデータに対して二次元または三次元まで次元削減を行う手法である．また SNE（Stochastic Neighbor Embedding）[13] の改良版でもある．結果を**図 7** に示す．プロットの色に関しては，紫: POP, 緑: Formal, 黄: Casual をそれぞれ表している．

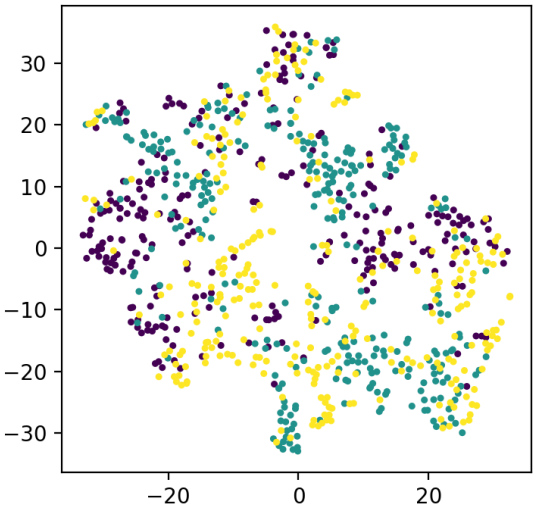


図 7　The Result of Dimensionality Reduction

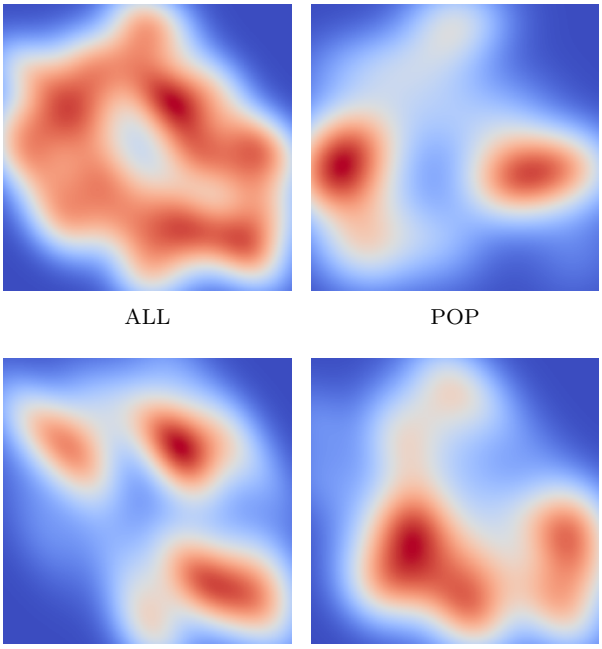


図 8　Distribution Heat Map

4.2　Kernel Density Estimation

二次元上における三種類のデータの分布を取得したが，それらの分布を可視化することを考える．本研究ではカーネル密度推定 [14] を使用し，全ての点，POP のみ，Formal のみ，Casual のみの四種類についての分布を取得した．python ライブラリの “Scipy[15]” を用いてカーネル密度推定を実行し，“matplotlib[16]” を用いて，**図 8** のようにヒートマップを作成した．密度が高い方を濃い赤色，低い方を濃い青色として表示している．

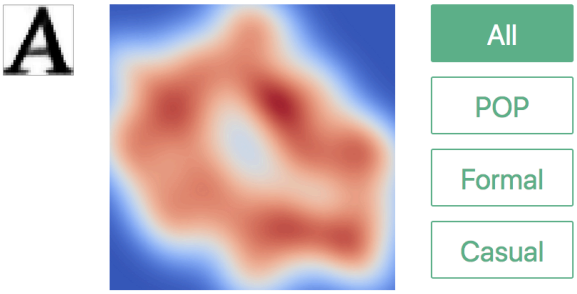


図 9　User Interface of Application

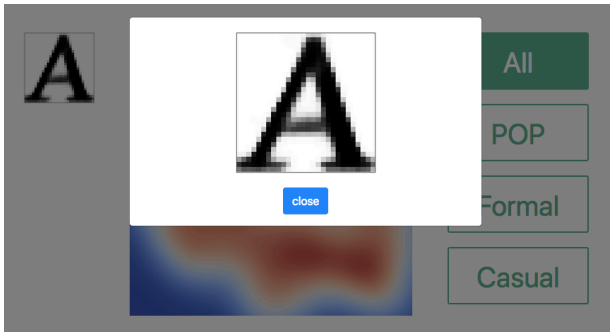


図 10　On click

4.3　User Interface

最終的なアプリケーションの UI は**図 9** のようになる．中央のヒートマップ上をマウスで移動させると，t-SNE を用いて得られた二次元空間において，そのマウス座標の最近傍に対応する文字画像を左上に表示する．

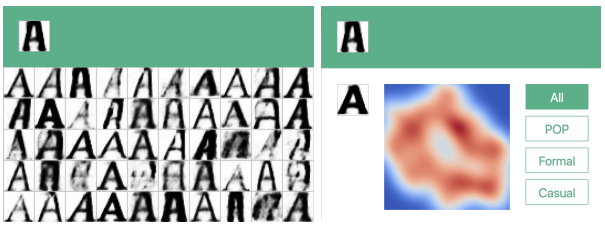
右側の四つのボタンをクリックすることによって，図 8 に示されている四種類のヒートマップに変更することが出来る．ヒートマップを変更することによって表示される文字画像が変わるわけではない．しかしながらその特徴を持つ文字画像がどの辺りに多く分布しているのか把握することが可能となる．

中央のヒートマップ上を移動し，所望の文字画像を見つけた場所でマウスをクリックすると，**図 10** のようにモーダルウィンドウが表示され，その文字画像を確認，保存することができる．

5.　Evaluation

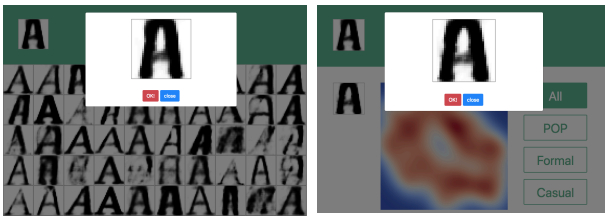
5.1　Overview

本研究のアプリケーションの有用性の確認のために評価実験を行なった．実験手法としては**図 11** に示すような二種類の UI を用いて，左上に表示されている文字フォント画像を探してもらう．図 11 における UI1 は，提案手法を用いた UI2 によって生成することが可能な 1592 種類の文字フォント画像すべてを表示しており，100*100 ピクセルの画像を 10 列，160 行並べることによって実現している．そしてこれらはともに Web アプリケーションとして実装



UI 1 UI 2

図 11 Evaluation UI



UI 1 UI 2

図 12 Selecting a Font

図 13 Test Dataset

されており，Web ブラウザをスクロールさせることによって，全て確認することが可能となる．

実験参加者は左上に提示されたものを見つけた場合，UI1 ではその画像を，UI2 ではそのマウス座標でクリックする．すると**図 12** のように，それぞれモーダルウィンドウによる確認画面が表示される．ここで赤色の “OK” ボタンをクリックすると，その画像が保存され，左上に提示された画像が次のものに変更される．

この評価実験では左上に提示する画像として，**図 13** のような 10 個の文字フォント画像によるデータセットを使用する．データセットはなるべく様々な種類を含むことを考慮した上で，無作為に 10 個の文字フォント画像を選定した．UI2 におけるデータセットのおおよその分布を**図 14** に示す．

この評価実験を通して，実験参加者が選択した画像の正確性と見つけるまでに要した時間を取得する．左上に提示されたデータセットの画像と選択された画像の正確性に関しては，Structural Similarity（SSIM）[17] を使用する．この計算は設定された局所領域ごとに行われ，得られた全体の値を平均したものが画像間の SSIM 値となる．そして SSIM 値は 0 から 1 の間で表現され，1 に近ければ近いほど類似度が高いとみなすことができる．

図 14 Test Dataset Distribution

表 2 SSIM Scores of UI1

Participant	1	2	3	4	5
Total Score	8.29	8.03	8.65	7.63	7.31
Participant	6	7	8	9	10
Total Score	7.67	7.89	8.04	8.47	7.76
Participant	11	12	13	14	15
Total Score	8.89	8.95	8.62	8.97	8.27
Participant	16	17	18	19	20
Total Score	7.01	7.50	8.61	8.12	8.53

表 3 SSIM Scores of UI2

Participant	1	2	3	4	5
Total Score	8.21	8.56	9.25	7.60	8.08
Participant	6	7	8	9	10
Total Score	7.64	8.47	7.88	8.14	8.60
Participant	11	12	13	14	15
Total Score	8.55	9.08	8.90	8.96	7.04
Participant	16	17	18	19	20
Total Score	7.06	7.28	8.39	8.49	7.17

5.2 Result

大学院の学生 20 名を対象にこの評価実験を行なった．**表 2**, **表 3**, **表 4**, **表 5** はそれぞれ UI1, UI2 における SSIM 値を示しており，上段の 1 から 20 は実験参加者を対応させている．実験参加者の 1 から 10 に対しては UI1 から実施し，11 から 20 に対しては UI2 から実施した．また，これら全ての値は小数点以下二桁で切り捨てて表示している．**図 15**, **図 16** は得られた結果をグラフにしたものである．x 軸は 20 名の実験参加者を示しており，それぞれピンク色が UI1，水色が UI2 の値を示している．

表 4　Average of Using Time of UI1 [s]

Participant	1	2	3	4	5
Average Time	134	86	96	59	13
Participant	6	7	8	9	10
Average Time	20	27	44	214	65
Participant	11	12	13	14	15
Average Time	101	193	320	140	18
Participant	16	17	18	19	20
Average Time	25	29	98	170	91

表 5　Average of Using Time of UI2 [s]

Participant	1	2	3	4	5
Average Time	37	68	91	48	18
Participant	6	7	8	9	10
Average Time	20	30	44	34	61
Participant	11	12	13	14	15
Average Time	53	119	86	64	14
Participant	16	17	18	19	20
Average Time	26	21	61	127	59

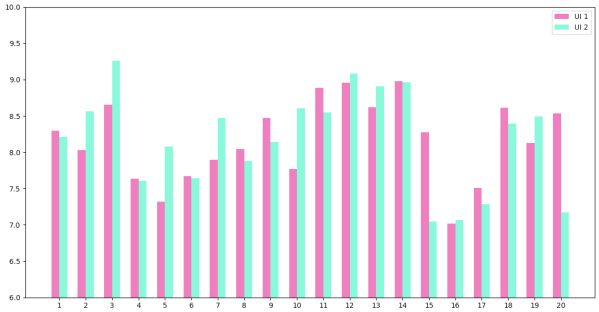


図 15　SSIM Scores

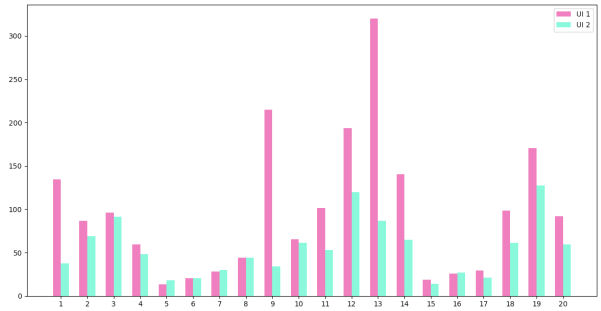


図 16　Time Usage

表 6　Average of Results

	SSIM	Using Time [s]
UI1	8.16	97
UI2	8.17	54

5.3　Discussion

まず，使用時間の合計に関しては個人差が大きい結果となった．しかし UI1 から実験を行なった 10 名と，UI2 から実験を行なった 10 名を比較しても，具体的な違いを観測することはできず，あくまで個人差が大きいという結果となった．

表 7　Total Time [s]

Participant	1	2	3	4	5
Total Time	171	154	187	107	31
Participant	6	7	8	9	10
Total Time	40	57	88	248	126
Participant	11	12	13	14	15
Total Time	154	312	406	204	32
Participant	16	17	18	19	20
Total Time	51	50	159	297	150

UI1, UI2 それぞれにおいて取得できた値を平均した結果を**表 6** に示す．単純に平均した結果を考えると，提案手法の UI2 の方が UI1 よりも SSIM の値が高く，使用時間は短いという結果となった．

SSIM の値に関しては，UI1, UI2 ともに有効数字二桁に丸め込むと一致するため，ほぼ同程度であると言うことができる．また，実験参加者のうち UI1 の値の方が高かった人が 11 名，UI2 の方が高かった人が 9 名であり，この観点からもほぼ同じ値を示していると言える．

一方で使用時間に関して平均値を見ると，UI1 が UI2 の約 1.7 倍もの時間を要している．これらに有意差があるかどうか確かめるために t 検定を行う．t 検定とは統計的検定の一つであり，二つの集団に有意差があるか調べる場合は，まずそれらに差が無いという仮説を立てる．そしてその条件下で統計量が t 分布に従うとしたとき，結果の起こる確率を計算，設定した有意水準と比較し，仮説の矛盾の有無を審議する．有意水準は 5%と設定されることが多いが，本研究の場合 0.71%と取得できたので，これらに有意差があると示すことができた．

20 人の実験参加者のうち，3 名（実験参加者 5, 7, 16）が UI1 の方が使用時間が短い結果となっているが，これらの人の共通点として，そもそもこの実験に対して使用した時間が短いという点がある．実験参加者全体における UI1, UI2 に使用した平均時間の和は 151 秒となる（**表 7** 参照）が，この 3 名はそれぞれ 31 秒，57 秒，51 秒と大幅に平均を下回っている．ここから考えられることとして，この評価実験において十分に時間をかけて，提示された画像を探していた実験参加者ほど，UI2 の使用時間の方が短い傾向がある．

さらに実験参加者のうち 4 名は，UI2 と比較して UI1 の方に 2 倍以上の長い時間をかけている（実験参加者 1, 9, 13, 14）．しかしながら，それぞれの SSIM の値に関して，4 名のうち 3 名が時間をかけた UI1 の方が高い値を示してはいるが，かけた時間の差と比較して明白な値の差がない．

6.　Conclusion

6.1　Summary

本研究では各印象項目におけるそれぞれの文字フォント

の分布を示すような，文字フォント推薦システムを提案した．まず，作成した “A”の画像データセットを用いて，潜在変数が 5 次元の VAE を学習させた．次にアンケート調査を用いて，その潜在空間内における “POP”，“Formal”，“Casual”の三種類の分布データを取得．そのデータに対して t-SNE を用いて 2 次元まで次元削減を実行，そして Kernel Density Estimation を用いて，確率密度関数を推定，各分布のヒートマップを作成した．そのヒートマップ上をマウスで移動することによって，文字フォント画像を連続的に変化させることができるアプリケーションを作成した．そして，ただ単に文字フォント画像を並べた UI と本手法の UI の両方を用いて，提示されたお題の文字画像を探してもらうという評価実験を行なった．その画像を見つけるまでに要した時間と，SSIM を用いて算出された，選択された画像と提示されたお題の画像との類似度を得た．この評価実験によって，本手法はただ並べられた UI よりも短い時間で文字フォントを検索することが可能であることが示された．

6.2 Limitation

まず，現状として “A”の一文字しか扱っていないということが問題である．フォントを実際に使用するとき，一文字のみを必要とする場面は基本的に考えにくい．そのため実用性を考えたとき，全アルファベットに対応する必要がある．この問題に関しては，2.1 節でも取り上げた手法を参考にすることで実現可能なのではないかと考えている．

また，出力画像が 28*28 ピクセルと小さいことも問題である．単純に画質を上げることも一つの解決策ではあるが，現在フォントの多くは，本研究でも使用した ttf（TrueType Font）に代表されるような，アウトラインフォントという形で保存，配布されている．アウトラインフォントとは，文字の輪郭などの情報を曲線パラメータで表現しているもので，拡大縮小などにもうまく対応することが可能なデータの型である．実用性を考える時，出力はこのような形であることが理想なのかもしれない．

6.3 Future Works

実用性を上げる方法として，既存の文字フォントを本手法で作成した潜在空間に対応させることが考えられる．“A”に対してエンコードすることによってこれは実現可能であり，VAE における生成という特徴が失われるが，現状の提案手法よりは実用的なアプリケーションが作成できるだろう．

また，本研究の潜在空間における人が受ける印象の分布を調査するという考え方は，フォント以外の分野についても応用可能であり，顔画像などを用いて実装しても面白いのではないかと考えている．

参考文献

[1] Google Fonts (online), available from ⟨https://fonts.google.com/⟩ (accessed 2019-1-18).
[2] FONT SQUIRREL (online), available from ⟨https://www.fontsquirrel.com/⟩ (accessed 2019-1-18).
[3] Paul Upchurch, Noah Snavely, and Kavita Bala: From A to Z: Supervised Transfer of Style and Content Using Deep Neural Network Generators, arXiv: 1603.02003 (2016).
[4] Samaneh Azadi, Matthew Fisher, Vladimir Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell: Multi-Content GAN for Few-Shot Font Style Transfer, arXiv: 1712.00516 (2017).
[5] Saemi Choi, Kiyoharu Aizawa, and Nicu sebe: FontMatcher: Font Image Paring for Harmonious Digital Graphic Design, ACM, pp.37–41 (2018).
[6] Neill D.F. Campbell, Jan Kautz: Learning a Manifold of Fonts, ACM Transactions on Graphics, Vol 33, 4 (2014).
[7] Yuan Guo, Zhouhui Lian, Yungmin Tang, and Jianguo Xiao: Creating New Chinese Fonts Based on Manifold Learning and Adversarial Networks, EUROGRAPHICS (2018).
[8] THE MNIST DATABASE of handwritten digits (online), available from ⟨http://yann.lecun.com/exdb/mnist/index.html⟩. (accessed 2019-2-4).
[9] Diederik P. Kingma, Max Welling: Auto-Encoding Variational Bays, In Proceeding of the International Conference on Learning Representations (ICLR). (2014).
[10] Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra: Stochastic Back-propagation and Approximate Inference in Deep Generative Models, Technical report, arXiv: 1401.4082. (2014).
[11] Keras: The Python Deep Learning Library (online), available from ⟨https://keras.io/⟩ (accessed 2019-2-4).
[12] Laurens van der Maaten, Geofferey Hinton: Visualizing Data using t-SNE, Journal of machine Learning Research 9, pp.2579–2605. (2008).
[13] Geoffrey Hinton, Sam Roweis: Stochastic Neighbor Embedding, In Advances in Neural Information Processing Systems, Vol15, pp.833-840. (2002).
[14] B. W. Silverman: Density estimation for statistics and data analysis, (1986).
[15] SciPy.org (online), available from ⟨https://www.scipy.org/⟩. (accessed 2019-2-4).
[16] Matplotlib: Python plotting (online), available from ⟨https://matplotlib.org/⟩. (accessed 2019-2-4).
[17] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh and Euro P. Simoncelli: Image Quality Assessment: From Error Visibility to Structural Similarity, IEEE Transaction on Image Processing, Vol.13, No.4. (2004).