RECOGNITION AND RECOMMENDATION OF HANGUL FONTS USING COMPUTER VISION BASED TECHNIQUES

TEAM 9
OUM TAEHWOI 2012-11114 KIM HYEONSEO 2015-12377
CHOO WONHO 2014-18868 CHUNG SEIJUN 2018-23341
Computer Vision 2019-1
Department of Computer Science
Seoul National University

Abstract

Objective: *The aim of this study is to find key features of Hangul fonts, in order for users to be informed of the name of font types or recommended.*

Background: *While there are many researches related and tools available in identifying fonts of Latin alphabets, research on Hangul letters is not active due to its complexity that Hangul fonts consist of 2,350(KS X 1001) or 11,172(KPS 9566) letters, each composed of onset, nucleus and codas.*

Method: *First, we separate a letter from input image, focusing on minimizing the marginal spaces beside the letter. Second, we identify which letter it is, such as '가' or '나'. And then we calculate a large dimensional feature vector for each fonts with HOG, Histogram of Oriented Gradient. Finally, by comparing the feature vector among all kinds of fonts, we could be notified which font the input text has. Results: We have achieved 75% of correct ratio for inferring the actual input font. For wrong predict of 25%, we found that the recommended font is very similar to input in aspect of structural shape and its slope.*

Conclusion: *We found that HOG Descriptor is efficient tool for recognizing. And it is very crucial in pre-processing to minimize the marginal spaces beside the letter to improve the correct ratio.*

1. Introduction

Fonts are predefined collections of letters, varying in design and size. From the monospaced fonts on a VT100 terminal to the Seoul fonts by Seoul city, the range of available fonts to users have expanded, due to the rise of tools for font creation. Different fonts are utilized based on the purpose of the user, as an intermediary of the user's purposes and emotions.

As more fonts are introduced to the market, the quest for the appropriate font has become an evergoing battle with vast images. Also, finding the appropriate font is not just a matter of choice, as it can spur legal disputes as well.[1]

Compared to Hangul fonts, English fonts consist of 26 upper and 26 lower case letters, a total of 52 letters. The limited number of letters, and their simple form makes English font recognition not as challenging and previous researches have produced high recognition rates.

However, Hangul fonts consist of 2,350(KS X 1001) or 11,172(KPS 9566) letters, each composed of onset, nucleus and codas. Due to this complicated format, research on font recognition of Hangul letters is not active, compared to other Latin alphabets. In this project, we will use KS X 1001 letter set.

The object of this paper is to address these issues, identifying Hangul letters from images to recognize their font and recommend similar fonts.

2. Background and related work

There are many researches related and tools available in identifying English fonts. One of the open source python libraries that help identifying fonts[2] uses Hausdorff distance and Shape context to identify fonts from an image. Also, one of the online font search websites[3] uses the representative characteristics of English fonts like serif, stem, tail, etc.

The Hangul system is quite different from that of English. Before we discuss the difference, the definition of a 'character' is any mark or symbol that can appear in writing. English font identifiers compare fonts by letters, e.g. 'a', which is a character that is a part of an alphabet. However, in Korean, the comparison unit should be like '가' or '간'. So in this paper, we will call '가' as 'letter' and 'ㄱ'

as ‘character’.

We assumed the problem of identifying fonts from images is similar to the problem of distinguishing the similar images and find the difference. To distinguish the difference between similar images, we can extract the representative features from images with many methods like SIFT, HOG, SURF, etc. These methods are called feature descriptors. A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. Then we can compare features.

So, as the fonts can be differentiated just from slight changes, we chose HOG to identify fonts in this project.

3. Algorithm

Algorithm 1 Font Recognition

```
1: split *letters* in *input image*
2: **for** $l$ in *letters* **do**:
3:     find $idx$ that $l$ == ref_font[$idx$]
4:     **for** *f* in *fonts* **do**:
5:         compare(l, font[idx])
6:     select most similar font
7: select font with most concensus
```

The algorithm above briefly explains how font recognition is done. First, extract letters from an image. Compare the letters with their equivalents in the reference font sets, and select the best match.

4. Method

Please refer to the author guidelines on the CVPR 2018 web page for a discussion of the policy on dual submissions.

4.1. Seperating Hangul letters from image

To find the most similar font to an input, pre-processing the input is needed. We need to decompose an input image to the images of each letter, remove paddings and resize it to predefined size to make comparing easier. We decomposed this pre-processing process into three parts.

First is to find the area where the font we are interested in exists. We did it by using tessoract and tesserocr library. By using this library, an input image is divided into the areas where korean letters exist line by line.

Second is to divide the input image into the image of letters. To do this, we first converted an input image to a grayscale image and binarized it. This was to eliminate noises. But, since the binarizing may change the shape of font a little bit, we only used it to make cuts. After making cuts, we used the original(not binarized) image.

Following is the process of making cuts. We did it by calculating the average value of pixel columns. If we assume white pixel value as 255 and black pixel value as 0, column average of the black part will be 255 and if there is a part of character in the column, the average will be less than 255. If the height is 100 pixels, and one pixel is black and all other pixels are white, the column average will become 252.45. So if we set the threshold of making cuts as 253   254, we can group the pixel of the input image into the blank part and the character part.

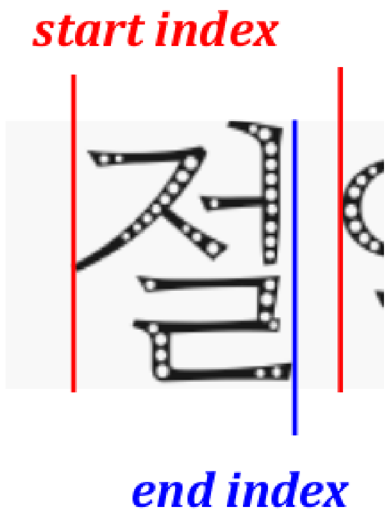

Figure 1. Start index and end index of a letter

After that, we made two lists of column indices, starts and ends. Since we grouped the pixel of the input image into the blank and the character part, there are some column indices that the parts alternate. When the blank part alternate to the character part, we appended that index to starts list. And when the character part alternate to the blank part, we appended that index to ends list.

But there was a problem. For example, if the letter is ‘가’, it is separated into ‘ㄱ’ and ’ ㅏ’ because there is a blank part between ‘ㄱ’ and ‘ ㅏ’. To solve this problem, I calculated the width between ends and starts. In case of ’ ㅏ’ it will be small compared to height. If the ratio of the width and the height is small enough, I deleted the index from starts and ends list.

After that, I made a list named cuts. It’s just an median of starts and ends. This process is needed to make a robust cut. Since we binarized image, the edge of the letter can be cut off. This may result in some problems when characterizing features. To avoid that problem, I made some space by averaging the starts and ends.

Since the cuts are made, an input image can be divided into letters. But there are spaces, which decrease the accuracy of font recognition. To solve this, the last part we did in pre-processing is removing spaces. By using opencv li-

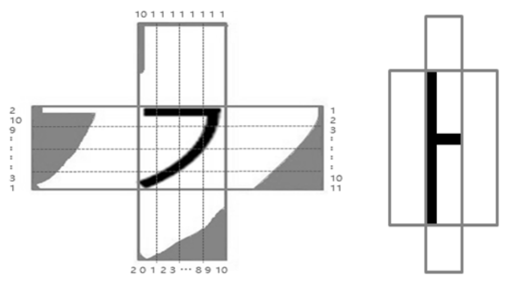

Figure 2. A 4 x 4 Contour Profile of Character ‘ㄱ’, of Letter ‘가’

brary, we could make a bounding box of a letter. We cut the image to the size of bounding box and resized it to (64, 64). There can be a distortion if we resize it, but the original font in database is also distorted in the same rate, so there is no problem in detecting fonts.

Overlength papers will simply not be reviewed. This includes papers where the margins and formatting are deemed to have been significantly altered from those laid down by this style guide. Note that this LaTeX guide already sets figure captions and references in a smaller font. The reason such papers will not be reviewed is that there is no provision for supervised revisions of manuscripts. The reviewing process cannot determine the suitability of the paper for presentation in eight pages if it is reviewed in eleven.

4.2. Identifying letters from letter image

Although fonts do not rely on the content of the text, recognizing the letter can reduce the search space of fonts. For this reduction to be significant, the computation required for letter recognition should be computationally efficient than that of font recognition among all possible samples.

The feature chosen for letter recognition was the Contour profile.[5] Profile is the number of pixels between the bounding box of the character, and the edge of the character. This feature was chosen as it is computationally efficient, compared to other operations like pixel-wise gradients. It also utilizes characteristics of Hangul letters, that has varying number of contours depending on its composition.

To construct profile for the left side, first locate the upper left corner and the bottom left corner of the contour. For each pixel on the vertical line, the number the white pixel parallel to the horizontal line is counted. We apply the same logic to the remaining 3 sides(right, top, and bottom), to get a feature vector for a character. The contour profile is computed for the rest of the characters, and the letter's feature vector is attained by concatenating the contour profile of its characters.

After the feature vector is computed, kNN classifier is used. The letter's feature vector is compared with other precomputed feature vectors of the dataset by Euclidean distance. For vectors of different dimensions, maximum distance was used, as letters with different numbers of characters are not neighbors. Among the k nearest neighbors, the most common letter was inferred as the input image's letter.

4.3. Comparing the input image and the database

We have identified which letter the input text is, such as ‘가’ or ‘나’. After that, we investigated the font of the input text Histogram of Gradient, HOG descriptor, which is the feature descriptor.

We selected HOG descriptor, as there might be distinguishable features with respect to the gradient of every single font even among those having similar structural shape. HOG descriptor shows excellent performance at describing the structural shape of an object.

Typically, a feature descriptor converts an image of size width x height x 3 (channels) to a feature vector / array of length n. In the case of the HOG feature descriptor, the input image is of size 64 x 128 x 3 and the output feature vector is of length 3780.

HOG captures local intensity gradients and edge directions, it also makes for a good texture descriptor.

Algorithm 2 Computing HOG

```
1: Load image and resize to 64X128 and Normalize
2: Compute the gradients (the vertical, the horizontal).
3: Compute the magnitude of the gradient.
4: Compute the direction(angle) of the gradient using arctan.
5: Split the image to cells and compute the histogram to each cell.
6: Split the image to blocks
7: compute block normalization.
8: Calculate the HOG feature vector.
```

We followed the standard HOG implementation in step 1 to 4. Then in step 5, we split the image into cells and compute the histogram of each cell. A cell size was (8X8) pixels.

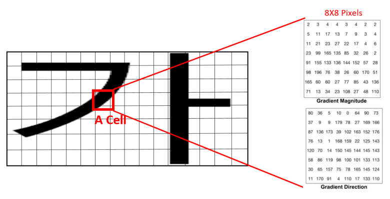

Figure 3. A Cell has the Magnitude and Direction of Gradient of each 64 pixels

We made the 9 bins based on the direction of gradient for the histogram: [0, 20, 40, 60, 80, 100, 120, 140,160]. And we distributed the magnitude of gradient proportional to the probability that the direction of gradient can select the bin.

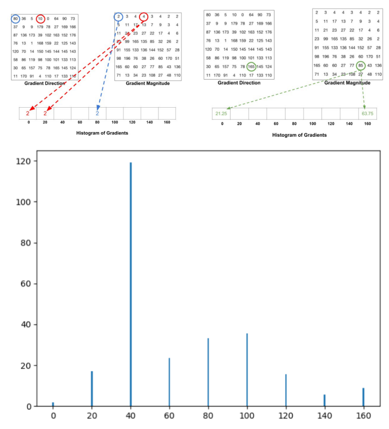

Figure 4. Example of calculating Histogram of cell

Then we grouped the cells into blocks and normalized it due to the fact that gradients of image are sensitive to overall lighting. We chose grouping (2 x 2) cells into one block as it obtains reasonable accuracy in most cases according to Dalal and Triggs[4].

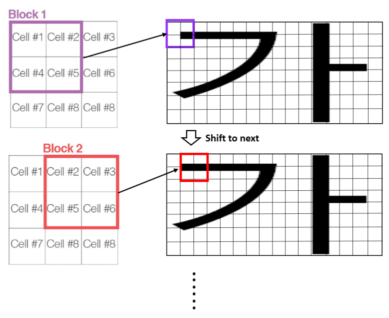

Figure 5. Block shift. Overlapping over the image

Then we normalized blocks using L2 norm. As 105 blocks can be generated in one image and each block contains 4 cells X 9 values of the histogram, the following dimension of HOG feature vector of an image is $36 \times 105 = 3780$ dimensional vector.

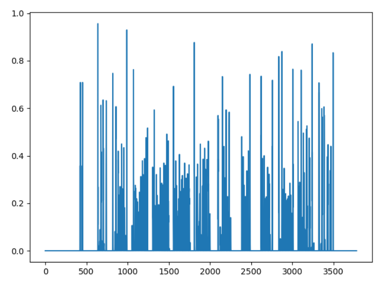
Figure 6. A HOG Feature Vector of a text image ‘가’

5. Experiment

We inferred the font type of input texts, as an experiment.

Through the procedure above, we obtain a $36 * 105 = 3780$ dimensional vector for a letter. Each letter has its unique HOG feature vector, even though they are of same letter.

Then, it is possible to discriminate the fonts among the same letters by comparing the HOG feature vectors to each other.

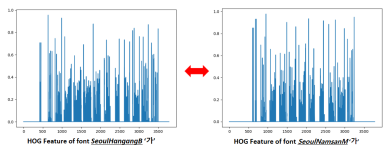

Figure 7. Different HOG Feature Vector observed for ‘가’ in font SeoulHangangB and SeoulNamsanM

Suppose that the input letter is identified as ‘가’, it is possible to calculate HOG feature vector for all kinds of fonts of ‘가’ and store these into a dictionary, in the form of pair font name : its HOG feature.

In sequence, we could calculate HOG feature vector of the input letter and compare the histogram to find a font identical to input in the dictionary.

We conducted the experiment with 21 sentences, each with different typefonts.

5.1. Experiment Results

We were able to achieve 75% hit ratio, as seen in the table.

Input No.	Actual Font (of Input)	Index of Font		Hit	Lowest in MSE
		Actual y	Predict y		
1	BareunBatangB	45	85	X	0.0051
2	BareunDotum1	128	147	X	0.0048
3	Binggrae	76	101	X	0.0052
4	drfont_daraehand_Basic	109	109	O	0.0047
5	DX80StarNyouB	122	122	O	0.0061
6	DXSePilMJL	33	125	X	0.0064
7	GyeonggiBatangBold	66	66	O	0.0065
8	Typo_BearRabbitM	5	5	O	0.0041
9	Typo_HelloPOP_OutlineM	21	21	O	0.0023
10	HoonDdukbokki	6	6	O	0.0045
11	HoonGomusinR	17	17	O	0.0049
12	HoonProvenceR	11	11	O	0.0046
13	HoonYunpilsupkyukR	27	27	O	0.0056
14	IropkeBatang	162	162	O	0.0053
15	ChosunilboMJ	Out of Database			
16	MaplestoryBold	3	3	O	0.0038
17	NanumGothic	101	101	O	0.0049
18	OseongandHanEum-Regular	159	159	O	0.0058
19	SeoulHangangM	59	148	X	0.0053
20	SeoulNamsanM	112	112	O	0.0044
21	THEdog	31	31	O	0.0043

For five cases of wrong prediction, we found that the inferred font is very similar to the input in aspect of structural shape and its slope.

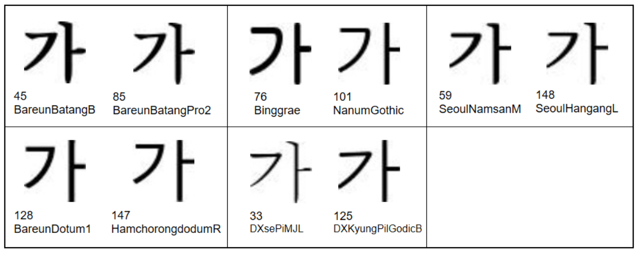

Figure 8. Comparison between Ground-truth and Prediction

6. Conclusion

Based on Computer-Vision techniques, we performed a series of processes to hit or recommend the font types of input text that users insert.

We found that HOG Descriptor is efficient tool for recognition.

When it comes to using HOG Descriptor, for better correct ratio, it is very crucial to minimize the marginal spaces beside the letter, not only when parsing the image, also generating database.

For the generalization of this technique to identify other languages' fonts as well, further processing of images to get another type of features is required. As the usage of HOG Descriptor does not matter the kinds of languages, if we get to know the letter then the problem is solved. But with the current method, the number of bounding boxes of a letter in English for example, is one. Then the success ratio of letter recognition can go lower, as we had more than one bounding boxes when doing letter recognition in Korean. Therefore, it is necessary to add supplementary letter recognition method when identifying fonts of other languages.

7. Data set

Since there's not enough data set existing, we created new dataset. We downloaded .ttf font files offered by Naver Corp. Using the downloaded font files, we generated 64 * 64 image files for 2,350 Korean letters. [6] Therefore, there will be 2,350 * (# of distinct fonts) image files of size 64 * 64.

As code [6] makes the data set as black background images with white letters on, we reversed the color of font images to match with input images. So the letter is written in black and the background color is white. After that, by using opencv library, we could make a bounding box of a letter. We cut the image to size of bounding box and resized it to (64, 64). Therefore, the recognition of fonts using input images would not be affected by the spaces in the database.

References

[1] The Font Bureau, Inc. v. NBC Universal, Inc. et al https://dockets.justia.com/docket/new-york/nyedce/1:2009cv04286/296847

[2] https://github.com/Vasile-Peste/Typefont

[3] http://www.bowfinprintworks.com/SerifGuide/serifsearch.php

[4] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2005

[5] I. D. TRIER, A. K. JAIN and T. TAXT. FEATURE EXTRACTION METHODS FOR CHARACTER RECOGNITION–A SURVEY. Pattern Recognition, 29(4), 641-662, 1996.

[6] https://github.com/IBM/tensorflow-hangul-recognition