Recognising Hiragana Characters Using Optical Character Recognition

Joonas Lõmps
Institute of Computer Science
University of Tartu
b12027@ut.ee

Abstract

Optical character recognition (OCR) has become more and more common technology, be it automated teller machines (ATM), office scanners or the scanners used at stores. In addition to that the automated recognition of handwritten characters is commonly studied problem in Computer Vision and has several applications in real life. This project focuses on recognising handwritten Japanese hiragana characters using an open source computer vision library OpenCV[1]. It also creates an application that adds pronunciations to detected hiragana characters with relatively high accuracy.

I. Introduction

Character recognition, or reading as we call it, comes to humans with some training and practice. They are even able to read texts that are written with poor handwriting. Possibilities and ways of automating that process have been studied for a long time. There are already several effective applications: recognition of postal codes, processing administrative forms and etc. As there are several applications for digits and English alphabet, and as a fan of Japanese culture and the interest of knowing if similar approach would work on more complex alphabet the author decided to test it for Japanese.

Japanese uses three different alphabets: hiragana, katakana and kanji. Hiragana and katakana both consist of 46 distinct characters as well as some functional marks and diacritics to add double consonants or soften them. Kanji consists of around 50 000 distinct characters but only a small portion of around 2000 kanji is enough for everyday use. Hiragana was chosen, since the author lacked a dataset of character images and had to make it himself.

The author focused on two well-known classifiers used for Machine Learning[2] such as k-Nearest Neighbour (k-NN) and Support Vector Machines (SVM). Since both of them, and much more, were supported in OpenCV [1] it was the library that was used.

Two different features of the character images are tested in this project: raw pixel data and the histogram of oriented gradients[3].

An application that is capable of reading handwritten Japanese sentences is implemented and introduced. The author also explains challenges that came with the application and his solutions to them.

First, the author describes how the dataset was generated, briefly explains how classifiers were used and data features in Section II. Then, in Section III the initial results of the algorithms with single character pictures are introduced. Finally, in Section IV the application is introduced.

II. Dataset, classifiers and data features

There are several datasets for handwritten digits and character on the Internet, but the author did not manage to find one for Japanese.

That is the reason why a single Japanese alphabet was chosen, hiragana. Since the author has studied Japanese for several years he decided to generate his own dataset. A digital drawing pad was tested, but it did not speed up the process and the handwriting suffered quite a bit because of it, so it was decided to use a 0.5mm gel pen and a sheet of white paper. Each of the 46 characters were written 10 times in a row. The characters were scanned in with a scanner. They were one by one cut out using Paint.NET[4], which after getting the hang of it did not take that long. After that the author ended up with 460 characters of different sizes around 60-90x60-90px each (Figure 1). A simple python code utilizing OpenCV[1] libraries was used to generate 20x20px pictures of every character. Every character was stored in a separate file. After running first tests with 20x20px images the author noticed that downgrading the images that much had removed some lines from the images and decided to use 50x50px images, which did not have that problem.

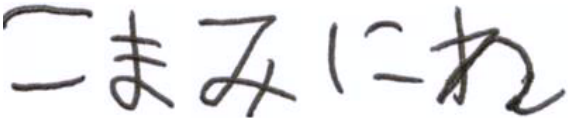

Figure 1. Cut out data from scanned image

As mentioned earlier two classifiers were used:

- *k*-**Nearest Neighbor** - Experiments showed that the best accuracy was achieved with $k = 4$. As shown in Section III, *k*-NN gives a rather good accuracy rate when testing on single character images.
- **Support Vector Machine** with **Linear kernel** - The parameters for this classifiers were found on the Internet. $C = 2.67$ and $\gamma = 5.383$ [6]. The author tried playing around with the parameters, but could not find a set of parameters that would increase the accuracy. SVM also provides high accuracy rate when testing on single character images, as it can be seen in Section III.

Features used in experimenting were:

- **Raw features** - Simplest and most straightforward feature: the 2500 pixels of 50x50 grayscale image, where every pixel representing its intensity in grayscale.
- **Histogram of oriented gradients (HOG)** - The image is divided into small connected regions, and for the pixels in each cell, a histogram of gradient directions is compiled. The way the author implemented it he gets 3200 features per image.

III. Single character results

For single character recognition the author divided the dataset of 10 images per character into two sets: training 8, testing 2 images. Before the images were ready for training or testing they were preprocessed for increased accuracy, that the author learned from Vincent Neiger [5]. Every image was loaded in, then threshold was taken and then the image was dilated with 2 iterations (Figure 2).

Figure 2. Left: initial image. Center: threshold image. Right: dilated image.

The results for different classifiers with different features can be seen in Table 1.

Classifier	Feature	Accuracy
k-NN	Raw	92.39%
k-NN	HOG	91.30%
SVM	Raw	94.57%
SVM	HOG	96.74%

Table 1. Accuracy rates for recognising single character

The author did not expect to see that high accuracy rates because of the similarity of multiple characters (Figure 3).

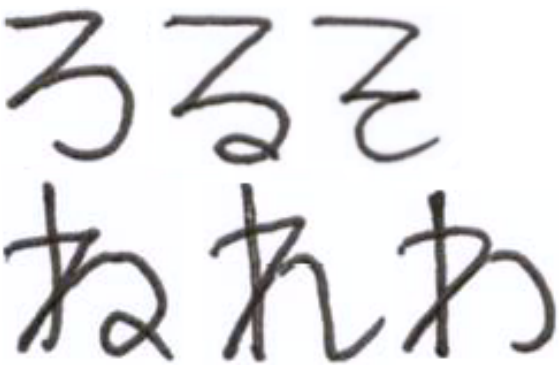

Figure 3. Some examples of similar characters with different pronunciation.

In contrary to the authors intuition, these cases did not produce incorrect results. A mistake that appeared in every combination of classifiers and features is shown in Figure 4.

Figure 4. Most common mistake ("ki" vs "sa").

The highest accuracy was achieved with a combination of SVM and HOG. Author of [5] also measured the time that it takes to train and recognise. From his experiments it can be seen, that training k-NN is times faster than SVM, but recognising is times slower than SVM.

IV. Recognition application

The general idea of the application was to take an image of an handwritten hiragana sentence and add pronunciations for every character. For the simplicity the author assumes that the characters are written on clean sheet and there is nothing else on the image.

The author decided to use a method called background subtraction to detect the characters. It works well when the every character itself is a single connected component. For hiragana it is not that simple, there are many characters that are not connected. When characters are not connected, then background subtraction detects every connected part as a separate object as it can be seen in Figure 5. It detects 6 objects instead of of the actual 3.

Figure 5. Background subtraction on 3 hiragana characters.

Luckily, the characters in hiragana have a similar pattern when they are not connected. They are three main cases:

- **On top of each other** - It can be seen on Figure 1 as the first character.
- **Side by side** - It can be seen on Figure 5 as the second character.
- **Both together** - It can be seen on Figure 5 as the first character.

For the case where the unconnected parts are on top of each other their starting point from left on x-coordinate is supposed to be on the same spot, it varies a bit depending on the handwriting. So what the author used to make those components into one, was that if $|x_1 - x_2| \leq 10$, where x_1, x_2 are the x coordinates of the back rectangle around the component, then they belong together. Result can be seen on Figure 6.

Figure 6. Detected components after merging ones that are on top of each other.

Similar logic was used for the case where unconnected parts are side by side. The equation that was used there was $|x_1 - x_2| \leq 50$.

It works because in perfect handwriting every character in hiragana should have the same width, which means that if there are two components which x-coordinates are close to each other, they belong to the same character. Result can be seen in Figure 7.

Figure 7. Detected components after merging ones that are close side by side.

Having transformed and updated the background subtraction data every detected character is cut out, resized to 50x50px, preprocessed like mentioned in Section III. The resulting image is used for character recognition and the resulting pronunciation is written below the character as seen in Figure 8.

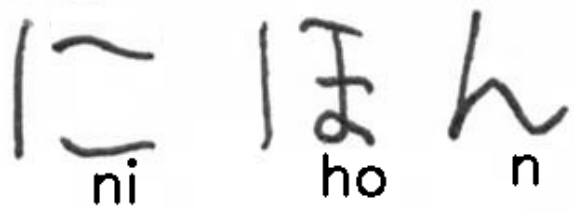

Figure 8. Final result of the application.

V. Conclusion

Machine Learning is an inseparable part of Computer Vision and this project provided an interesting first insight to Computer Vision for the author. Simple Machine Learning algorithms were studied and used to implement and application that is capable of providing pronunciation to Japanese hiragana character on an image. The applications gave an accuracy far beyond the authors expectations even with a small training dataset. The author believes that the accuracy can be increased even more with bigger dataset, better parameters for classifiers and combining results of different features. The application itself could also be improved to support images with multiple rows of characters. Still, while there is room for improvement, the author is fascinated by the field of Computer Vision and Machine Learning.

References

[1] "The OpenCV library website" *http://opencv.org/*

[2] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE 86.11 (1998): 2278-2324.*

[3] Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 1. IEEE, 2005.*

[4] "The Paint.NET website" *http://www.getpaint.net/index.html*

[5] Vincent Neiger "Handwritten digits recognition using OpenCV" *Machine Learning in Computer Vision 2015.*

[6] "SVM Paramaters" *http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_svm/py_svm_opencv*