Bachelor's thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Mobile Application for Recognition of Japanese Writing System

Jan Zdeněk

Program: Open Informatics
Field: Computer and Information Science

May 2014
Supervisor: prof. Ing. Pavel Zahradník, CSc.

Acknowledgement

First of all, I would like to express my gratitude to my supervisor, prof. Ing. Pavel Zahradník, CSc., for giving me the opportunity to work on this project. I appreciate his guidance and comments on my work and I would like to thank him for giving me the freedom to explore on my own.

I would like to acknowledge the help of my friend Jakub Černý who gave me his opinion on the thesis and provided me with helpful comments.

Finally, I want to thank my parents for their love, encouragement and constant support throughout my studies.

Declaration / Prohlášení

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

In Prague, May 23, 2014 ....
Jan Zdeněk

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne, 23. května 2014 ....
Jan Zdeněk

Abstract / Abstrakt

The objective of this work was to implement and compare various methods which can be used for optical character recognition (OCR) of characters used in the Japanese language and create a mobile application which could recognize characters in an image captured by the camera of a device and present the user with a translation of the words into English.

The engine for recognition has been trained on a database with 2,908 unique characters. Using the moment shape normalization, the Histogram of Oriented Gradients for feature extraction, and quadratic discriminant function for classification, recognition rate of 99.9% has been achieved on isolated characters in standard fonts used in print publications and the engine outperformed some of the free OCR engines for Japanese.

An initial alpha version of a mobile application for Japanese OCR for Android operating system has been created and the engine with optimal results has been successfully implemented therein. The author intends to continue the development of the application and expand the range of its possible use. The OCR engine could also be used in other applications requiring Japanese OCR.

Keywords: Pattern Recognition; OCR

Cílem práce bylo implementovat a otestovat různé metody použitelné pro optické rozpoznávání znaků (OCR) používaných v japonštině a vytvořit mobilní aplikaci, která bude rozpoznávat znaky v obraze zachyceném fotoaparátem zařízení a poskytne uživateli překlad textu do angličtiny.

Systém pro rozpoznávání byl natrénován na databázi obsahující 2908 unikátních znaků. Za použití momentové normalizace pro normalizaci tvaru znaku, histogramu orientovaných gradientů pro extrakci příznaků a kvadratické diskriminační funkce pro klasifikaci bylo dosaženo úspěšnosti rozpoznání 99,9% na izolovaných znacích ve standardním fontu používaném v tištěných publikacích. Systém byl úspěšnější než některé existující volně šířené systémy pro OCR japonštiny.

Byla vytvořena počáteční alfa verze aplikace pro operační systém Android a úspěšně v něm byla implementována verze systému dosahující nejlepších výsledků. Autor chce nadále pokračovat ve vývoji aplikace a rozšířit spektrum její použitelnosti. OCR systém aplikace by mohl bý použit i v dalších aplikacích vyžadujících OCR japonštiny.

Klíčová slova: Rozpoznávání; optické rozpoznávání znaků

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 State of the Art and the Goal

Optical Character Recognition (OCR) is the process of converting an image containing text into machine-readable information representing the characters in the image. OCR is one of the oldest fields of computer vision and pattern recognition and the first ideas giving birth to OCR date back to the late 1920's [1]. The first attempts at OCR were realized in the 1950's after the rise of modern computers and the approach that was applied involved optical and mechanical matching of the character being recognized with character templates. Rays of light were cast at the character and their reflection was intercepted by photo-sensitive detectors. The acquired analog values were digitized and compared with the templates. At first, the OCR systems were limited only to recognition of characters written in one specific font, but systems that could recognize characters in multiple fonts eventually started appearing [2], and as more time passed, the range of usability expanded and it became possible to recognize handwritten characters up to a certain extent, too. Contemporary OCR systems work with digital images, using various advanced techniques used in pattern recognition, and achieve very high recognition rates even on handwritten characters that are difficult to read.

A lot of approaches to OCR have been tried out over the years, but all approaches have something in common and that is the presence of two essential components:

- **Feature extractor**, which extracts information about the character from an image. The methods of extraction can be divided into two categories.
  - **Template matching**, where the image is compared with character templates, usually using image pixels as features.
  - **Structural analysis**. There are various attributes that can be used as features in an image, such as corner positions, stroke orientations, character holes, etc. This approach is more complex, but it is also more robust than simple template matching.
- **Classifier**, which uses the extracted information, features, to classify the character in the image as one of the characters in the set of characters it can recognize.

There are many factors that make OCR a difficult task due to which OCR is still an active field of research in pattern recognition, despite the performance of the best systems being remarkable. The most significant ones are:

- **Similarity of characters**. When two characters have a similar shape, the features extracted from them is also likely to be similar and they are more prone to be classified incorrectly. The more characters the OCR system is supposed to recognize, the smaller the differences between characters will get and the harder will their recognition become.

- **Variability of typography**. The text may be printed in a wide variety of fonts or it may be handwritten. The spacing and the size may also differ.
- **Segmentation**. Individual characters may be connected with each other and they first need to be correctly segmented before they can be classified. Even if the characters in text are not connected, they have to be correctly separated. Most characters in the Latin alphabet are single connected components, but many characters in other scripts do not have this attribute and may consist of several connected components.
- **Image quality**. There can be stains, artifacts, and other defects in the image. The lighting conditions can be bad and the image resolution can be low. All of this makes the recognition more difficult.

Today, there are a lot of OCR systems[1] available, both commercial and free, and they are used in various applications across many fields. OCR can be used for example for ID card checking, license plate recognition, making of editable text versions of books and other printed documents, assistance for visually impaired people, etc.

OCR for a language which uses a very complex writing system such as the Japanese language is a task even more demanding than OCR for a language using the Latin alphabet due to the large difference in number of characters. Japanese writing system uses Chinese characters - usually called kanji (lit. Chinese character) when referring to their use in Japanese - imported from China and two sets of syllabaries developed from kanji. It is impossible to specify an exact number of characters used in Japanese language, but approximately 3,000 characters are commonly used [3]. Despite the complexity of the task, a lot of research has been done over the years in the field of OCR and nowadays, one can find OCR engines supporting Japanese language in commercial applications, scanners, and other machines.

Various research institutes compete with their leading-edge engines for optical recognition of Chinese characters in competitions organized by the International Conference on Document Analysis and Recognition (ICDAR) [4], [5] and the best engine has achieved accuracy of recognition that reaches 94.77% on individual, hardly legible handwritten Chinese characters from GB2312-80 character set[2] consisting of 3,755 Chinese characters and 88.76% on handwritten natural Chinese text. However, it goes without saying that such engines often have high computing power and memory requirements and their range of possible use is limited by that.

With the increasing computing power of mobile devices, it has become possible to create a mobile application that would recognize Chinese characters and characters from the Japanese syllabaries in real time. However, only a few attempts have been made to bring OCR for Japanese language to the field of mobile applications. While there are a few mobile OCR applications supporting Japanese language available for download on the Play Store[3] of Android operating system and the App Store[4] of iOS operating system, their performance leaves a lot to be desired in most cases. Some of the applications, mostly those that are built for multi-language support, lack the desired accuracy of recognition and their usability is very limited. Other applications only support recognition of text in a standard font used in printed publications and cannot handle text printed in more ornate fonts, fonts that resemble handwritten text and real handwritten text.

As the Japanese language uses a complex script that requires years for a person to learn, a well-performing application which can recognize and translate a text in Japanese can be very

[1]Comparison of optical character recognition software - `https://en.wikipedia.org/wiki/Comparison_of_optical_character_recognition_software`

[2]GB2312 character set - `https://en.wikipedia.org/wiki/GB_2312`

[3]Application distribution center for Android maintained by Google.

[4]Application distribution center for iOS maintained by Apple.

useful not only for tourists traveling to Japan, but also for students of Japanese all over the world. Looking up a word in a dictionary requires a certain amount of time when one does not recognize the characters used in the word because one cannot type the word on a keyboard without knowing how to read it, neither one can find it in a regular printed Japanese-English dictionary. One has to analyze the characters and find the individual characters in a kanji dictionary by means of common patterns[5] appearing in the characters, and only then it becomes possible to look up a word in an electronic dictionary. Another way is to draw the characters on a device supporting online recognition of characters used in the Japanese language, which is performed by analyzing how the character has been drawn, particularly by the order and direction of individual strokes. Both approaches are rather time-consuming and being able to scan the unknown word using a mobile device would save one a lot of time in the long run.

The goal of this work is to test and compare various methods applicable for OCR of characters used in Japanese language and create a mobile application which will be able to scan text in Japanese using the camera of a mobile device and recognize the characters therein. Subsequently, the application will display the scanned word or several words on its screen and provide the user with its reading - the way the word is pronounced in Japanese - and translation into English.

1.2 Related Work

There are several mobile applications for OCR of Japanese text for Android OS and iOS. *Mobile OCR* by Smart Mobile Software[6], *OCR* by Angeldroid Studio[7], and *OCR Instantly Free* by TheSimplest.Net[8] are OCR applications for Android with multi-language support. *Mobile OCR* can only recognize text in images or photos saved on the device. *OCR* offers support for Japanese language, but it admits in its description that the performance is poor. *OCR Instantly Free* also works only on saved images and admits that the performance on Japanese language is lacking. Anyway, none of these three applications provide translation into another language and only convert an image with text into a machine-readable information about the characters in the text.

Japanese Text/Kanji OCR by space.works[9] is an application focusing on recognition and translation of Japanese text, but according to its developers, it is in beta-phase and no update has been made since June 2013, which indicates that the development may be on halt. The recognition takes a lot of time and the accuracy is low. Another application for Japanese OCR with built-in translation, *Kanji Yomi* by inda3[10], is available on Android. It requires internet connection to perform recognition, because it uses the services of the WeOCR project, which allows it to perform recognition on its servers using the free OCR engine for Japanese language, NHocr, introduced below. However, the latest update was made in September 2010 and the ratings and reviews of the application are very diverse.

The situation on the App Store for iOS is much brighter. There are two applications for Japanese OCR with subsequent translation of the recognized text whose performance is outstanding. *Japan Goggles* by LucSens Oy[11] does not seem to be updated anymore - the latest update was made in May 2011 - but its performance on printed text with good contrast is very

[5]Kanji can be looked up using so called radicals, which are common patterns appearing in more kanji characters.

[6]Mobile OCR - `https://play.google.com/store/apps/details?id=com.smartmobilesoftware.mobileocrfree`

[7]OCR - `https://play.google.com/store/apps/details?id=app.angeldroid.ocr`

[8]OCR Instantly Free - `https://play.google.com/store/apps/details?id=com.thesimplest.ocr`

[9]Japanese Text/Kanji OCR - `https://play.google.com/store/apps/details?id=space.works.jocrfree`

[10]Kanji Yomi - `https://play.google.com/store/apps/details?id=jp.ne.biglobe.inda3.kanjiyomi`

[11]Japan Goggles - `http://japangoggles.lucsens.com/`, `https://itunes.apple.com/us/app/japan-goggles/id397724055?mt=8`

good and the description says it can recognize over 3,000 characters. A newcomer OCR application on the App Store is *Yomiwa* by Vivien Seguy[12] whose first version was released in 2013. It is frequently updated with improved performance and its recognition capabilities are very good. Neither *Japan Goggles*, nor *Yomiwa* requires internet connection to perform character recognition and both applications have got positive ratings.

There are some mobile phones on the Japanese mobile phone market which have built-in OCR systems, but their performance is limited [6]. Since these mobile phones can only be used in Japan and are aimed at the Japanese, they only recognize a character or a compound of characters and provide the user with their reading together with an explanation of meaning of the word in Japanese.

Besides mobile applications for Japanese OCR mentioned above, there are also free OCR engines available, which can be used for development of OCR applications. They either support multiple languages or are made especially for recognition of text in Japanese. Two of them will be used for OCR engine performance comparison in this work.

- The first one is **NHocr**[13]. NHocr is an OCR engine designed for recognition of machine-printed Japanese characters. According to the author, NHocr may be the first open-source Japanese OCR engine. Its development began in 2008 and the author says that it is a product of his weekend programming. NHocr uses Peripheral Local Moment (P-LM) for feature extraction proposed by Hori et al. in the late 90's in [7].

- The second one is **Tesseract**[14], an open-source OCR engine sponsored by Google. It was dormant between 1995 and 2006, but it has been largely improved since then and it has become one of the best known free OCR engines. It offers multi-language support, provided it is used with data trained for the required language, and there are many datafiles with trained data available for download on the website of Tesseract, including a trained datafile for Japanese.

1.3 Japanese Writing System

The modern Japanese writing system consists of a combination of three scripts: Chinese characters, usually called kanji, and two syllabaries, called hiragana and katakana. Japanese started adopting the Chinese characters along with other cultural aspects in the 4th century [8] because they lacked a writing system of their own. At first, Chinese characters were used for writing in Classical Chinese and only later a system for writing Japanese using Chinese characters was developed. The system called *man'yōgana* [8] designated a phonetic value derived from Chinese readings to kanji instead of their semantic meanings. The modern syllabaries hiragana and katakana are simplifications of *man'yōgana* [9].

A lot of new words and concepts which Japanese language had no equivalent for came to Japan from China, thus a lot of words entered Japanese directly with pronunciations similar to the original Chinese ones. The reading derived from Chinese is known as *on'yomi* (lit. sound reading) and words using this reading are classified as Sino-Japanese. However, the native Japanese language included a lot of words that were equivalent by their meaning to kanji borrowed from Chinese and literate people started using kanji to represent these words. This reading is referred to as *kun'yomi* and is used together with *on'yomi* in modern Japanese. A kanji can have none, one or multiple *kun'yomi* and *on'yomi* readings and the reading depends on

[12]Yomiwa - `http://www.yomiwa.net/`, `https://itunes.apple.com/us/app/yomiwa-japanese-camera-translator/id670931120?mt=8`
[13]`https://code.google.com/p/nhocr/`
[14]`https://code.google.com/p/tesseract-ocr/`

あ　ア　枝

Figure 1.1: *Examples of hiragana (left), katakana (middle), and kanji (right) characters.*

the word in which it is used. For example, the character 通 is read *tō* in the word 通る (*tōru*, to go by), *kayo* in the word 通う (*kayou*, to attend (school, etc.)), *dō* in the word 裏通り (*uradōri*, side street) and *tsū* in the word 通貨 (*tsūka*, currency). In modern Japanese, kanji are usually used for words that carry the content of text:

- nouns, e.g. 女 (*onna*, woman), 音楽 (*ongaku*, music).
- stems of verbs and adjectives, e.g. 話 in 話す (*hanasu*, to speak), 詳 in 詳しい (*kuwashii*, detailed)
- personal and place names, e.g. 佐藤 (Satō), 京都 (Kyōto). However, certain names are written in hiragana, katakana or combination of syllabaries and kanji.

The roots of modern hiragana and katakana syllabaries go back to the 9th century. Both syllabaries have undergone changes over time and were finally codified in 1900 [9]. Hiragana consists of 48 unique characters, 2 of which are obsolete in contemporary Japanese. The use of diacritic marks and digraphs (syllables consisting of two unique characters) extends this set to 94 syllables. Hiragana is usually used for these script elements:

- native or naturalized Japanese words which lack a kanji or their kanji equivalent is not in common use.
- *okurigana* (送り仮名) - grammatical patterns for inflectional endings in verbs and adjectives, e.g. える in 変える (*kaeru*, to change) and its corresponding past tense form えた in 変えた (*kaeta*, changed), い in 薄い (*usui*, thin) and its past tense form かった in 薄かった (*usukatta*, was thin).
- *joshi* (助詞) - grammatical particles marking sentence topics, subjects and objects, having a purpose similar to prepositions in English. E.g. を in 彼を (*kare wo*, him) to describe the sentence object.
- *furigana* (振り仮名) - phonetic representation of kanji placed above or beside the kanji character. Usually used in books for children and non-native speakers, alternatively when a kanji character is uncommon or its intended meaning is unusual. E.g. ろうそく over $\overset{\text{ろうそく}}{\text{蝋燭}}$ (*rōsoku*, candle).

Katakana comprises the same number of characters as hiragana, that is 48 unique characters, 2 of which are obsolete and 1 of which is uncommon in modern Japanese. Similarly to hiragana, katakana is extended by diacritic marks and digraphs to represent 94 distinct syllables. Katakana is commonly used for:

- transliteration of foreign words and names. E.g. パソコン (*pasokon*, personal computer), カリフォルニア (*kariforunia*, California)

- names of animals and plants which have no equivalent in kanji or the equivalent is not in common use, e.g. イルカ (*iruka*, dolphin).
- emphasis on word, similar to capitalization in latin alphabet.
- phonomimes (words mimicking actual sounds), e.g. ケロケロ (*kerokero*, ribbit - frog sound), phenomimes (words depicting non-auditory senses), e.g. キラキラ (*kirakira*, being sparkling), and psychomimes (words describing psychological states), e.g. ワクワク (*wakuwaku*, being excited).

Modern Japanese also uses Latin alphabet and Arabic numerals for abbreviations, foreign names and phrases, or to invoke a foreign flavor. However, since the aim of the proposed application is to help the user understand a Japanese word which they cannot read, Latin alphabet and Arabic numerals are not crucially important and are not included in the set of supported characters for testing of optical character recognition methods which will be introduced in the following chapters.

1.4 Thesis Structure

- Chapter 1, Introduction, gives a brief review of history of OCR and explains the complexity of Japanese writing system.
- Chapter 2, Dataset Generation, describes the creation of the training data and the set of characters which are to be supported and possible to recognize by the OCR engine.
- Chapter 3, Preprocessing and Feature Extraction, describes the transformations which an image of a character undergoes before it is used for training or before it is classified. Extraction of features from the image is explained as well.
- Chapter 4, Learning and Classification, introduces procedures and classification methods which have been tested and compared, and presents the results that have been achieved.
- Chapter 5, Text Extraction, explains the methods for extraction of pure text from a captured picture which have been tested and used.
- Chapter 6, Implementation, gives information about the implementation of the mobile application and execution of training and testing of various methods introduced in other chapters.
- Chapter 7, Conclusion, recaps and summarizes what has been performed and accomplished, and what needs to be done in the future.

Chapter 2

Dataset Generation

A program for optical character recognition needs a classifier which will be able to perform recognition and classification of individual characters.

Definition 1 *Classification in a classifier is executed by a classification function $\mathcal{Y}$. It is a function that assigns a class label y_i to an input feature vector x.*

In order to create a classifier, it is necessary to generate a dataset which will be used for its supervised learning.

Definition 2 *Supervised learning is a task where a classification function $\mathcal{Y}$ is derived from labeled training data $\mathbb{T}$. The training data is made of training samples and each sample is a pair of an input object x_i and its desired output value y_i.*

- *Input:* $\mathbb{T} = \{(x_1, y_1), \dots (x_n, y_n)\}$
- *Output:* $\mathcal{Y}$

The dataset will consist of pairs of images of individual characters and their unicode hex code numbers. Only the characters in the dataset used for training of a classifier can be recognized. It is impossible for a classifier to recognize something that it does not know. This chapter describes the creation of a dataset.

2.1 Supported Characters

To make a dataset, samples of each character that the classifier should recognize have to be acquired. It is clear that the classifier has to be able to recognize every character of hiragana and katakana syllabaries, except the obsolete ones. However, it is impossible to state an exact number of kanji used in modern Japanese since there is a large number of kanji carrying a meaning in Japanese which are not in common use. The Japanese Ministry of Education, Culture, Sports, Science and Technology manages a list of kanji which students after 12 years of education are expected to understand, called *jōyō kanji* (lit. kanji for everyday use) [10]. The current list comprises 2,136 characters, which is 191 more than prior to the latest update issued in 2010[1]. However, you can easily encounter a character that is not on the list of *jōyō kanji*, and some characters absent in this list can be actually seen more often than certain characters included on the list. Therefore, creating a list of kanji which should be supported by an optical recognition application is, in itself, a task which requires a certain amount of effort to be put into.

[1]196 new characters were added and 5 characters were removed from the list. `https://en.wikipedia.org/wiki/J%C5%8Dy%C5%8D_kanji`

Using only the kanji in the *jōyō kanji* list would cover most kanji you can see on an everyday basis, but it would be far from exhaustive. The Japanese encoding standard JIS X 0208[2] features two lists of kanji. There are 2,965 characters in the level 1 set, which includes the more common kanji, and another 3,390 characters in the level 2 set, containing the less frequent ones. However, although the level 2 set consists mostly of kanji which are not in common use, there are quite a few examples of characters in the level 2 set which are not exactly uncommon. Therefore, opting for the level 1 set, which is what some researchers engaged in Japanese OCR do for experiments [11], would not be the best option for an application whose goal is to be practical for use.

In the end, I have decided to use the list of 2,500 most frequent kanji in newspapers [12] derived from the KANJIDIC [13] file as the basis for the list of supported characters. It takes into account only the usage in newspapers, which makes it biased towards the kanji likely to be found in newspapers; however, it provides much better and much more practical foundation than the list of *jōyō kanji* or the level 1 set of JIS X 0208 standard. The list was, then, extended by the *jōyō kanji* which are not listed among the 2,500 most frequent kanji in newspapers, adding up to 2,599 characters. After that, I proceeded to manual selection from the *jinmeiyō kanji* list (lit. kanji for use in personal names) and from kanji which do not appear on the *jinmeiyō kanji* list, *jōyō kanji* list and the list of 2,500 most frequent kanji in newspapers, but often appear in literature, specialized texts, and other contents where less common kanji do appear. The finalized list consists of 73 hiragana characters with and without diacritic marks, 73 katakana characters with and without diacritic marks and 2,762 kanji characters, summing up to 2,908 characters in total.

2.2 Character Image Generation

It was essential to acquire images of all characters that are to be supported to prepare the training data. 82 free digital font files were used for this purpose and bitmap images of individual characters were extracted from these files by means of Batik Java library[3] for manipulation with scalable vector graphics. The extracted images are grayscale and all further processing is performed in 8-bit grayscale color depth to prevent undesirable effect of aliasing of character strokes which would have a negative influence on the recognition performance because feature extraction in this work is based on stroke direction. With the effect of aliasing, the stroke directions would be distorted and the extracted information would not be sufficient enough.

The final training dataset consists of pairs of character images and Unicode hex code numbers that specify what class the character belongs to, which means what character is in the image in this case.

[2]`https://en.wikipedia.org/wiki/JIS_X_0208`

[3]Apache™ Batik SVG Toolkit - `http://xmlgraphics.apache.org/batik/`

Chapter 3
Preprocessing and Feature Extraction

Every image in the dataset of character images is processed to acquire a feature vector which can be used for training of a classification model. The image processing consists of several phases. First, the images are normalized on the gray scale so that their pixel intensities span the whole color space. Next, the shape of the character in the image is normalized in order to reduce differences between various images of the same character, which will make their feature vectors closer to each other in the feature space and improve separability of each class. Gaussian filtering is performed after the shape normalization to smooth rough and jagged stroke edges. Finally, a feature vector which will be further used for training of a classifier is extracted from the image. The same process is applied when extracting a feature vector from an image of a character which is to be classified. Each phase of the process is described in detail in the following sections.

3.1 Grayscale Normalization

The grayscale image is normalized so that the intensity values of its pixels range from 0 to 255 and utilize the whole 8-bit space [14], thus enhancing the contrast between the text and the background. Linear normalization is applied to achieve this. Let $A(m,n)$ be an input image, whose size is $M \times N$, $m = \{1, 2, \dots, M\}$, and $n = \{1, 2, \dots, N\}$. The output image $B(m,n)$ is defined as:

$$B(m,n) = \frac{255}{A_{max} - A_{min}} (A(m,n) - A_{min}) \tag{3.1}$$

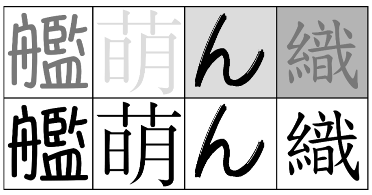


Figure 3.1: *Examples of grayscale normalization performed on different images.*

3.2 Shape Normalization

Shape normalization is performed by mapping the input image on an image plane of a given size. Let $A(x,y)$ be the input image and $B(x,y)$ the normalized image. For 1D normalization methods, each coordinate (x, y) from the input will be mapped to a coordinate (x', y') in the output by a mapping function:

$$x' = f(x)$$
$$y' = g(y)$$

This is called forward mapping. Alternatively, it is possible to use backward mapping and map each coordinate (x', y') in the normalized output image to coordinate (x, y) in the input image:

$$x = f^{-1}(x')$$
$$y = g^{-1}(y')$$

I have implemented and tested three normalization methods: linear normalization (LN), moment normalization (MN), and bi-moment normalization (BMN). MN and BMN for Chinese characters were first described by Liu et al. [15]

3.2.1 Aspect Ratio Adaptive Normalization

The final image for feature extraction will be square; however, in order to preserve the original aspect ratio of a character up to some extent and reduce the deformation of oblong characters, aspect ratio adaptive normalization (ARAN) [16] is adopted to calculate the output width and height of the characters for shape normalization. Let W_1 be the width of the input image and H_1 be the height thereof. The aspect ratio R_1 is defined as:

$$R_1 = \begin{cases} \frac{H_1}{W_1}, & \text{if } H_1 < W_1 \\ \frac{W_1}{H_1}, & \text{otherwise} \end{cases}$$

If W_1 is bigger than H_1, W_2, the width of the output image, is set deliberately. Otherwise, H_2, the height of the output image is specified. The aspect ratio R_2 of the output image and the size of the other side is calculated by:

$$R_2 = \sqrt{\sin(\frac{\pi}{2} R_1)}$$
$$\begin{cases} H_2 = W_2 R_2, & \text{if } H_1 < W_1 \\ W_2 = H_2 R_2, & \text{otherwise} \end{cases} \tag{3.2}$$

3.2.2 Linear Normalization

The first normalization method I have implemented is linear normalization [17]. Linear normalization aligns the boundaries of the input image to the boundaries of the output image. It linearly shrinks or extends the image of size $W_1 \times H_1$ in both dimensions to match the desired size $W_2 \times H_2$. Let A be the input image and B the normalized output image. The forward mapping functions which map each pixel $A(x, y)$ in the input image to a pixel $B(x', y')$ in the normalized image are then defined as:

$$\begin{aligned} x' &= \frac{W_2}{W_1} x \\ y' &= \frac{H_2}{H_1} y \end{aligned} \tag{3.3}$$

Since mapping described in 3.3 is not guaranteed to be surjective, to make sure that each pixel in the normalized image receives a value from the input and blank space is not generated, backward mapping is performed instead. Each pixel $B(x', y')$ in the normalized output image is mapped to a pixel $A(x, y)$ in the input image and receives its value. The mapping functions are as follows:

$$\begin{aligned} x &= \tfrac{W_1}{W_2}x' \\ y &= \tfrac{H_1}{H_2}y' \end{aligned} \tag{3.4}$$

3.2.3 Moment Normalization

Since the strokes of a character are sometimes, particularly in handwriting, not evenly spread and are accumulated in one part of the character, shifting the centroid of character strokes from the geometric center of the character, the visual information about the character gets condensed in a smaller area and the shape deviates from the standard shape of character used in print publications. Stretching the condensed part of the character would spread the visual information and make the shape resemble the standard shape more. Linear normalization cannot do that, because it can only linearly change the shape, which means that it cannot stretch one part of the character more than the rest. Therefore, a non-linear method is needed to perform this task.

The moment normalization method [15] aligns the centroid[1] of the input image $(c_x,\, c_y)$ to the geometric center of the normalized output image $(x'_c,\, y'_c)$. The geometric center of output image is defined as $(x'_c,\, y'_c) = (\frac{W_2}{2},\, \frac{H_2}{2})$. The width W_1 and height H_1 of the input image are replaced by modified width δ_x and height δ_y in the calculations. δ_x and δ_y are derived using the second order 1D central moments μ_{20} and μ_{02}:

$$\begin{aligned} \delta_x &= 4\sqrt{\mu_{20}} \\ \delta_y &= 4\sqrt{\mu_{02}} \end{aligned}$$

Consequently, the boundaries of the input image $(0,\, W_1)$ and $(0,\, H_1)$ are shifted to $(c_x - \frac{\delta_x}{2},\, c_x + \frac{\delta_x}{2})$ and $(c_y - \frac{\delta_y}{2},\, c_y + \frac{\delta_y}{2})$. The definition of central moment [18] is given by:

$$\mu_{ij} = \sum_{x=1}^{X}\sum_{y=1}^{Y}(x - c_x)^i (y - c_y)^j f(x, y) \tag{3.5}$$

In this case, the function $f(x, y)$ denotes the pixel intensity $A(x, y)$ at position $[x,\, y]$ in the image. Using the definition to calculate the central moment is not the most efficient method to do so. It is possible to calculate central moments from several raw moments, which are easier to calculate. Raw moments M_{ij} for discrete 2D functions are calculated by:

$$M_{ij} = \sum_{x=1}^{X}\sum_{y=1}^{Y} x^i y^j f(x, y) \tag{3.6}$$

The centroid $(c_x,\, c_y)$ of an image can be calculated by the following formulas, where M_{00} is the

[1]Centroid refers to the centroid of character stroke pixels here.

sum of pixel intensities over the whole image.

$$\begin{aligned} c_x &= \frac{\sum_{x=1}^{X}\sum_{y=1}^{Y} xA(x,y)}{\sum_{x=1}^{X}\sum_{y=1}^{Y} A(x,y)} \\ &= \frac{M_{10}}{M_{00}} \\ c_y &= \frac{\sum_{x=1}^{X}\sum_{y=1}^{Y} yA(x,y)}{\sum_{x=1}^{X}\sum_{y=1}^{Y} A(x,y)} \\ &= \frac{M_{01}}{M_{00}} \end{aligned} \tag{3.7}$$

Starting with the definition of the second order central moment, one can derive a formula consisting solely of raw moments.

$$\begin{aligned} \mu_{20} &= \sum_{x=1}^{X}\sum_{y=1}^{Y}(x-c_x)^2(y-c_y)^0 A(x,y) \\ &= \sum_{x=1}^{X}\sum_{y=1}^{Y}(x-c_x)^2 A(x,y) \\ &= \sum_{x=1}^{X}\sum_{y=1}^{Y}(x^2-2xc_x+c_x^2)A(x,y) \\ &= \sum_{x=1}^{X}\sum_{y=1}^{Y} x^2 A(x,y) + \sum_{x=1}^{X}\sum_{y=1}^{Y} -c_x(2x-c_x)A(x,y) \\ &= M_{20} - c_x\sum_{x=1}^{X}\sum_{y=1}^{Y}(x+x-c_x)A(x,y) \\ &= M_{20} - c_x\left(\sum_{x=1}^{X}\sum_{y=1}^{Y} xA(x,y) + \sum_{x=1}^{X}\sum_{y=1}^{Y}(x-c_x)A(x,y)\right) \\ &= M_{20} - c_x\left(\sum_{x=1}^{X}\sum_{y=1}^{Y} xA(x,y) + 0\right) \\ &= M_{20} - c_x M_{10} \end{aligned} \tag{3.8}$$

The derivation of the final formula for the second order central moment μ_{02} using the raw moments is analogous, only the x coordinates and centroid coordinates component c_x are replaced with y and c_y, respectively. The coordinate mapping for moment normalization is then calculated by:

$$\begin{aligned} x' &= \frac{W_2}{\delta_x}(x-c_x)+x'_c \\ y' &= \frac{H_2}{\delta_y}(y-c_y)+y'_c \end{aligned} \tag{3.9}$$

For the same reason as in linear normalization, backward mapping is used to set a value to each pixel of the normalized image. Coordinates $(x,\ y)$ in the original image will be calculated by:

$$\begin{aligned} x &= \frac{\delta_x(x'-x'_c)}{W_2}+c_x \\ y &= \frac{\delta_y(y'-y'_c)}{H_2}+c_y \end{aligned} \tag{3.10}$$

The coordinates $(x,\ y)$ range from $(c_x - \frac{\delta_x}{2},\ c_y - \frac{\delta_y}{2})$ to $(c_x + \frac{\delta_x}{2},\ c_y + \frac{\delta_y}{2})$, as the boundaries of the input image have been shifted. To retrieve a pixel from the input image, it is necessary to calculate the original coordinates. Let $(x_o,\ y_o)$ be the original coordinates in range from (0, 0) to $(W_1,\ H_1)$, and $(x_s,\ y_s)$ be the shifted coordinates from range $(c_x - \frac{\delta_x}{2},\ c_y - \frac{\delta_y}{2})$ to $(c_x + \frac{\delta_x}{2},\ c_y + \frac{\delta_y}{2})$. It is obvious that $f(c_x - \frac{\delta_x}{2}) = 0$, $f(c_x + \frac{\delta_x}{2}) = W_1$, and also $f(c_x) = c_x$. Using the

Lagrange polynomial for interpolation, the coordinate x_o is calculated by:

$$\begin{aligned}
x_o &= L_x(x_s) = f(x_1)\cdot\tfrac{x_s-x_2}{x_1-x_2}\cdot\tfrac{x_s-x_3}{x_1-x_3} + f(x_2)\cdot\tfrac{x_s-x_1}{x_2-x_1}\cdot\tfrac{x_s-x_3}{x_2-x_3} + f(x_3)\cdot\tfrac{x_s-x_1}{x_3-x_1}\cdot\tfrac{x_s-x_2}{x_3-x_2} \\
&= c_x\cdot\tfrac{x_s-(c_x-\frac{\delta_x}{2})}{c_x-(c_x-\frac{\delta_x}{2})}\cdot\tfrac{x_s-(c_x+\frac{\delta_x}{2})}{c_x-(c_x+\frac{\delta_x}{2})} + W_1\cdot\tfrac{x_s-(c_x-\frac{\delta_x}{2})}{c_x+\frac{\delta_x}{2}-(c_x-\frac{\delta_x}{2})}\cdot\tfrac{x_s-c_x}{c_x+\frac{\delta_x}{2}-c_x} \\
&= c_x\cdot\tfrac{x_s-c_x+\frac{\delta_x}{2}}{\frac{\delta_x}{2}}\cdot\tfrac{x_s-c_x-\frac{\delta_x}{2}}{-\frac{\delta_x}{2}} + W_1\cdot\tfrac{x_s-c_x+\frac{\delta_x}{2}}{\delta_x}\cdot\tfrac{x_s-c_x}{\frac{\delta_x}{2}} \\
&= 2W_1\tfrac{x_s^2-2x_sc_x+c_x^2+\frac{\delta_x}{2}(x_s-c_x)}{\delta_x^2} - 4c_x\tfrac{x_s^2+c_x^2-\frac{\delta_x^2}{4}-2x_sc_x}{\delta_x^2}
\end{aligned} \tag{3.11}$$

Similarly, one can derive the function for coordinate y:

$$y_o = L_y(y_s) = 2H_1\frac{y_s^2-2y_sc_y+c_y^2+\frac{\delta_y}{2}(y_s-c_y)}{\delta_y^2} - 4c_y\frac{y_s^2+c_y^2-\frac{\delta_y^2}{4}-2y_sc_y}{\delta_y^2} \tag{3.12}$$

3.2.4 Bi-moment Normalization

In a similar fashion to moment normalization, the bi-moment normalization [15] method aligns the centroid of the image to the geometric center of the output image. Whilst the moment normalization method calculates one central moment for x-coordinate and one for y-coordinate and resets the image boundaries so that the centroid is in the middle of the computed interval, bi-moment normalization method utilizes four central moments, two for each coordinate. For x-coordinate, one central moment is calculated in the part of the image left of the centroid, and the other one for the part on the right. Similarly, for y-coordinate, one central moment is calculated in the part below the centroid, and the other one above the centroid. The width and height are not treated as symmetrical around the centroid.

$$\begin{aligned}
\mu_x^- &= \textstyle\sum_{x<c_x}\sum_{y=1}^{Y}(x-c_x)^2A(x,y), \\
\mu_x^+ &= \textstyle\sum_{x>c_x}\sum_{y=1}^{Y}(x-c_x)^2A(x,y), \\
\mu_y^- &= \textstyle\sum_{y<c_y}\sum_{x=1}^{X}(y-c_y)^2A(x,y), \\
\mu_y^+ &= \textstyle\sum_{y>c_y}\sum_{x=1}^{X}(y-c_y)^2A(x,y).
\end{aligned} \tag{3.13}$$

The image boundaries $(0, W_1)$ and $(0, H_1)$ are shifted to $(c_x-\delta_x^-, c_x+\delta_x^+)$ and $(c_y-\delta_y^-, c_y+\delta_y^+)$, where

$$\begin{aligned}
\delta_x^- &= 2\sqrt{\mu_x^-}, \\
\delta_x^+ &= 2\sqrt{\mu_x^+}, \\
\delta_y^- &= 2\sqrt{\mu_y^-}, \\
\delta_y^+ &= 2\sqrt{\mu_y^+}.
\end{aligned}$$

Mapping is performed identically to moment normalization as is shown in equation (3.10). However, as the centroid of the image is not in the middle of the interval as in the moment method, mapping from the output image to the bi-moment coordinates is not linear, thus it is necessary to perform interpolation twice to map the output image coordinates to the bi-moment coordinate system, and the bi-moment coordinates to the original input image coordinates.

In Figure 3.2, you can see examples of shape normalization by means of linear, moment, and bi-moment normalization methods. The first row shows the original trimmed characters in a square box, the second row contains the same characters after linear normalization, the third row are shape-normalized characters using moment normalization, and characters in the fourth row have been normalized by the bi-moment method. The first example is very similar in all four cases, because it uses a standard font which you could find for example in books and newspapers. The second example shows that moment-based normalization has a huge advantage over linear normalization in case that there remains some undesirable noise in the image. Small noise or a blot cannot alter the centroid of the image very much, therefore the centroid in the original image is shifted heavily to the right side and is located approximately in the middle of the actual character. Moment and bi-moment normalization transforms the image so that the centroid is in the geometric center of the output image. Both linear and moment-based normalization methods are global transformations and have a desirable property that the shape of the character is not heavily deformed by the transformation, which could happen if a local transformation method was used instead [15].

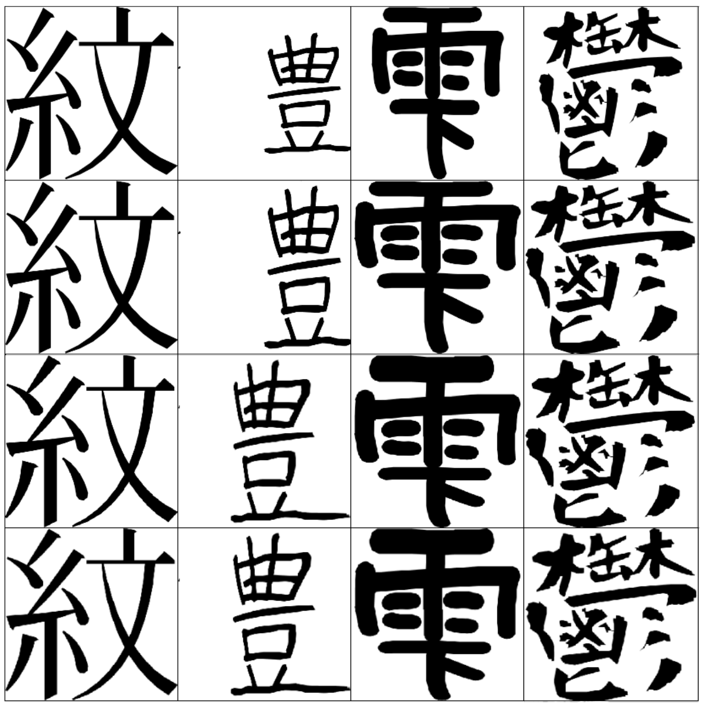


Figure 3.2: *Examples of shape normalization performed on different images. First row: cropped original images fit into square; Second row: images after linear normalization; Third row: images after moment normalization.*

3.2.5 Smoothing

Gaussian blur is applied after shape normalization in order to smooth character strokes and alleviate the effect of ruggedness, which could be present in the original image of a character or might be caused by shape normalization. The filter is applied by convolving the image with a Gaussian function [19], [20]. One-dimensional Gaussian function without the normalization constant

$$G(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{3.14}$$

is used to fill the kernel for convolution. The size of the kernel is set to $\lceil 6\sigma + 1 \rceil$ ($\lceil \cdot \rceil$ is the ceiling function), where σ denotes the standard deviation of the Gaussian distribution, and the values are normalized so that their sum is unity. The size of the kernel is decided by the aforementioned rule so that the kernel coefficients make up most of the sum of coefficients over the Gaussian function. In other words, increasing the size of the kernel would have virtually no effect on the result, because the coefficients in the added area would be small enough to be considered zero as is shown in the following figure. Let σ be 1. Then, the 1D kernel is

0.004	0.054	0.242	0.399	0.242	0.054	0.004

2D Gaussian blur is a separable filter, which means it can be expressed as a product of two vectors [21]. Therefore, it can be applied as two independent one-dimensional operations for a 2D image. First, the image is blurred horizontally / vertically, and then it is blurred in the other way. Using this approach for filtering is more efficient in computational terms as it can be achieved in $O(W_{ker} \cdot W_{img} \cdot H_{img}) + O(H_{ker} \cdot W_{img} \cdot H_{img})$ time, while the two-dimensional approach takes $O(W_{ker} \cdot H_{ker} \cdot W_{img} \cdot H_{img})$ time.

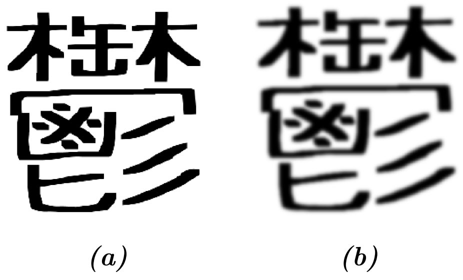


Figure 3.3: *Example of an image before applying a Gaussian filter of $\sigma = 2$ on the left and after filtering on the right.*

3.3 Feature Extraction

Most kanji are composed of several parts which may be an existing kanji themselves, see Figure 3.4, or they at least reappear in multiple characters. Patterns that appear frequently are called radicals and they have been traditionally used for finding kanji characters in dictionaries. The six most common radicals make up as much as 25% of *jōyō kanji*[2]. It might be possible to make use of the fact that most kanji consist of several patterns and turn the process of recognition of a complex character into a task of recognizing several simpler patterns. However, it would be necessary to correctly divide the character into several distinct patterns, which would be a demanding task because there is no rule as to how many common patterns can be found in a kanji and how to divide

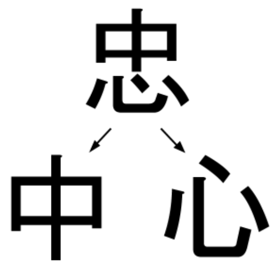


Figure 3.4: *An example of a character consisting of multiple existing kanji. Character 忠 - meaning loyalty, fidelity - is composed of two existing kanji characters: 中 - meaning center, inside - and 心 - meaning heart, mind.*

[2]`https://en.wikipedia.org/wiki/Table_of_Japanese_kanji_radicals`

the kanji to extract them. Therefore, it sounds more reasonable to adopt a more conservative statistical approach.

3.3.1 Zonal Line Density Method

The first algorithm for feature extraction which I have implemented - building on introduction to OCR approaches in a work by Line Eikvil [2] - and tried out is a zonal approach in which the character is divided into several blocks and features are calculated in each of these blocks separately. The algorithm calculates the density of pixel lines and outputs these measurements as a feature vector that can be used for training of a classifier and subsequently classification of new samples. The process is described below:

(1) Resize the grayscale image to 120 × 120 pixels and add 4 pixel large margins on each side.

(2) Normalize the pixel intensities such that their mean is 0 and they have unit variance. Let $A(x, y)$ be the input matrix and $B(x, y)$ the output matrix. The output $B(x, y)$ is defined as:

$$B(x,y) = \frac{A(x,y) - \overline{A}}{\sqrt{\frac{1}{MN-1}\sum_{m=1}^{M}\sum_{n=1}^{N}(A(m,n) - \overline{A})^2}} \tag{3.15}$$

(3) Divide the image into 64 blocks of 16 × 16 pixels in size, that is 8 blocks in a row and 8 blocks in a column.

(4) Sum up the pixel values in 1 × 16 pixel columns in every block and divide it by 16. This will yield 1,024 values which will be used as a feature vector. Let a_{ij} be a pixel in a block whose value ranges from 0.0 to 1.0.

$$v_j = \frac{1}{16}\sum_{i=1}^{16} a_{ij}, \quad j = \{1, 2, \dots, 16\} \tag{3.16}$$

(5) Sum up the pixel values in every row and every column of the whole 128 × 128 pixel large image and divide it by 128. This will yield another 256 values for a feature vector, adding up to 1280-dimensional feature vector.

$$\begin{aligned} c_j &= \frac{1}{128}\sum_{i=1}^{128} a_{ij}, \quad j = \{1, 2, \dots, 128\} \\ r_i &= \frac{1}{128}\sum_{j=1}^{128} a_{ij}, \quad i = \{1, 2, \dots, 128\} \end{aligned} \tag{3.17}$$

(6) Scale the values in the feature vector so that they range from 0.0 to 1.0.

The above algorithm is simple and while its results are not exactly poor, it does not reach the desired recognition rate as will be shown in section Results.

3.3.2 Histogram of Oriented Gradients in Local Regions

A more sophisticated algorithm suggested by Dong et al. [22] uses edge detection and histogram of oriented gradients (HOG) to analyze the stroke characteristics. The algorithm is a modification of a method proposed by Kimura et al. [23] in 1997, which uses a histogram of chain codes in local regions and works on binary images. By applying a similar idea on grayscale

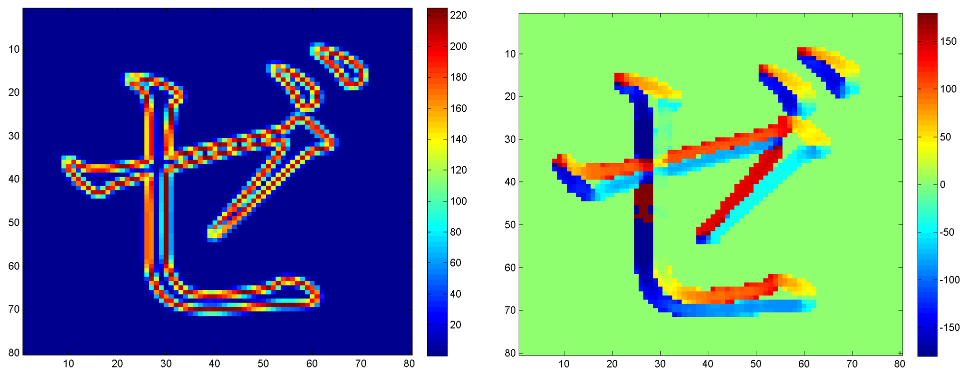


Figure 3.5: Visualized gradient strength of an image of character セ on the left and the gradient direction of the same image on the right.

images using oriented gradients instead of chain codes, Dong et al. were able to achieve better results. This algorithm has been proven to yield very good results on various writing systems [24]. In this work, the algorithm is slightly modified in the way that the pixel intensities are normalized so that they have unit variance and their mean is 0, which provided slightly better results than the approach in the original version of the algorithm, where the pixel intensities are scaled so that they range from 0 to 1. The feature extraction process is as follows:

(1) Resize the grayscale image to 64 × 64 pixels and center it in a 80 × 80 pixel large square by adding 8 pixel margins on each side. This is performed in order to utilize the peripheral strokes as much as the strokes closer to the middle.

(2) Normalize the pixel intensities such that their mean is 0 and they have unit variance (3.15).

(3) Calculate the image gradient strengths and gradient directions in each pixel using Roberts cross operator for edge detection. [25]

First, the original image is convolved with kernels:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} , \quad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Let $A(x,y)$ be a pixel in the original image and $G_1(x,y)$, $G_2(x,y)$ be points produced by convolving the original pixel with the first and second kernel, respectively. The gradient strength ∇ is then defined by the following equation:

$$\nabla A(x,y) = \sqrt{G_1^2 + G_2^2} \tag{3.18}$$

The gradient direction θ is defined as:

$$\theta A(x,y) = \text{atan2}\left(G_2(x,y), G_1(x,y)\right) \tag{3.19}$$

(4) Divide the image into 16 × 16 pixel large overlapping blocks such that there is 9 blocks in each row and column. Each block overlaps exactly a half of its adjacent blocks. There will be 81 blocks in total.

(5) Divide each block into 4 subareas such that subarea S1 is 4 × 4 pixels in the middle of the block, subarea S2 is 8 × 8 pixels in the middle exclusive of S1, subarea S3 is 12 × 12 pixels exclusive of S1 and S2, and subarea S4 is the whole 16 × 16 pixel block exclusive of S1, S2, and S3. See Figure 3.6.

(6) In each block, sum up the gradient strengths over 32 quantized gradient directions in each subarea S1, S2, S3, S4, creating a histogram of oriented gradients. Use a mask vector (4, 3, 2, 1) for the subareas to eliminate the difference in area size of each subarea S1, S2, S3, S4. The result will be 9 × 9 32-dimensional vectors x_k.

$$\begin{aligned} a_i &= \nabla A(x,y), \quad A(x,y) \in S1 \cup S2 \cup S3 \cup S4 \\ v_i &= \delta A(x,y), \quad A(x,y) \in S1 \cup S2 \cup S3 \cup S4 \end{aligned}$$

$$\begin{aligned} x_k = \sum_{v_i \in I} 4\left(\sum_{a_i \in S1} a_i\right) + 3\left(\sum_{a_i \in S2} a_i\right) + 2\left(\sum_{a_i \in S3} a_i\right) + \sum_{a_i \in S4} a_i, \\ I = \left\langle -180 + (k-1)\left(\frac{360}{32}\right); -180 + k\left(\frac{360}{32}\right)\right\rangle \\ k = \{1, 2, \ldots, 32\} \end{aligned} \tag{3.20}$$

(7) Downsample the 32-dimensional vectors to obtain 16-dimensional vectors using binomial filtering with mask vector (1, 4, 6, 4, 1). Join the generated 16-dimensional vectors into one large 1296-dimensional feature vector (9 × 9 × 16).

(8) Exponentiate each element of the feature vector by 0.4 so that the distribution is closer to Gaussian distribution. [26]

(9) Rescale the values of the elements in the feature vector so that they range from 0.0 to 1.0.

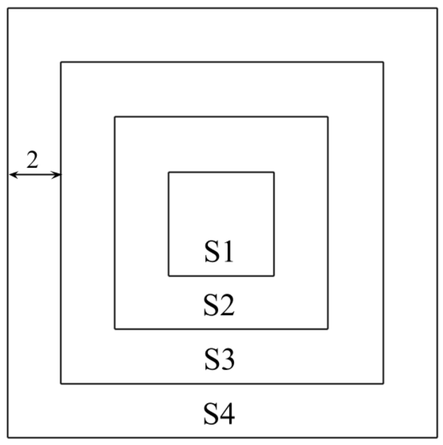


Figure 3.6: *Division of a block into 4 subareas.*

Chapter 4
Learning and Classification

With feature vectors extracted from the images as described in the previous chapter, it is possible to train a classifier for character recognition. Before creating a classifier, an algorithm to reduce the size of feature vectors and improve the performance is applied.

Various combinations of algorithms for reduction of feature vector dimensionality and methods for classification have been tested on the character set described in Chapter titled Dataset Generation. Each class is represented by 82 training samples.

4.1 Dimensionality Reduction

The feature vectors acquired in the feature extraction are high-dimensional; however, majority of the useful information stored in the feature vectors could be represented in a space of much lower dimensionality. Dimensionality reduction algorithms are used to map the data from one space to a lower dimensional space, which results in lower space and computational complexity, and alleviation of the negative effect of the phenomena called ”the curse of dimensionality” [27], which in terms of machine learning refers to reduction of effectiveness of an algorithm due to reduced contrast of distance differences caused by irrelevant dimensions in high-dimensional space.

Most of the engines for OCR of Chinese characters participating in competitions use LDA for reduction of dimensionality and some engines use modified versions of LDA, such as heteroscedastic LDA [4], [5].

4.1.1 Principal Component Analysis

Principal Component Analysis (PCA) [28] is a linear transformation which transforms the data and represents them in a new coordinate system whose first basis vector, the principal component, has the highest variance of data, and the variance of each succeeding principal component is as high as possible under the condition that it is orthogonal to all preceding principal components. The objective is to represent the data in a space L of lower dimension than M while minimizing the approximation error and preserve as much information as possible.

Let N be the number of observations and M be the number of variables describing one observation. Data are arranged in a $M \times N$ matrix $\mathbf{X}$, where each observation is a column of the matrix. Each variable is normalized so that the sum of each row in the matrix is zero, which is accomplished by subtracting the mean of each row from every value in the corresponding row.

$$\begin{aligned} \mu_m &= \tfrac{1}{N}\textstyle\sum_{n=1}^{N} X_{m,n}, \quad m = 1,\dots,M \\ \bar{\mathbf{X}} &= \mathbf{X} - \boldsymbol{\mu} \end{aligned} \tag{4.1}$$

Let us compute the empirical sample covariance matrix $\mathbf{\Sigma}$ from the dataset matrix,

$$\mathbf{\Sigma} = \bar{\mathbf{X}}^T\bar{\mathbf{X}} \tag{4.2}$$

The matrix $\mathbf{\Sigma}$ is guaranteed to be positive semi-definite,

$$\begin{aligned} &\mathbf{a}^T\mathbf{\Sigma}\mathbf{a}, && \mathbf{a} \in \mathbb{R}^m \\ &\mathbf{a}^T(\bar{\mathbf{X}}^T\bar{\mathbf{X}})\mathbf{a} && \\ &(\bar{\mathbf{X}}\mathbf{a})^T(\bar{\mathbf{X}}\mathbf{a}) && \geq \quad 0 \end{aligned}$$

which means that it also has real and positive eigenvalues. Let us compute the eigenvalues λ_i and eigenvectors $\mathbf{w}_i$ of the matrix $\mathbf{\Sigma}$. Next, the eigenvalues are normalized so that their sum is unity and the eigenvectors are normalized so that their length is unity, that is $\|\mathbf{w}_i\|_2 = 1$. Then, the eigenvalues and their corresponding eigenvectors are sorted from the largest to the smallest one. The first eigenvector holds the largest amount of information about the data. A necessary number of eigenvectors will be kept so that the desired percentage of information is preserved and the allowed loss of information is not exceeded. The percentage loss of information ϵ is

$$\epsilon = 1 - \sum_{i=1}^{L} \lambda_i \tag{4.3}$$

Let $\mathbf{W}$ be the matrix of size $M \times L$ with L first eigenvectors sorted by their corresponding eigenvalues and $\bar{\mathbf{X}}$ be the matrix of size $M \times N$ with N observations in columns. The reduced representation of observations $\mathbf{Y}$ is a linear combination of the principal components and is calculated by:

$$\mathbf{Y} = \mathbf{W}^T\bar{\mathbf{X}} \tag{4.4}$$

PCA ignores the class distribution of samples in the dataset and only tries to maximize the variance of each principal component.

4.1.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a method to find a linear combination of features, which best explains the data, and unlike PCA, it tries to model the differences between classes and separate them.

LDA for cases with more than two classes seeks a projection matrix $\mathbf{W}$ such that classes in projection $\mathbf{Y}$,

$$\mathbf{Y} = \mathbf{W}^T\mathbf{X}, \tag{4.5}$$

are separated as best as possible [29], which means that samples from the same class should be projected close to each other and samples from different classes should be as far as possible from each other.

Let us define the within-class scatter matrix $\mathbf{S}_W$,

$$\mathbf{S}_W = \sum_{i=1}^{K} \sum_{\mathbf{x} \in k_i} (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T, \tag{4.6}$$

where $\mathbf{x}$ is a column in the matrix $\mathbf{X}$, K is the number of classes, k_i means that the sample belongs to the i-th class,

$$\boldsymbol{\mu}_i = \frac{1}{N_i} \sum_{\mathbf{x} \in k_i} \mathbf{x}, \tag{4.7}$$

and N_i gives the number of samples in the i-th class. Let us also define the between-class scatter matrix $\mathbf{S}_B$,

$$\mathbf{S}_B = \sum_{i=1}^{K} N_i(\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T, \tag{4.8}$$

where

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{\forall \mathbf{x}} \mathbf{x}. \tag{4.9}$$

The projection that will best discriminate between individual classes can be found by maximizing Fisher's linear discriminant, which maximizes the scatter between classes and minimizes the scatter within classes, given by

$$J(\mathbf{W}) = \frac{|\mathbf{W}^T\mathbf{S}_B\mathbf{W}|}{|\mathbf{W}^T\mathbf{S}_W\mathbf{W}|}. \tag{4.10}$$

After differentiation and equation to zero, the problem requires the solution of

$$\mathbf{S}_W^{-1}\mathbf{S}_B\mathbf{W} = J(\mathbf{W})\mathbf{W} \Rightarrow |\mathbf{S}_W^{-1}\mathbf{S}_B - J(\mathbf{W})\mathbf{I}| = 0, \tag{4.11}$$

where $J(\mathbf{W})$ is a scalar and it is also an eigenvalue of $\mathbf{S}_W^{-1}\mathbf{S}_B$. The best projection vectors of the projection matrix $\mathbf{W}$ are, therefore, the eigenvectors which correspond to the largest eigenvalues of $\mathbf{S}_W^{-1}\mathbf{S}_B$. The eigenvector $\mathbf{w}_1$ corresponding to the largest eigenvalue $\lambda_1 = J(\mathbf{w_1})$ separates the classes in the projection the best of all eigenvectors. The smaller the eigenvalue, the less it can separate the classes in the data. That means that it is possible to keep only a certain number of eigenvectors which can best describe the data in the projection matrix $\mathbf{W}$, which will result in a projection of $\mathbf{X}$ into a lower-dimensional space while separating individual classes as best as possible.

4.2 Classification Methods

Each training example in the training data is a pair consisting of a feature vector extracted from an image and its desired Unicode value output. A supervised learning algorithm is needed to analyze the training data and infer a function therefrom, which can be used to classify new examples whose Unicode value is not known.

When choosing an algorithm, it is important to bear in mind where the classification model is going to be used and what kind of limits are imposed on it. Since the classification model in this case will be used in a mobile application, a light-weight model which does not need a large amount of data and is fast enough to be used for real-time recognition is what is needed. A method which needs all or a large portion of training samples, such as k-nearest neighbors, cannot be used in the application (k-NN was used for comparison purposes). Discriminant function has turned out to be a suitable method, which achieves high recognition rate, allows fast classification, and the size of the resulting data necessary for classification is around 10 megabytes (MB).

Most systems for recognition of Chinese characters seem to use the modified quadratic discriminant function (MQDF) proposed by Kimura et al. [30] in 1987. In recent years, artificial neural networks and support vector machine (SVM) classifier have been also increasingly used. However, the training phase of the regular SVM for a task with a large amount of classes requires a lot of time, as well as the training of a neural network, which was not suitable in this case due to limited computing power of my working environment. The size of the classification models used in leading-edge OCR engines which participate in international competitions appears to be a few hundred megabytes on average, with a classifier produced by Fujitsu presented on the ICDAR 2013 Chinese Handwriting Recognition Competition [4] reaching the size of 2,460 MB. Classifiers of this size are not exactly well suited for use in contemporary mobile devices, even though the data size restrictions, as well as computing power restrictions have been growing weaker in recent years.

The next subsection introduces the classifiers used for testing.

4.2.1 k-Nearest Neighbors

k-Nearest Neighbor is an algorithm which selects k training samples $\mathbf{t}_i$ from the training data $\mathbf{T}$ which are closest in the feature space to a new sample and uses them for classification[28]. The distance can be calculated by various metrics, such as Euclidean distance or Manhattan distance, and the performance of the classifier is dependent on the selection of an appropriate metric.

k-NN does not have a regular training phase; training consists only of storing feature vectors and labels of training samples. All computation is performed in the classification phase. Various metrics can be used to determine the distance in k-NN algorithm. Euclidean distance is used here and it is calculated by:

$$d_e(\mathbf{x}, \mathbf{t}_i) = \sqrt{\sum_{i=1}^{N}(x_i - t_i)^2}, \tag{4.12}$$

where $\mathbf{t}$ is a sample in the training data, $\mathbf{x}$ is the unlabeled data point which is to be classified, and N is the length of the feature vector. k training samples with shortest computed distance from the new data point are selected and the class which is the most frequent among the selected samples is the output used to classify the new data point. Let $\mathcal{Y}$ be a function assigning class labels to feature vectors. Class $\mathcal{Y}(\mathbf{x})$ of a new data point is derived by:

$$\mathcal{Y}(\mathbf{x}) = \text{mode}\{l(\mathbf{t_1}), \dots, l(\mathbf{t_k})\} \tag{4.13}$$

This method needs a supplementary decision rule to determine the class of a new sample in case of a tie. One of the possible rules is that the training sample of one of the classes that are tied which is the closest to the feature vector of a new sample has the final extra vote in the classification process and the output class is selected accordingly. In other words, if two classes are tied, the one whose member is closer to the new sample wins.

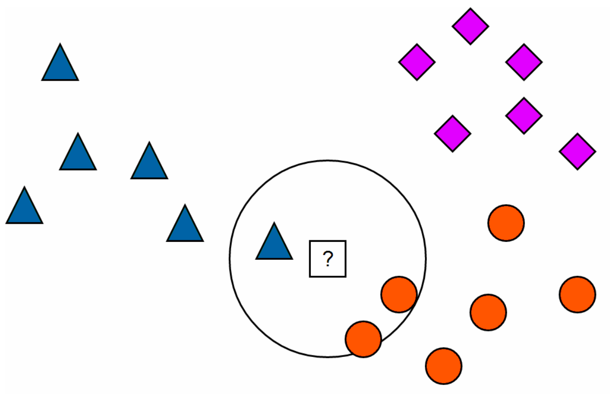


Figure 4.1: An example of k-NN classifier with 3 classes. The number k used is 3. The classifier finds 3 samples closest to the new sample which has to be classified. The class which is prevalent among the 3 selected samples is chosen as the class the new sample belongs to. In this example, the new sample is classified as an orange circle.

4.2.2 Linear Discriminant Function

This and the following subsection introduce the linear discriminant function (LDF) [31] for classification and its quadratic version. The training data is represented by a matrix $\mathbf{T}$ in which each column is the feature vector of one training sample. Let us begin with Bayes' theorem. We want to maximize the posterior probability that a sample belongs to class k. This method is called maximum a posteriori estimation (MAP). The posterior probability is given by

$$P(k|\mathbf{x}) = \frac{P(\mathbf{x}|k)P(k)}{\sum_{i=1}^{K} P(\mathbf{x}|k)P(k)}, \tag{4.14}$$

where $\mathbf{x}$ is the feature vector of a new sample that we want to classify, K is the number of classes, $P(k)$ is the prior probability of class k, and $P(\mathbf{x}|k)$ is the probability density function of class k. The classification function $\mathcal{Y}$ is defined as

$$\mathcal{Y}(\mathbf{x}) = \underset{k=1,2,\dots,K}{\arg\max}\, P(k_i|\mathbf{x}). \tag{4.15}$$

The classifier assumes that each class in the training data has a Gaussian distribution. Let N be the number of training samples and M be the number of variables in a feature vector. The multivariate Gaussian density of a class is given by

$$P(\mathbf{x}|k) = \frac{1}{(2\pi^M|\mathbf{\Sigma}_k|)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_k\right)^T \mathbf{\Sigma}_k^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_k\right)\right), \tag{4.16}$$

where $|\mathbf{\Sigma}_k|$ is the determinant of $\mathbf{\Sigma}_k$, $\boldsymbol{\mu}_k$ are class means defined as

$$\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_{n=1}^{N} C_{kn}\mathbf{t}_n, \tag{4.17}$$

$\mathbf{t}_n$ is a feature vector of one sample from the training data $\mathbf{T}$, N_k is the number of samples of class k, C_{kn} is an element of a membership matrix $\mathbf{C}$ of size K-N and

$$C_{kn} = \begin{cases} 1 & \text{if the n-th sample belongs to class } k \\ 0 & \text{otherwise} \end{cases} \tag{4.18}$$

Matrix $\mathbf{\Sigma}_k$ is the within-group covariance matrix of class k. We suppose that the covariance matrix is identical for all classes and it can be calculated by

$$\mathbf{\Sigma} = \frac{1}{N-K}\sum_{n=1}^{N}\sum_{k=1}^{K} C_{kn}\left(\mathbf{x}_n-\boldsymbol{\mu}_k\right)\left(\mathbf{x}_n-\boldsymbol{\mu}_k\right)^T \tag{4.19}$$

The prior probability of all classes is considered to be equal, $P(k) = \frac{1}{K}$, because the frequency of characters undoubtedly differs depending on the setting in which the recognition is performed and it would be difficult to determine the prior probabilities in a way that would make them plausible. Under equal prior probabilities, the outcome can be determined by maximizing the Gaussian density function.

Probability that a class k will have a member $\mathbf{x}$ is the same as the function of likelihood that $\mathbf{x}$ belongs to a class k. $P(\mathbf{x}|k) = \mathcal{L}(k|\mathbf{x})$. It will be hereinafter referred to as the likelihood function. Since the prior probabilities are omitted and only the likelihood function remains, MAP becomes maximum likelihood estimation (MLE). Instead of the likelihood function, it is possible and more convenient to operate with the natural logarithm of the likelihood function, because logarithm is a monotonically increasing function and the logarithm of a function reaches its maximum in the same point as the function itself. The classifying function becomes a linear discriminant

$$\log \mathcal{L}(k|\mathbf{x}) = -\frac{1}{2}\log|\mathbf{\Sigma}| - M\log\pi - \frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_k\right)^T \mathbf{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_k\right) \tag{4.20}$$

The first and the second term on the right side is identical for all classes, which means it cannot affect the result and can be omitted. The constant $\frac{1}{2}$ before the third term does not have any effect, either. The function can be, therefore, reduced to the following form:

$$\mathcal{Y}(\mathbf{x}) = \underset{k=1,2,\ldots,K}{\arg\max}\ \log\mathcal{L}(k|\mathbf{x}) \equiv \underset{k=1,2,\ldots,K}{\arg\max}\ -\left(\mathbf{x}-\boldsymbol{\mu}_k\right)^T \mathbf{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_k\right), \tag{4.21}$$

The third term in equation 4.20 above is the Mahalanobis distance between a class k and a new unclassified sample $\mathbf{x}$. Therefore, another possible way to interpret this classification function is that it tries to minimize the Mahalanobis distance, because minimizing a function is equivalent to maximizing its additive inverse.

$$d_m = \left(\mathbf{x}-\boldsymbol{\mu}_k\right)^T \mathbf{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_k\right) \tag{4.22}$$

$$\mathcal{Y}(\mathbf{x}) = \underset{k=1,2,\ldots,K}{\arg\min}\ d_m(\mathbf{x},k) \tag{4.23}$$

4.2.3 Quadratic Discriminant Function

Quadratic discriminant function (QDF) [31] is derived in the same way as LDF, but the difference is that each class has its own covariance matrix, which is given by

$$\mathbf{\Sigma}_k = \frac{1}{N-1}\sum_{n=1}^{N} C_{kn}\left(\mathbf{x}_n-\boldsymbol{\mu}_k\right)\left(\mathbf{x}_n-\boldsymbol{\mu}_k\right)^T \tag{4.24}$$

Since each class has its own covariance matrix, it means that it also has its own $\log|\mathbf{\Sigma}_k|$, and the first term in the discriminant function

$$\log \mathcal{L}(k|\mathbf{x}) = -\frac{1}{2}\log|\mathbf{\Sigma}_k| - M\log\pi - \frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_k\right)^T \mathbf{\Sigma}_k^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_k\right) \tag{4.25}$$

cannot be left out. After omitting the second term, canceling common constant, and reverting maximization to minimization, the final function for classification is

$$\mathcal{Y}(\mathbf{x}) = \underset{k=1,2,\ldots,K}{\arg\min}\ \log|\mathbf{\Sigma}_k| + \left(\mathbf{x}-\boldsymbol{\mu}_k\right)^T \mathbf{\Sigma}_k^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_k\right) \tag{4.26}$$

4.3 Results

The following figures display the performance of introduced methods. The tests are performed on the set of 2,908 characters described in previous chapters. LDA is used for dimensionality reduction, HOG for feature extraction, and QDF for classification where not stated otherwise. Samples of two fonts are always set apart for testing and not used in the training phase so that the classifier cannot benefit from training on the data which it has to classify afterwards. This process is repeated four times, always with different two fonts, and the final result is the average of the four error rates. The error rate is calculated as

$$\text{Error} = \frac{\text{Number of incorrect classifications}}{\text{Number of all classified characters}}. \tag{4.27}$$

Figures 4.2 and 4.3 show the performance of recognition using different techniques for normalization of character shape. LDF was used as a classifier for this test. It is evident that the effect of shape normalization on characters in a standard font used in print publications is negligible. It is not surprising, because the examples of shape normalization presented in the previous chapter showed that the shape of a character in a standard print font after any of the normalization methods does not differ very much from the original. However, non-linear shape normalization displays its strength on handwritten characters where both moment and bi-moment normalization outperform linear normalization and recognition with no shape normalization whatsoever. Using bi-moment normalization instead of moment normalization does not seem to result in better performance.

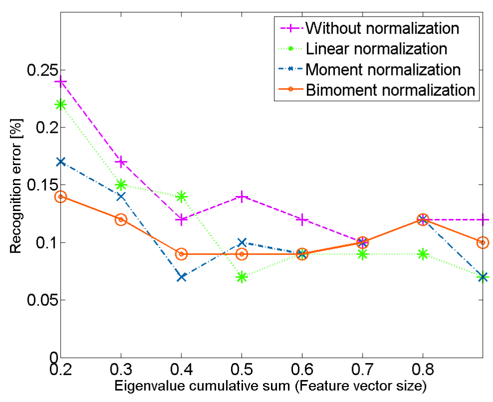


Figure 4.2: *Recognition error rates of tested shape normalization methods with varying sizes of feature vectors. Tested on standard print fonts.*

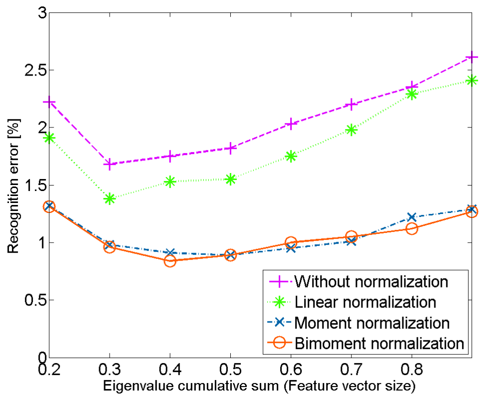


Figure 4.3: *Recognition error rates of tested shape normalization methods with varying sizes of feature vectors. Tested on fonts resembling handwriting.*

Figures 4.4 and 4.5 present the results of recognition by different classifiers. Moment normalization is used for the normalization of character shape in this test. There is virtually no difference between the linear and quadratic discriminant functions on characters in standard print font, but the quadratic version outperforms the linear one on characters whose shape deviates more from the standard form. 3-NN achieves the best recognition rate on handwritten characters, but it requires much more time for classification than LDF and QDF. The difference between the accuracy of 3-NN and QDF is too insignificant to justify the use of 3-NN even if space requirements of the classifier did not have to be considered.

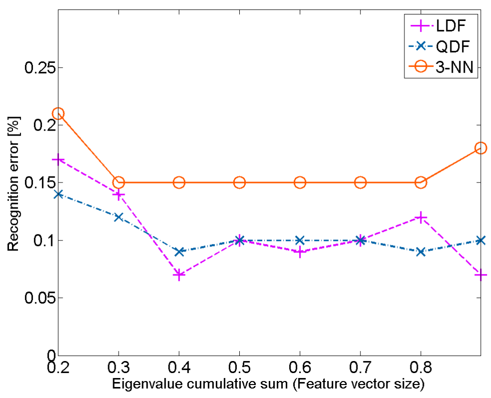


Figure 4.4: *Recognition error rates using different classification methods with varying sizes of feature vectors. Tested on standard print fonts.*

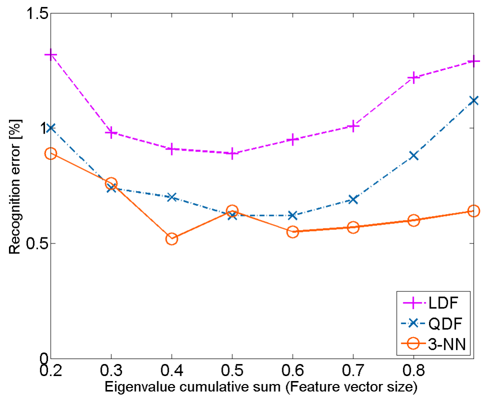


Figure 4.5: *Recognition error rates using different classification methods with varying sizes of feature vectors. Tested on fonts resembling handwriting.*

Figures 4.6 and 4.7 tell us what happens if PCA is used instead of LDA for reduction of feature space dimensionality. The result is not surprising. Since the projection of features into a lower-dimensional space in PCA ignores any differences between classes, the contrast with the performance of LDA is striking.

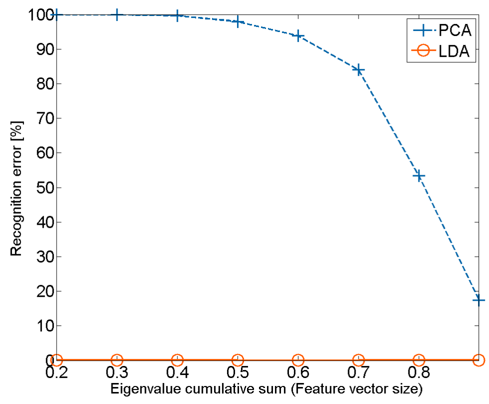


Figure 4.6: *Recognition error rates using different methods for feature dimensionality reduction with varying sizes of feature vectors. Tested on standard print fonts.*

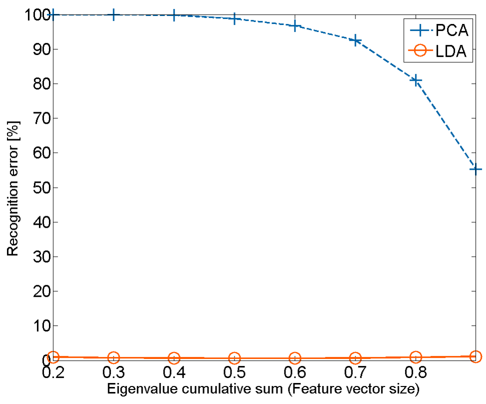


Figure 4.7: *Recognition error rates using different methods for feature dimensionality reduction with varying sizes of feature vectors. Tested on fonts resembling handwriting.*

Finally, Figures 4.8 and 4.9 show the comparison of feature extraction methods. The method using HOG is, as expected, superior in both cases. The Zonal Line Density method I have designed can recognize characters with quite a good precision when the characters are in a standard print font, but since it is not a translation-invariant method, which means that it is largely dependent on the position of pixels in the image, the accuracy of recognition on characters with a shape that deviates from a standard template drops drastically. The other method, which uses HOG, is translation-invariant up to a certain extent, because the gradient magnitudes and directions are the same regardless of the position of the strokes, so if there is a stroke in one of the 16 $\times$ 16 pixel blocks of the image, the stroke can be anywhere in the block and the extracted features will be alike.

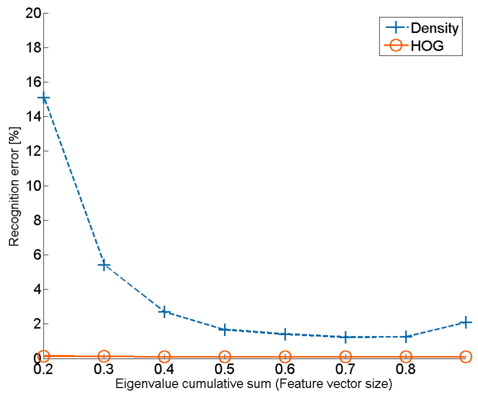


Figure 4.8: *Recognition error rates using different methods for feature extraction. Tested on standard print fonts.*

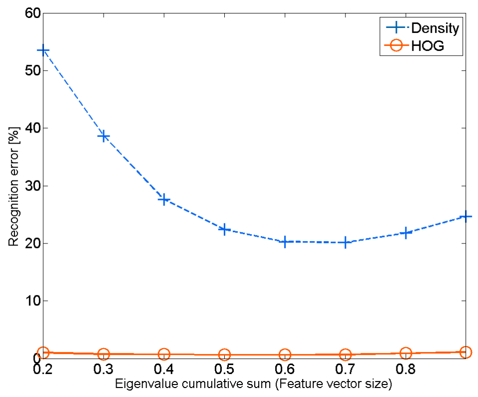


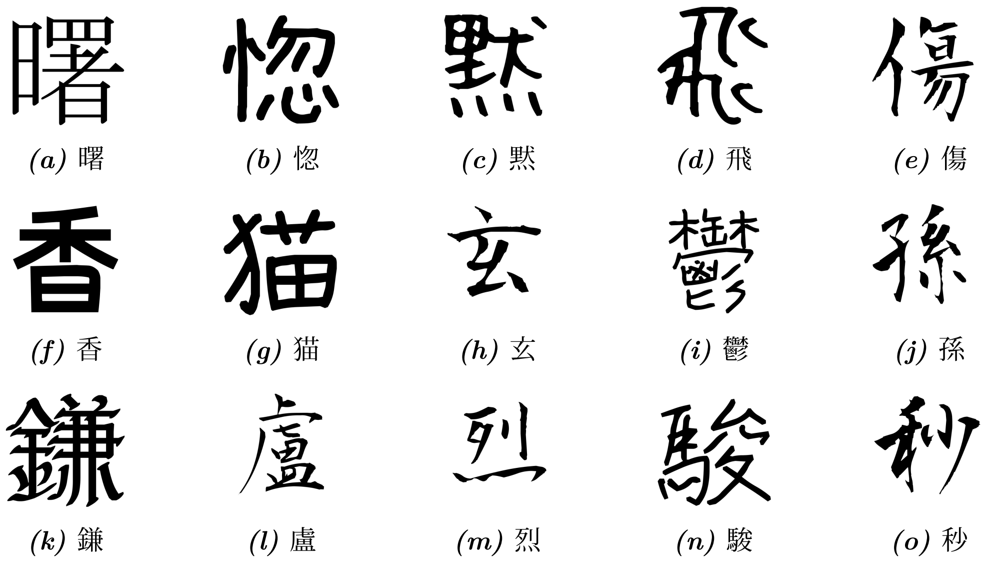
Figure 4.9: *Recognition error rates using different methods for feature extraction. Tested on fonts resembling handwriting.*

Table 4.1 presents the achieved minimum recognition error of all three tested classifiers, using moment method for shape normalization. The feature vectors in all cases had the size of about 200 dimensions.

	LDF	QDF	3-NN
Standard font	0.07%	0.09%	0.15%
Neat handwriting font	0.89%	0.62%	0.52%
Brush-writing font	13.94%	10.91%	10.56%

Table 4.1: *Minimum recognition error rate of different classification methods using moment normalization and LDA for dimensionality reduction.*

It is virtually impossible to achieve 100% correct recognition even on characters in standard print font. One of the reasons is the lack of difference between hiragana characters へ (*he*), べ (*be*) and ぺ (*pe*), and katakana ヘ (*he*), ベ (*be*) and ペ (*pe*).

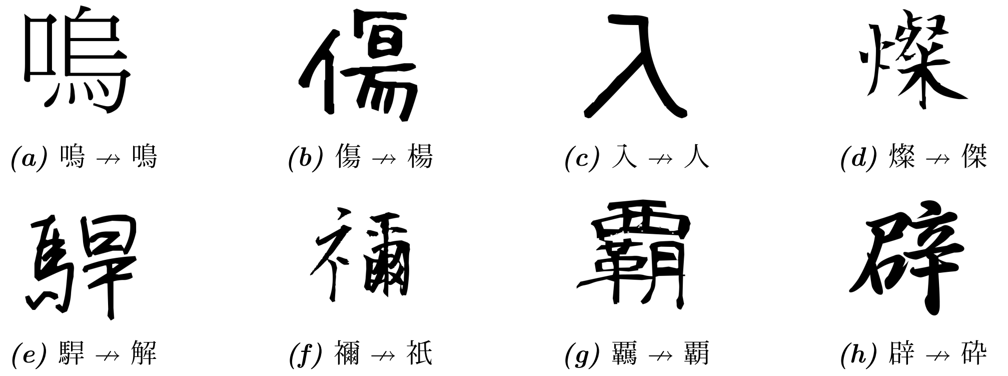


(a) 曙 (b) 惚 (c) 黙 (d) 飛 (e) 傷

(f) 香 (g) 猫 (h) 玄 (i) 鬱 (j) 孫

(k) 鎌 (l) 盧 (m) 烈 (n) 駿 (o) 秒

Figure 4.10: *Examples of characters correctly classified by my system.*



(a) 嗚 $\nrightarrow$ 鳴 (b) 傷 $\nrightarrow$ 暢 (c) 入 $\nrightarrow$ 人 (d) 燦 $\nrightarrow$ 傑

(e) 騂 $\nrightarrow$ 解 (f) 禰 $\nrightarrow$ 祇 (g) 覊 $\nrightarrow$ 覇 (h) 辟 $\nrightarrow$ 砕

Figure 4.11: *Examples of characters which were incorrectly classified by my engine. The correct character is shown on the left side and the incorrect classification is shown on the right side in the captions under the images.*

Table 4.2 shows the comparison of performance of my engine, NHocr, and Tesseract. I have used moment normalization, LDA for dimensionality reduction, and QDF classifier for this test. A set of 160 isolated characters in a standard print font, fonts resembling handwriting, and a font resembling writing with a brush were used for testing. None of the samples for testing have been used for training of my classifier so as not to give it an advantage. The precision of recognition is calculated by dividing the number of correctly classified characters by the number of all characters used in the test.

Both, my system and NHocr had no trouble recognizing characters in regular print font and neat handwriting. The difference in performance developed on fonts which deviate a lot from the standard form of character used in print fonts, which is not surprising because NHocr is not suited for recognition of handwritten characters according to the creator.

The lack of performance of Tesseract OCR was surprising. It had trouble recognizing characters even in regular print font. However, it is impossible to make any conclusions as to why the performance was so poor due to the absence of information about how the data for classification, which is available on the website of Tesseract, was trained.

	Tesseract	NHocr	My engine
Recognition precision	35.0%	86.3%	98.8%

Table 4.2: *Percentage of correctly recognized characters by the tested OCR systems.*

Chapter 5
Text Extraction

If one wants to perform optical character recognition in an application that is meant for real-life practical use, it is necessary to extract the text from an image taken by a camera and split the text into individual characters. First of all, a choice had to be made as what kind of approach to text extraction will be adopted. One of the options was extraction with assistance, where the user specifies what part of image they are interested in and want the application to recognize. Another one was extraction without any assistance, which means that an algorithm has to find out whether there is any text in the image and return a bitmap only with the text extracted from the whole image as its output. The former is easier to implement and also more suitable for the purpose of this work, because if the user is interested only in a part of text, there is no point in extracting and doing OCR on the whole text captured by the camera as it would produce extra information that the user is not interested in, which would impair the whole user experience. Text detection without assistance might be useful in some cases too, but the user can always specify the part of the image they are interested in if they are given means to do so. If they were not given a way to do it, they could never select only a part of captured text. Therefore, it was decided that the user of the application would have to draw a rectangle on the screen of a device and only the section of the image bounded by the rectangle would be processed and forwarded to the OCR engine of the application and the rest of the image would be discarded.

There are two major categories of localization of text in an image and extraction of it therefrom:

- **Localization and extraction of text from printed documents** - Locating and extracting text from a document is a task which has been studied for a long time, and now there are methods which achieve precision of over 99% [32]. One of the ways to extract text from a document is by binarization of the scanned image.

- **Localization and extraction of text from a scene in real world** - While localization of text in printed documents can be considered a solved task, localization of text in a scene from the real world is still an open problem which requires much more sophisticated techniques. There are two main categories of methods: methods using sliding windows, and methods using region grouping [33].

In the end, only a simple method for text extraction using binarization of image has been implemented in the application. The idea is that the text and the background will be binarized as inverse colors, either black text on white background or white text on black background. It does not try to locate what appears as text in the image like advanced methods do. The following assumptions are made for the extraction process.

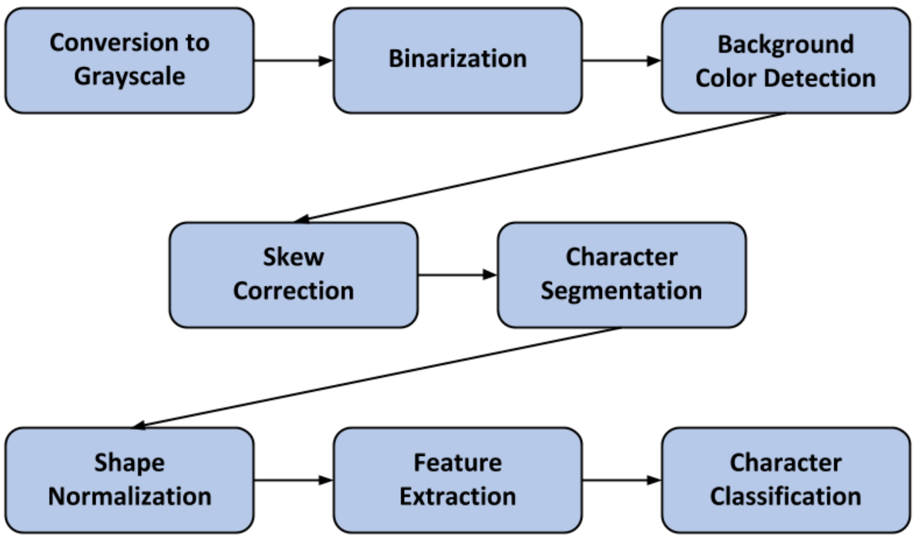


Figure 5.1: *Diagram of the steps from image acquisition to character classification.*

- The text is in contrast with the background. If the text is of the same color as the background, the extraction will fail. The ideal case is when there is a large difference between the luminance of the color of the text and the luminance of the background color.
- The background is not complex and does not consist of light and dark colors.

If an image does not meet the following criteria, it is unlikely that the text will be extracted successfully therefrom.

The following subsections describe all methods for text extraction and character segmentation which have been tested and used.

5.1 Binarization

First, the 32-bit image is reduced to 24-bit depth by dropping the alpha channel with transparency information, which is irrelevant here. Then, the 24-bit image is transformed into an 8-bit grayscale image by the luminance-preserving method [34] which is computed as a weighted sum of the intensity of the red, green, and blue channels of the original image:

$$Gray = 0.2126 \times Red + 0.7152 \times Green + 0.0722 \times Blue \tag{5.1}$$

The resulting grayscale intensity values of individual pixels range from 0 to 255. The grayscale conversion is over and binarization process follows.

All pixels whose intensity values on the gray scale are higher than a certain threshold are transformed into white pixels, and all pixels whose intensities are lower are assigned as black pixels. Otsu's thresholding method [35] has been used to find an optimum threshold value to separate the foreground from the background. The algorithm works with a grayscale image and it separates its gray-level histogram into two classes - black and white - so that the sum of their weighted between-class variances is maximal. The sum of between-class variances is defined as

$$\sigma_B^2 = \omega_0\omega_1(\mu_1 - \mu_0)^2, \tag{5.2}$$

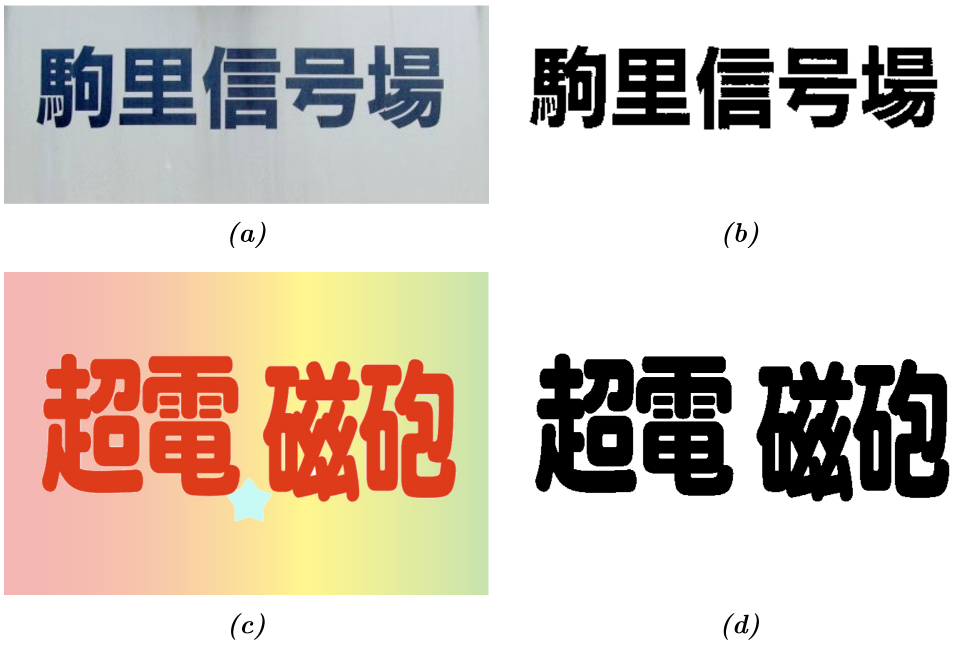


Figure 5.2: Example of binarization using Otsu's thresholding method. (a) and (c) are original images. (b) and (d) are the binary image results.

where ω_0 and ω_1 are the percentages of foreground and background pixels given by

$$\begin{aligned}\omega_0 &= \textstyle\sum_{i=1}^{k} p_i \\ \omega_1 &= \textstyle\sum_{i=k+1}^{L} p_i,\end{aligned} \tag{5.3}$$

where L is the total number of gray levels, k is a threshold between the two classes, and p_i is the percentage of pixels at gray level i, which means that

$$\sum_{i=1}^{L} p_i = 1, \quad p_i \geq 0. \tag{5.4}$$

Values μ_0 and μ_1 are the class mean levels given by

$$\begin{aligned}\mu_0 &= \textstyle\sum_{i=1}^{k} \frac{ip_i}{\omega_0} \\ \mu_1 &= \textstyle\sum_{i=k+1}^{L} \frac{ip_i}{\omega_1},\end{aligned} \tag{5.5}$$

The optimum threshold value that maximizes σ_B^2 is found by performing a sequential search over the histogram levels. Each gray level is selected as the threshold parameter and its corresponding σ_B^2 is calculated. The gray level for which σ_B^2 is maximal is chosen as the best candidate for the threshold. Should the area of maximum σ_B^2 stretch over two or more neighboring levels, its rounded average is selected as the best candidate.

5.2 Black on White / White on Black Detection

After performing the binarization, it is necessary to determine whether the text is black on white background or vice versa. I have designed the following algorithm for this purpose.

Assuming that the text is horizontal, it searches for a column with the largest number of white pixels and a column with the largest number of black pixels. If the largest number of white pixels found is larger than the largest number of black pixels, the background is considered to be white. Otherwise, the background is believed to be black.

```
Input: BinaryImage
whiteMax ⟵ 0;
blackMax ⟵ 0;
for $i$ ⟵ 1 to number of columns do
    colWhite ⟵ 0;
    colBlack ⟵ 0;
    for $j$ ⟵ 1 to number of rows do
        if BinaryImage[i][j] is black then
            colBlack ⟵ colBlack + 1;
        else
            colWhite ⟵ colWhite + 1;
        end
    end
    if $colWhite > whiteMax$ then
        whiteMax ⟵ colWhite;
    end
    if $colBlack > blackMax$ then
        blackMax ⟵ colBlack;
    end
end
if $whiteMax > blackMax$ then
    the background is white;
else
    the background is black;
end
```

Assuming that there is some space on the side by the text, it is not necessary to go through all columns of the picture, but only analyze the first column. If the first column consists of white pixels, the background is considered to be white. Otherwise, the background is black and the colors need to be swapped because the gradient directions in feature extraction would have opposite orientation.

5.3 Skew Correction

The deviation of text baseline from the horizontal axis - in case of horizontal text - is called a skew, and it is possible to correct it by rotation of the text. The algorithm for character segmentation as well as the character recognition process is likely to fail if the text is significantly skewed. Therefore, it is desirable to detect the skew angle of the text and rotate the text so that it is aligned with the horizontal axis in case of horizontal text. Radon transform method proposed by Aithal et al. [36] is applied to detect the skew angle.

The Radon transform function computes projections of an image matrix along specified directions [21]. The original version of the algorithm can be explained by supposing that there

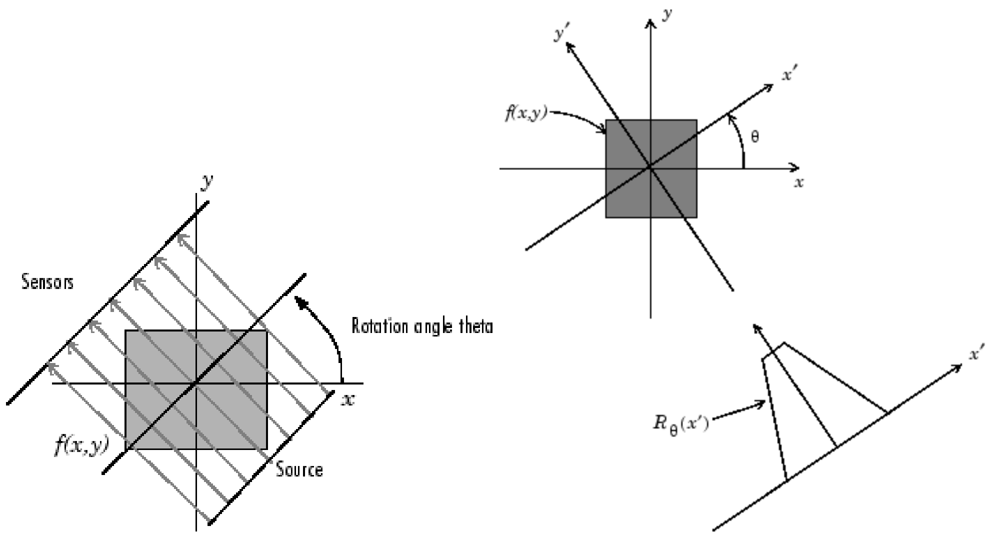


Figure 5.3: *Radon transform mechanism on the left and an example of one projection on the right. Figures taken from [37]*

is a source of parallel beams that are emitted towards the image matrix and their projection is registered by a sensor line behind the image matrix, which is perpendicular to the beams. This idea of figurative explanation is used on Matlab Documentation Center [37]. In my implementation, the source of beams is stationary and the image matrix rotates around its center; however, identical results can be achieved. The projection is computed under 180 different angles, with a 1° step between each angle.

```
Input: OriginalImage
Output: R
R ⟵ new int[180][hypot($W_{OriginalImage}$, $H_{OriginalImage}$)];
for $i$ ⟵ 1 to 180 do
    RotatedImage ⟵ rotate OriginalImage by i degrees;
    R[i] ⟵ sum up columns of RotatedImage;
end
```

Each projection in the Radon transform is stored in a matrix **R** of sizes $180 \times diag_{img}$, where $diag_{img}$ is the length of the diagonal of the image. The main idea of this method is that the highest intensity value in the projections will be in the column which corresponds to the skew angle. The reason for that is that pixels of text projected along its skew angle will be projected on the least number of bins. Let the intensity of a projection be the number of pixels collected in a projection on a single bin. The skew angle of the text is determined by the following process wherein the angle with the highest found intensity is found.

```
Input: R
maximum ⟵ 0;
skewAngle ⟵ 0;
for $i$ ⟵ 1 to 180 do
    for $j$ ⟵ 1 to $diag_{img}$ do
        if $R[i][j] > maximum$ then
            maximum ⟵ R[i][j];
            skewAngle ⟵ i;
        end
    end
end
skewAngle ⟵ -(skewAngle - 90);
```

The image used for the Radon transform should be reduced in size in order to make the computation faster. The time necessary for the computation grows quadratically with increasing size of the image.

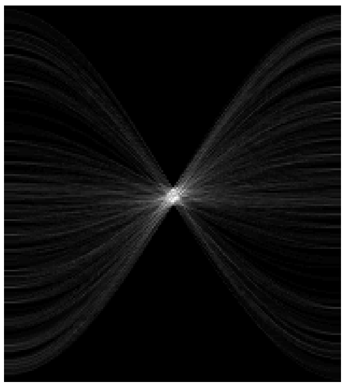


Figure 5.4: *An example of Radon transform on a correctly aligned line of text.*

The performance of Radon transform for skew correction is very good on a long line of text. However, when the text consists only of a few characters, this approach tends to fail and does not produce exact results. The reason for the drop in performance is that the vertical projection of the text does not differ very much from projections in different directions because the number of pixels in a line is not that big. As a result, a projection in a different direction might have a higher intensity than the projection in the direction of the skew angle.

A possible solution for the problem described above might be not to look for the highest intensity, but to look for the longest sequence of bins with zero intensity in a projection. The projection line has the same length for all examined angles, which means that when the baseline of text is perpendicular to the projection line, the area of bins which nothing is projected on is the largest as can be seen in 5.5. If there are at least two characters of aspect ratio close to 1:1 in the image, the rotation of the text is determined correctly in most cases. However, if there is only one character in the image, neither this method can determine the rotation correctly. This simple method is also likely to fail if something else than the text, for example residual noise, is present in the image, but since the whole character extraction process assumes that the background has been cleaned by binarization, this problem does not have to be considered here.

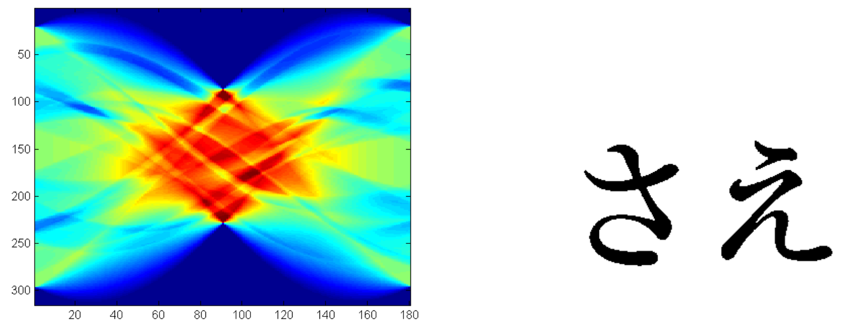


Figure 5.5: *Radon transform of the image on the right. Blue color denotes zero intensity, red color denotes high intensity. The location of the longest zero projection sequence in the Radon transform corresponds with the skew angle of the text and it can be found in the 91th column of the Radon transform matrix. The point of the highest intensity is found in the 107th column and the estimated skew angle differs from the correct one.*

5.4 Character Segmentation

Unlike the ISO basic Latin alphabet wherein all characters printed in a standard font, except lowercase 'i' and 'j', are single connected components, Chinese characters and characters from Japanese syllabaries may consist of several unconnected components, which makes extraction of individual characters from text a more complex task. If only characters printed in a standard font, in which individual characters do not overlap each other and heights of the characters are almost equal to their widths, were considered, an approach based on aspect ratio might be adopted. However, a method using only aspect ratios to extract individual characters will fail if the characters are handwritten or printed in a non-standard font, wherein the heights are likely to differ from the widths of the characters, and two or more characters might be connected at some point or overlap each other.

A modified version of algorithm presented by Chen et al. [38] has been used to perform the character segmentation in this work. It makes use of projection of the text on a line, which is analyzed to derive several features which are then used to segment the individual characters.

5.4.1 Projection Profile Analysis

Segmentation of horizontal text written from left to right will be described here. However, a very similar process can be used to extract individual characters from horizontal text written from right to left and from vertical text as well. The projection of horizontal text is acquired by the following process.

```
Input: BinaryImage
Output: projection
for $i \longleftarrow 1$ to number of columns in the image do
    columnSum ⟵ 0;
    for $j \longleftarrow 1$ to number of rows in the image do
        /* assumes black text on white background */
        columnSum ⟵ columnSum + 255 - BinaryImage[i][j];
    end
    projection[i] ⟵ columnSum;
end
```

Calculating Projection Block Features

1. Block width W_i - The width of a projection block.

2. Block height H_i - The height of a projection block.

3. Block gap G_{ij} - The distance between two neighboring projection blocks.

4. Average block width $avgW_B = \frac{1}{N}\sum_{i=1}^{N} W_i$ - The average width of a projection block.

5. Average character width $avgW_C = \frac{1}{M}\sum_{i=1}^{M} W_i, \quad \forall W_i \in W : W_i > avgW_B$ - The average width of a projection block which is wider than the average block width, therefore likely to be a single character and not just a part of it.

6. Average block height $avgH_B = \frac{1}{N}\sum_{i=1}^{N} H_i$ - The average height of a projection block.

N is the total number of projection of blocks and M is the number of projection blocks that are wider than $avgW_B$.

Each projection block is categorized into one of the following four groups of blocks based on the profile features described above. The classification of blocks into the four groups follows exactly the same rules as presented by Chen et al. [38]

1. Mark: The width of the projection block is smaller than $avgW_C \times T_{mark}$, where $T_{mark} = 0.3$.

2. Half-character: The width of the projection block is larger or equal to $avgW_C \times T_{mark}$ and smaller than $avgW_C \times T_{half}$, where $T_{half} = 0.7$.

3. Single character: The width of the projection block is larger or equal to $avgW_C \times T_{half}$ and smaller than $avgW_C \times T_{single}$, where $T_{single} = 1.5$. The block is likely to be a single character.

4. Multi-character: The width of the projection block is larger or equal to $avgW_C \times T_{single}$. The block is probably a compound of two or more characters.

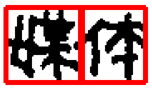


Figure 5.6: *An example of a multi-character block correctly split into two blocks with kanji* 媒 *and* 体.

Splitting Multi-character blocks

The blocks that are classified as multi-character blocks are likely to consist of two or more overlapping characters and need to be split. However, there is no easy way to correctly tell where to split the block into two. Common characteristics of kanji and Japanese syllabaries suggest that it is reasonable to assume that a suitable splitting point will be somewhere near the point that is $avgW_C$ distant from the side of the block. Therefore, a candidate area for a splitting point which ranges from the distance of $0.55 \times avgW_C$ from the side of the block to $1.45 \times avgW_C$ is defined. Subsequently, the following process is performed:

```
Divide the candidate area into 9 sections. Section 1 is the closest to the side of the
block and section 5 is where the point $avgW_C$ distant from the side is. ;
Find the minimum $min_i$ of the histogram projection in each of the 9 sections. ;
Find the maximum $max_H$ of the histogram projection in the whole candidate area.
;
if the minimum in sections 4, 5 or 6 is lower than $\frac{max_H}{4}$ then
    the splitting point is the minimum of $min_i$, where $i \in \{4, 5, 6\}$. ;
else
    the vector of minimums $min_i$ is multiplied by elements with vector
    | 3.0 | 2.5 | 2.0 | 1.5 | 1.0 | 1.5 | 2.0 | 2.5 | 3.0 | ;
    The minimum of modified $min_i$, where $i \in \{1, 2, \dots 9\}$ is selected as the
    splitting point. ;
end
```

The algorithm tries to split the block in the point near $avgW_C$ where the density of character stroke pixels is minimal, which is likely to occur on the side of the characters.

When all multi-character blocks are split, another two features are calculated. It is the estimate of the average gap between two characters ($avgG_B$) and the estimate of the average gap between two parts of one character ($avgG_W$).

1. $avgG_B = \frac{1}{N} \sum_{i=1}^{N} G_{ij}$, where G_{ij} is a gap between projection blocks B_i and B_j, and B_i is a multi-character, single character or a half-character, B_j is any of the four categories, and N is the total number of gaps meeting these conditions.

2. $avgG_W = \frac{1}{M} \sum_{i=1}^{M} G_{ij}$, where G_{ij} is smaller than $avgG_B$ and it is a gap between projection blocks B_i and B_j, and B_i is a half-character or a mark, B_j is , and M is the number of gaps meeting these conditions.

Merging Mark Blocks

The blocks classified as marks are very unlikely to be single characters and most of them need to be merged with a neighboring block. Let B_i, B_j, and B_k be three neighboring blocks and B_j be classified as a mark block. If the gap G_{ij} between blocks B_i and B_j is smaller than the gap G_{jk} between B_j and B_k, and G_{ij} is also smaller than the average within-character gap $avgG_W$, B_j is merged with B_i. If G_{jk} is smaller than G_{ij} and $avgG_W$, B_j is merged with B_k.

If a mark did not meet either of the above criteria and was not merged with a neighboring block, its height is compared with the average block height $avgH_B$. In case the height of the block mark is larger than a half of $avgH_B$, it is unlikely to be a punctuation mark, comma, or residual background noise, and it ought to be merged with a neighboring block if one of its neighbors is a mark block or a half-character block. Should both neighbors belong to one of the two categories, a mark block is given higher priority for merging. If the two neighbor blocks are both marks or both half-characters, the currently examined block is merged with the one that is closer to it. When all mark blocks have been processed, the average gaps $avgG_B$ and $avgG_W$ are recalculated.

```
/* $B_i$, $B_j$, and $B_k$ are neighboring blocks                    */
foreach block $B_j \in$ mark blocks do
    markConnected $\longleftarrow$ 0;
    if $G_{ij} < G_{jk}$ and $G_{ij} < avgG_W$ then
        merge $B_i$ and $B_j$;
        markConnected $\longleftarrow$ 1;
    else if $G_{jk} < G_{ij}$ and $G_{jk} < avgG_W$ then
        merge $B_j$ and $B_k$;
        markConnected $\longleftarrow$ 1;
    end
    if $markConnected = 0$ and $H_j > \frac{avgH_B}{2}$ then
        if $B_i$ is mark and $B_k$ is mark then
            if $G_{ij} < G_{jk}$ then
                merge $B_i$ and $B_j$;
            else
                merge $B_j$ and $B_k$;
            end
        else if $B_i$ is mark then
            merge $B_i$ and $B_j$;
        else if $B_k$ is mark then
            merge $B_j$ and $B_k$;
        else if $B_i$ is half-character and $B_k$ is half-character then
            if $G_{ij} < G_{jk}$ then
                merge $B_i$ and $B_j$;
            else
                merge $B_j$ and $B_k$;
            end
        else if $B_i$ is half-character then
            merge $B_i$ and $B_j$;
        else if $B_k$ is half-character then
            merge $B_j$ and $B_k$;
        end
    end
end
```

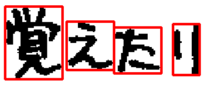


Figure 5.7: *An example of mark blocks correctly merged into hiragana character り.*

Merging Half-character Blocks

The last phase of the segmentation deals with half-character blocks. There is a high chance that a half-character block is a part of a character and should be merged with another block; however, it can also be a character on its own, whose width is simply smaller than the average. Therefore, it is impossible to easily tell whether a half-character should be merged with another block or not. The information about the gap sizes around the half-character block are used in the decision whether a half-character block should be merged or not. If a gap between a half-character block and one of its neighboring blocks is smaller than the average within-block gap $avgG_W$ or smaller than 2 times $avgG_W$ in case the neighbor is a half-character itself, the half-character block may be merged with its neighbor and the possibility is explored further.

The block resulting from the merge should not be significantly wider than the average width of a character, so a limit for the resulting width is set. Should the merge result exceed the limit, the merge is not performed and the half-character block is considered to be a single character. The limit was set as $1.2 \times avgW_C$. The following pseudocode describes the process.

```
/* $B_i$, $B_j$, and $B_k$ are neighboring blocks                    */
foreach block $B_j \in$ half-character blocks do
    if $G_{ij} < G_{jk}$ then
        if ($B_i$ is single character block and $G_{ij} < avgG_W$) or ($B_i$ is a
        half-character block and $G_{ij} < 2 \times avgG_W$) then
            if $W_i + G_{ij} + W_j \leq 1.2 \times avgW_C$ then
                merge $B_i$ and $B_j$;
            end
        end
    else
        if ($B_k$ is single character block and $G_{jk} < avgG_W$) or ($B_k$ is a
        half-character block and $G_{jk} < 2 \times avgG_W$) then
            if $W_j + G_{jk} + W_k \leq 1.2 \times avgW_C$ then
                merge $B_j$ and $B_k$;
            end
        end
    end
end
```

After performing the last phase of the segmentation, each block is treated as a single character which can be processed in order to extract its feature vector, which is subsequently used for classification of the character using one of the classifying methods introduced in the previous chapter.

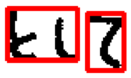


Figure 5.8: *An example extracted from a longer line of text where hiragana characters と and し, which were both classified as half-character blocks, were incorrectly merged into one block due to their small widths and proximity.*

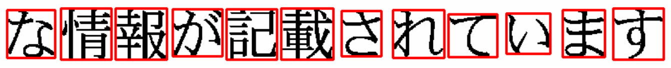


Figure 5.9: *An example of correctly segmented line of printed text.*

	Successful extraction
Handwritten text	277/315 (87.9%)
Printed text	260/262 (99.2%)

Table 5.1: *Table shows the percentage of individual characters successfully extracted from text.*

Table 5.1 shows the accuracy of segmentation algorithm on printed text and handwritten text. The accuracy is calculated as a ratio between correctly extracted characters and total number of characters. A segmentation mistake where two characters are merged into one, therefore, counts as two errors. The cases where the algorithm failed on printed characters were situations in which there were only a few characters in the picture, so the information about characteristics of the text was limited and was not sufficient to derive a correct decision. Therefore, it might be a good idea to abandon this method when only a few projection blocks are detected and try adopting an aspect ratio approach which would attempt to join neighboring blocks whose width/height aspect ratio is significantly lower than 1, and cut blocks whose aspect ratio is significantly higher than 1.

Chapter 6 Implementation

6.1 Training

The training phase and all testing of my OCR engine was performed in Matlab. The following toolboxes and applications that are not a part of an official Matlab distribution were used for training or testing of different approaches in Matlab:

- **Statistical Pattern Recognition Toolbox for Matlab** - A toolbox with implementations of various methods used in pattern recognition and machine learning created and maintained by Vojtěch Franc and Václav Hlaváč. `http://cmp.felk.cvut.cz/cmp/software/stprtool`
- **Feature Analysis** - Code by Iftekhar Tanveer, `http://www.mathworks.com/matlabcentral/fileexchange/26895-feature-analysis`. Its LDA implementation was used for dimensionality reduction step.

6.2 Comparison

The following tools were used for comparison of my OCR engine with other free OCR engines for Japanese language:

- **Tess4J** - Java wrapper for Tesseract OCR API. `http://tess4j.sourceforge.net/`
- **WeOCR Project website** - Web-based online OCR services with various OCR engines, including NHocr. `http://weocr.ocrgrid.org/`

6.3 Application

An alpha version of a mobile application for the Android OS has been created. The application is written entirely in Java language, and the minimum Android API requirement for the application is API 11[1], that is Android 3.0 Honeycomb. The application makes use of the built-in camera of a device to scan a text which the user wants the application to recognize and translate. The user can see the camera preview on the screen of the device and he or she has to draw a rectangle over the part of the image where the text he or she wants to analyze is located. After the user makes a selection, the specified section of the image is forwarded and processed further. The pure text is extracted by methods described in Chapter 5, but skew correction is

[1]Android API levels `http://developer.android.com/guide/topics/manifest/uses-sdk-element.html`

not used in the current implementation of the application[2]. The extracted text is then sent to the OCR engine. The optimal combination of preprocessing, feature extraction, and classifier as presented in Chapters 3 and 4 is used.

To acquire a sharp and clean image of a text, the text has to be in focus of the camera. By default, the camera of a device does not try to focus on any object on its own, but it is possible to change its behavior so that it handles focusing automatically. Continuous picture auto focus mode (FOCUS_MODE_CONTINUOUS_PICTURE)[3], which is an API level 14 parameter, is used on devices that support this feature to make the camera perform auto-focusing. Continuous video focus mode (FOCUS_MODE_CONTINUOUS_VIDEO) is used on devices supporting only API 13 and lower. Continuous picture focus mode is preferable as it is more dynamic, while the video focus mode results in smoother changes in focus according to the official Android documentation. It is possible to use the flashlight integrated in a mobile device to improve legibility of text under bad lighting conditions. The flashlight can be turned on and off by setting the parameters of the camera.

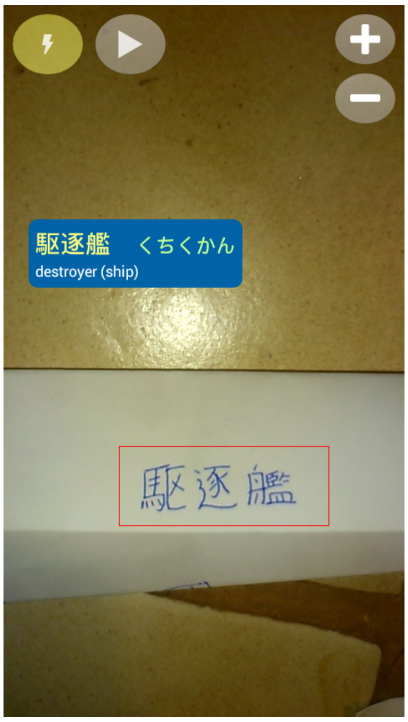


Figure 6.1: Screenshot of the Android application.

The original idea was that the application would periodically scan the view of the camera and provide the user with recognition results which will be automatically updated if the user moves with the device so that the selection rectangle drawn by the user on the screen specifies a different part of text, thus different characters. However, it has proven to be too much of a burden for the device to take high-resolution pictures one by one in a loop as fast as it can, which was roughly a period of one picture per half a second. The device would run out of memory despite attempts to recycle the pictures which were not necessary anymore by the garbage collector in Java. Therefore, the current implementation requires the user to draw a selection rectangle on the screen and press a button which makes the camera take one picture. Subsequently, the section of the picture defined by the selection rectangle is forwarded to the OCR engine of the application, and the result of the recognition is shown on the screen when the classification process is over. The picture taken by the camera is not stored anywhere and is recycled when the OCR engine does not need it anymore.

Mathematical operations are performed using the ElPsy Java library which I have made for this purpose. The library is small and does not feature many functions, but it features all functions that were needed for the application to run which are not in the standard Java Math library.

The JMDict (Japanese-Multilingual Dictionary)[4], a popular electronic Japanese dictionary file, is used to translate Japanese words into English. Since JMDict is distributed only as a single XML file, it has been exported into a SQLite database using the script available at `http://repo.or.cz/w/jblite.git`. The program tries to match sequences of characters recognized

[2]Skew correction is used in the Matlab demo application presented in Appendix E.

[3]Android Documentation - Camera.Parameters `http://developer.android.com/reference/android/hardware/Camera.Parameters.html`

[4]The JMDict Project - `http://www.edrdg.org/jmdict/j_jmdict.html`

by the OCR engine with dictionary entries in the database. The application assumes that the sequence is not very long, so it starts with the whole sequence of characters and reduces the length of the sequence one by one from the end of the sequence until a match in the database, a word or a phrase, is found. The subsequence of characters found in the dictionary is removed from the whole sequence and the same process is repeated on the rest of the sequence until the whole sequence is processed and translated. After that, all entries found in the database are displayed on the screen. Access into database is rather slow and needs to be optimized.

Sony Ericsson Xperia Ray with OS Android 4.0.4 Ice Cream Sandwich (API Level 15) has been used for testing. The device has a 1 GHz single-core CPU and dual-channel 333 MHz LPDDR2 memory. The resolution used by the camera is 2048 × 1536 pixels. The time between the moment when the user presses the button for recognition and when the picture from camera is ready for further processing is approximately 1,200 milliseconds and there is currently no way to speed up this phase as it is hardware-dependent. The time necessary to retrieve a selection of size 170 × 140 pixels from the whole picture is about 140 milliseconds. Extracting a feature vector from a picture of the aforementioned size takes roughly 750 milliseconds. Classification of a feature vector of 170 dimensions requires roughly another 350 milliseconds.

Chapter 7
Conclusion

7.1 Summary

Several methods for optical character recognition of Japanese writing system have been tested and compared and the optimal method has been implemented in a mobile application for devices with Android OS. The application is in alpha phase, has not been given a name yet, and needs further development. Several essential improvements should be done in the near future so that the application can be released to public.

The results show that a non-linear method for shape normalization results in better performance on characters which deviate from the standard form of the character. It is an expected result, because non-linear normalization methods focus on the parts of the character that need to transformed in order to get closer to a standard shape of the character used in print publications. However, there was no significant difference in performance between the two tested non-linear methods, the moment and bi-moment methods.

The created OCR engine was tested using a character set of 2,908 characters defined specifically for this application. The achieved recognition rate on isolated characters in standard fonts used for instance in books and newspapers was 99.91% when using moment method for shape normalization of the character, HOG for feature extraction, LDA for dimensionality reduction of feature vectors, and QDF for classification. The optimal feature vector size was approximately 150-180 dimensions. The recognition rate dropped to 99.38% on fonts imitating easily legible handwriting. The engine has outperformed NHocr, a free OCR engine for Japanese language which is used by a handful of applications, including those for mobile devices.

Text extraction from image is the weakest point of the system that requires attention the most in the near future. Extraction of the text is performed only by using the difference in contrast between the text and the background, which poses restrictions on the range of usability of the system. Thankfully, in many cases in which a user might be interested in the meaning of text, such as books, signs, and product labels, the text usually is in significant contrast with the background. Most often it is black text on white or whitish background.

Projection profile analysis is used for character segmentation and it has displayed good performance on longer lines of text. When the text for recognition is only one or a few characters long, there is not enough information for the algorithm to learn the properties of the text and the performance is not optimal. Implementing a different approach for such cases, e.g. an approach using aspect ratio of characters, is necessary.

The high accuracy of recognition of isolated characters is a good sign that the application might be used in a wide range of situations when other pieces of the system, text extraction in particular, are improved. It would be possible to use the created OCR engine in other applications that require Japanese OCR, too.

7.2 Future Work

Extraction of Text from Background

The performance of the application is largely dependent on whether the text is extracted correctly from the image or not. Currently, only extraction from a simple background is possible and extraction from more complex backgrounds will fail, thus the extraction process is what needs the most attention now. Improvement of text extraction would expand the scope of use of the application and it would be possible to recognize text even in situations where it is more difficult to separate the text from background.

Vertical Text and Horizontal Text with Text Flow from Right to Left

The method for segmentation of characters has been tested on horizontal text with flow of text from left to right and it is also the only supported text flow in the current implementation. Since Japanese is traditionally written vertically from top to bottom with columns ordered from right to left, and both the horizontal and vertical flow is used today - the vertical text flow being the prevalent text flow in paper publications and the horizontal text being prevalent in handwriting - it is necessary for the application to support both text flows. In addition, one may encounter cases in which the text is horizontal, but it flows from right to left. It is not a common text flow and it was mostly used in the past in cases where it was necessary to write in a horizontal line, for example on horizontal wooden signs. However, as it is a text flow which is not a foreign concept for the Japanese language, its support in future is desirable too.

Skew and Slant Correction

A method for skew correction to align the text baseline with the horizontal axis for horizontal text has been implemented and tested and it showed good results on longer lines of text. The performance on text which consists of one or a few characters, however, is not satisfactory and modifications need to be done if skew correction should be used in the application.

Text can also be slanted, which means that the direction of the left and right bounding line of every character in the text deviates from the direction perpendicular to the text baseline. If a text is significantly slanted, the segmentation and character recognition algorithms are likely to fail. For that reason, a slant correction would also increase the performance and operational scope of the application.

Speed Optimization

The code has been written with processing speed in mind; however, there is space for further improvement and the improvement is necessary. A possible solution to increase the speed of the OCR phase might be to write the code responsible for OCR in C++ to make the demanding computations run faster and wrap it in a Java shell. The dictionary database and accessing thereof has to be optimized and its speed improved. A new SQLite database should be created from the dictionary XML file, omitting the parts unnecessary for the application and organizing the database so as to minimize redundancy.

Support for Alphanumeric Characters

I would like to include support for recognition of characters from the basic Latin alphabet and Arabic numerals in the future, as there are phrases and acronyms using these literals alone or in combination with kanji, hiragana, and katakana characters in Japanese. While a user would be able to read a sequence of Latin letters and Arabic numerals, they might not understand their meaning in Japanese and the application would not be able to find the phrase in the dictionary as it could not recognize the characters.

References

[1] S. Mori, C. Suen, and K. Yamamoto, “Historical review of ocr research and development”, *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1029–1058, Jul. 1992. DOI: `10.1109/5.156468`.

[2] L. Eikvil, “Optical character recognition”, 1993. [Online]. Available: `http://www.nr.no/~eikvil/OCR.pdf`.

[3] S. Makino, Y. Abe Hatasa, and K. Hatasa, *Nakama 1: Japanese Communication, Culture, Context*. Houghton Mifflin College Div, Jun. 1998, ISBN: 978-0618135721.

[4] F. Yin, Q.-F. Wang, X.-Y. Zhang, and C.-L. Liu, “Icdar 2013 chinese handwriting recognition competition”, in *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, Aug. 2013, pp. 1464–1470. [Online]. Available: `http://www.nlpr.ia.ac.cn/events/CHRcompetition2013/competition/ICDAR%202013%20CHR%20competition.pdf`.

[5] C.-L. Liu, F. Yin, Q.-F. Wang, and D.-H. Wang, “Icdar 2011 chinese handwriting recognition competition”, in *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, Sep. 2011, pp. 1464–1469. [Online]. Available: `http://www.nlpr.ia.ac.cn/events/HRcompetition/ICDAR2011%20CHR%20Competition%20Final.pdf`.

[6] Y. Sobu and H. Goto, “Binary tree-based accuracy-keeping clustering using cda for very fast japanese character recognition”, in *Proceedings of the IAPR Conference on Machine Vision Applications (IAPR MVA 2011)*, 2011, pp. 299–302. [Online]. Available: `http://www.mva-org.jp/Proceedings/2011CD/papers/09-18.pdf`.

[7] K. Hori, K. Nemoto, and A. Itoh, “A study of feature extraction by information on outline of handwritten chinese characters : peripheral local outline vector and peripheral local moment”, *The transactions of the Institute of Electronics, Information and Communication Engineers*, vol. 82, no. 2, pp. 188–195, Feb. 1999, ISSN: 09151923. [Online]. Available: `http://ci.nii.ac.jp/naid/110003228675/en/`.

[8] A. Kent, H. Lancour, and J. E. Daily, *Encyclopedia of Library and Information Science: Volume 13 - Inventories of Books to Korea: Libraries in the Republic of*, ser. Encyclopedia of Library and Information Science. Taylor & Francis, 1975, pp. 269–273, ISBN: 9780824720131. [Online]. Available: `http://books.google.cz/books?id=uyPhAAAAMAAJ`.

[9] W. Hadamitzky and M. Spahn, *Japanese Kanji and Kana. A Guide to the Japanese Writing System.* Ser. Tuttle Language Library. Charles E. Tuttle, 1996, ISBN: 9780804820776. [Online]. Available: `http://books.google.cz/books?id=BCGylyOazSYC`.

[10] *List of jōyō kanji*, Official website of Japanese Agency for Cultural Affairs. [Online]. Available: `http://www.bunka.go.jp/kokugo_nihongo/joho/kijun/naikaku/pdf/joyokanjihyo_20101130.pdf`.

[11] M. Nakagawa and K. Matsumoto, "Collection of on-line handwritten japanese character pattern databases and their analyses", *IJDAR*, vol. 7, no. 1, pp. 69–81, 2004. [Online]. Available: http://www.tuat.jp/~nakagawa/pub/2004/pdf/nakagawa0404a-e.pdf.

[12] *List of 2500 most frequent kanji in newspapers*. [Online]. Available: http://www.atgc.org/kanji/kanji_2500_master_table_fr.html.

[13] J. W. Breen, *Electronic dictionary file kanjidic*, Monash University, 2000. [Online]. Available: http://www.csse.monash.edu.au/~jwb/kanjidic.html.

[14] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, *Contrast stretching*, 2003. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm.

[15] C.-L. Liu, H. Sako, and H. Fujisawa, "Handwritten chinese character recognition: alternatives to nonlinear normalization", in *Proceedings of the Seventh International Conference ond Document Analysis and Recognition*, IEEE, 2003, pp. 524–528.

[16] C.-L. Liu, M. Koga, H. Sako, and H. Fujisawa, "Aspect ratio adaptive normalization for handwritten character recognition", in *Advances in Multimodal Interfaces —ICMI 2000*, ser. Lecture Notes in Computer Science, vol. 1948, Springer, 2000, pp. 418–425.

[17] C.-L. Liu and K. Marukawa, "Pseudo two-dimensional shape normalization methods for handwritten chinese character recognition", *Pattern Recognition*, vol. 38, no. 12, pp. 2242–2255, Dec. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2005.04.019.

[18] M.-K. Hu, "Visual pattern recognition by moment invariants", *Information Theory, IRE Transactions on*, vol. 8, no. 2, pp. 179–187, Feb. 1962. DOI: 10.1109/TIT.1962.1057692.

[19] E. R. Davies, *Machine Vision: Theory, Algorithms, Practicalities.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004, pp. 49–51, ISBN: 0122060938.

[20] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, *Gaussian smoothing*, 2003. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm.

[21] B. Batchelor, *Machine Vision Handbook*. Springer, 2012, pp. 716–718, 764–765, ISBN: 978-1-84996-168-4.

[22] J.-x. Dong, A. Krzyzak, and C. Y. Suen, "An improved handwritten chinese character recognition system using support vector machines", *Pattern Recognition Letters*, vol. 26, no. 12, pp. 1849–1856, 2005, ISSN: 0167-8655. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2005.03.006.

[23] F. Kimura, T. Wakabayashi, S. Tsuruoka, and Y. Miyake, "Improvement of handwritten japanese character recognition using weighted direction code histogram", *Pattern Recognition*, vol. 30, no. 7, pp. 1329–1337, 1997. [Online]. Available: http://dx.doi.org/10.1016/S0031-3203(96)00153-7.

[24] M. Sagheer, C. L. He, N. Nobile, and C. Suen, "Holistic urdu handwritten word recognition using support vector machine", in *Pattern Recognition (ICPR), 2010, 20th International Conference on*, IEEE, Aug. 2010, pp. 1900–1903. [Online]. Available: http://dx.doi.org/10.1109/ICPR.2010.468.

[25] L. G. Roberts, "Machine perception of three-dimensional solids", *Optical and Electro-Optical Information Processing*, 1965.

[26] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed. Academic Press, 1990, pp. 76–77. [Online]. Available: http://books.google.cz/books?id=BIJZTGjTxBgC.

[27] M. Verleysen, “Learning high-dimensional data”, in *Limitations and Future Trends in Neural Computation 186*, IOS Press, 2003, pp. 141–162. [Online]. Available: `http://perso.uclouvain.be/michel.verleysen/papers/nato03mv.pdf`.

[28] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin: Springer, 2006, ISBN: 0387310738.

[29] J. Matas, University lecture on PCA and LDA. [Online]. Available: `http://cw.felk.cvut.cz/wiki/_media/courses/a4b33rpz/2010.11.29-pca.pdf`.

[30] F. Kimura, K. Takashina, S. Tsuruoka, and Y. Miyake, “Modified quadratic discriminant functions and the application to chinese character recognition”, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-9, no. 1, pp. 149–153, Jan. 1987. [Online]. Available: `10.1109/TPAMI.1987.4767881`.

[31] *Discriminant analysis*, MATLAB Documentation Center. [Online]. Available: `http://www.mathworks.com/help/stats/discriminant-analysis.html`.

[32] X. Lin, “Reliable ocr solution for digital content re-mastering. document recognition and retrieval”, in *IX, Proceedings of SPIE 4670*, 2001, pp. 223–231.

[33] L. Neumann and J. Matas, “Text localization in real-world images using efficiently pruned exhaustive search”, in *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, Sep. 2011, pp. 687–691. DOI: `10.1109/ICDAR.2011.144`.

[34] C. A. Poynton, “Rehabilitation of gamma”, *Proc. SPIE*, vol. 3299, pp. 232–249, 1998. [Online]. Available: `http://dx.doi.org/10.1117/12.320126`.

[35] N. Otsu, “A threshold selection method from gray-level histograms”, *Systems, Man, and Cybernetics, IEEE Transactions on*, vol. 9, no. 1, pp. 62–66, Jan. 1979. [Online]. Available: `http://www.tecgraf.puc-rio.br/~mgattass/cg/trbImg/Otsu.pdf`.

[36] P. K. Aithal, G Rajesh, D. U. Acharya, and P. Siddalingaswarny, “A fast and novel skew estimation approach using radon transform”, in *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 5, 2013, pp. 337–344. [Online]. Available: `http://eprints.manipal.edu/78905/1/A_Fast_and_Novel_Skew_Estimation_Approach_using_Radon_Transform.pdf`.

[37] *Radon transform*, MATLAB Documentation Center. [Online]. Available: `http://www.mathworks.com/help/images/radon-transform.html`.

[38] J.-L. Chen, C.-H. Wu, and H.-J. Lee, “Chinese handwritten character segmentation in form documents”, in *Document Analysis Systems: Theory and Practice*, S.-W. Lee and Y. Nakano, Eds., ser. Lecture Notes in Computer Science, vol. 1655, Springer, 1999, pp. 348–362. [Online]. Available: `10.1007/3-540-48172-9_28`.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	Jan Z d e n ě k
Study programme:	Open Informatics
Specialisation:	Computer and Information Science
Title of Bachelor Project:	Mobile Application for Recognition of Japanese Writing System

Guidelines:

The task is to create a mobile application for optical recognition of Japanese writing system, i.e. Chinese characters used in Japanese, and Japanese syllabaries.

The application is supposed to recognize the characters in a given sign or text and provide a translation into English.

Bibliography/Sources:

[1] Christopher M. Bishop: Pattern Recognition and Machine Learning. 2006. ISBN: 978-0-387-31073-2.

[2] Daniel Lopresti, Jianying Hu, Ramanujan Kashi (Eds.): Document Analysis Systems V. 5th International Workshop, DAS 2002 Princeton, NJ, USA, August 19–21, 2002 Proceedings. ISBN: 978-3-540-44068-0 (Print) 978-3-540-45869-2 (Online).

[3] Proceedings of the 7th International Conference on Document Analysis and Recognition. August 2003, Edinburgh, Scotland, UK. IEEE. ISBN: 0-7695-1960-1.

[4] International Conference on Pattern Recognition, Proceedings. 20th International Conference on Pattern Recognition, ICPR 2010, Istanbul, Turkey, 23-26 August 2010. ISBN: 978-1-4244-7542-1.

Bachelor Project Supervisor: prof. Ing. Pavel Zahradník, CSc.

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
Head of Department	**Dean**

Prague, January 10, 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jan Z d e n ě k

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Mobilní aplikace pro rozpoznávání japonského písma

Pokyny pro vypracování:

Úkolem je vytvořit aplikaci pro rozpoznávání japonského písma, tj. čínských znaků používaných v japonštině, a japonských slabičných abeced.

Aplikace by měla rozpoznat nápis a nabídnout pro něj anglický překlad.

Seznam odborné literatury:

[1] Christopher M. Bishop: Pattern Recognition and Machine Learning. 2006. ISBN: 978-0-387-31073-2.
[2] Daniel Lopresti, Jianying Hu, Ramanujan Kashi (Eds.): Document Analysis Systems V. 5th International Workshop, DAS 2002 Princeton, NJ, USA, August 19–21, 2002 Proceedings. ISBN: 978-3-540-44068-0 (Print) 978-3-540-45869-2 (Online).
[3] Proceedings of the 7th International Conference on Document Analysis and Recognition. August 2003, Edinburgh, Scotland, UK. IEEE. ISBN: 0-7695-1960-1.
[4] International Conference on Pattern Recognition, Proceedings. 20th International Conference on Pattern Recognition, ICPR 2010, Istanbul, Turkey, 23-26 August 2010. ISBN: 978-1-4244-7542-1.

Vedoucí bakalářské práce: prof. Ing. Pavel Zahradník, CSc.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Chapter A
Abbreviations and Symbols

A.1 Abbreviations

1D	One-dimensional.
2D	Two-dimensional.
API	Application Programming Interface.
ARAN	Aspect Ratio Adaptive Normalization.
HOG	Histogram of Oriented Gradients.
ICDAR	International Conference on Document Analysis and Recognition.
ISO	International Organization for Standardization.
JIS	Japanese Industrial Standard.
k-NN	k-Nearest Neighbors.
LDA	Linear Discriminant Analysis.
LDF	Linear Discriminant Function.
MAP	Maximum A Posteriori Probability.
MLE	Maximum Likelihood Estimation.
MQDF	Modified Quadratic Discriminant Function.
NN	Neural Network.
OCR	Optical Character Recognition.
PCA	Principal Component Analysis.
QDF	Quadratic Discriminant Function.
SVM	Support Vector Machine.

A.2 Symbols

$\mathbf{C}$	Class membership matrix.
c_x	X-axis component of centroid.
c_y	Y-axis component of centroid.
d_e	Euclidean distance.
d_m	Mahalanobis distance.
L	Lagrange polynomial.
$\mathcal{L}$	Likelihood function.
O	Asymptotic computational complexity.
$\mathrm{P}(\cdot)$	Prior probability.
$\mathrm{P}(\cdot \mid \cdot)$	Conditional probability.
R	Aspect ratio.
$\mathbf{R}$	Radon transform.
$\mathbf{T}$	Training data as feature vectors in matrix form.
$\mathbb{T}$	Training data.
$\mathbf{t}_i$	Feature vector of a sample from the training data.
$\mathbf{w}$	Eigenvector.
x	Coordinate on the x-axis.
x_c	X-axis component of geometric center.
y	Coordinate on the y-axis.
y_c	Y-axis component of geometric center.
$\mathcal{Y}$	Estimated classification.
θ	Gradient direction.
λ	Eigenvalue.
μ	Mean.
$\boldsymbol{\mu}$	Mean vector.
μ_{ij}	Central moment of image.
π	Number pi, the ratio of a circle's circumference to its diameter.
σ	Standard deviation.
σ^2	Variance.
$\mathbf{\Sigma}$	Empirical sample covariance matrix.
∇	Gradient strength.

Chapter B
CD Content

- **Android Application** - A project with the Android application built in Android Software Development Kit[1] which can be installed and run on a mobile device with Android OS. The application is called Japanese OCR for the time being and it is located in the AppTest folder. Mathematical operations of the application require the ElPsy Java library included in the Apptest
libs folder.

- **Datasets** - Image datasets obtained by processing PNG image files of individual characters. There are four datasets in the folder. A different method for normalization of character shape was used for each of them. BMN denotes bi-moment normalization, MN moment normalization, LN linear normalization, and NN means that no shape normalization method was used.

- **Image Samples**
  - **Images for testing** - This folder contains images with short text lines or individual characters which may be used for testing in the Matlab demo application.
  - **Samples from the training dataset** - This folder contains a selection of images which were used for training of classifiers. The image size is 500×500 pixels in most cases. The whole dataset used for training is not included on the CD as it is 7.32 GB large.

- **Font Processing** - Java Net Beans project with scripts for converting TrueType font files into SVG files, and subsequently into PNG image files. Uses Java Batik library. There are text files with lists of Unicode hex codes for various character sets, including the 2,908 character set used for the OCR engine, in this folder, too.

- **Matlab files** - Matlab scripts used for this work. images2mat.m lets you create a dataset from image files. get_feats_all.m extracts and stores feature vectors from images in a specified dataset. training.m trains a classifier from a selected dataset. The third-party tools mentioned in Chapter Implementation are necessary to run the files and can be downloaded by accessing the provided links.

- **Matlab OCR Demo** - All files necessary to run the Matlab demo are included in this folder. The demo can be launched by opening the file called demo_for_ocr.m.

- **PDF** - This thesis in PDF format, together with the XƎLaTeX file used to create the PDF and all figures and tables presented in the thesis.

[1]Android SDK - `http://developer.android.com/sdk/index.html`

- **Tess4J** - Java wrapper for Tesseract OCR used for testing.
  'Tess4J/src/net/sourceforge/tess4j/example/' contains the source file created for tests.

Chapter C
Application Manual

The interface and the controls of the application are very simple. The main screen of the application shows the camera preview. The region of interest for recognition can be specified by drawing a rectangle on the screen of the device with a finger. When you are satisfied with your selection, you can press the OCR start button located on the top of the screen. After that, the result of OCR together with the reading of the result in hiragana and the English translation are displayed on the screen in a blue box.

The settings menu contains only the About page and it can be accessed via the settings button of the device.

1 - Flash button. Pressing this button turns on/off the flashlight of the device.
2 - Start OCR button. OCR and translation is performed after pressing this button.
3 - Camera zoom-in button. Pressing this button increases the zoom of the camera.
4 - Camera zoom-out button. Pressing this button decreases the zoom of the camera.
5 - Recognized word in kanji.
6 - Translation of the recognized word into English.
7 - Reading of the recognized word in hiragana.
8 - A word selected for recognition by a red rectangle drawn on the screen.

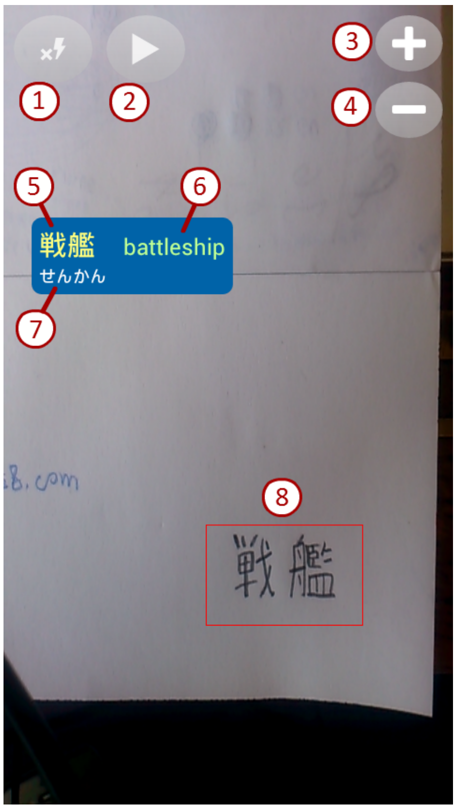


Figure C.1: *Screenshot of the Android application with description of elements.*

Chapter D
Dataset Format

There are four datasets included on the attached CD - one processed by linear shape normalization, one by moment normalization, one by bi-moment normalization, and one without shape normalization of characters. Each dataset is a simple Matlab structure named Character_Data where each element represents a picture of a character. Each element has three fields:

- char_image - 80 $\times$ 80 pixel pre-processed image of the character. The image is shape-normalized, smoothed, and shrunk to the needed size.
- label - Unicode hex code number of the character
- font - Font number denoting which of the used fonts this character image belongs to.

235,256 image files with pictures of individual characters were used to create datasets. The image files follow this naming convention: fXX_YYYY.png, where XX denotes the font number (1-83) and YYYY is the unicode hex code of the character. Font number 56 was deemed unsuitable and was not used in the end. Only a selection of the used image files is included on the CD because the entire set of images has 7.32 GB in size.

Chapter E
Matlab Demo for OCR

I have made a demo application in Matlab which performs OCR on image files. It uses the optimal engine presented in the thesis. The demo application can be launched by running the demo_for_ocr.m file in the Matlab OCR Demo folder on the attached CD. There are also image samples included on the CD which can be used to try the application out and see the performance of the OCR engine.

When the checkbox is not checked, the application only performs single character recognition regardless of the input. When it is checked, it can recognize a line of text and it also uses the proposed modification of the Radon transform method for skew correction.

- **1** - Filepath of the selected image.
- **2** - Opens file dialog.
- **3** - Choose between recognition of a single character and recognition of a line with multiple characters.
- **4** - Click to start recognition.
- **5** - Displayed result of recognition.
- **6** - Unicode hex codes of recognized characters.
- **7** - Displayed image which recognition was performed on.

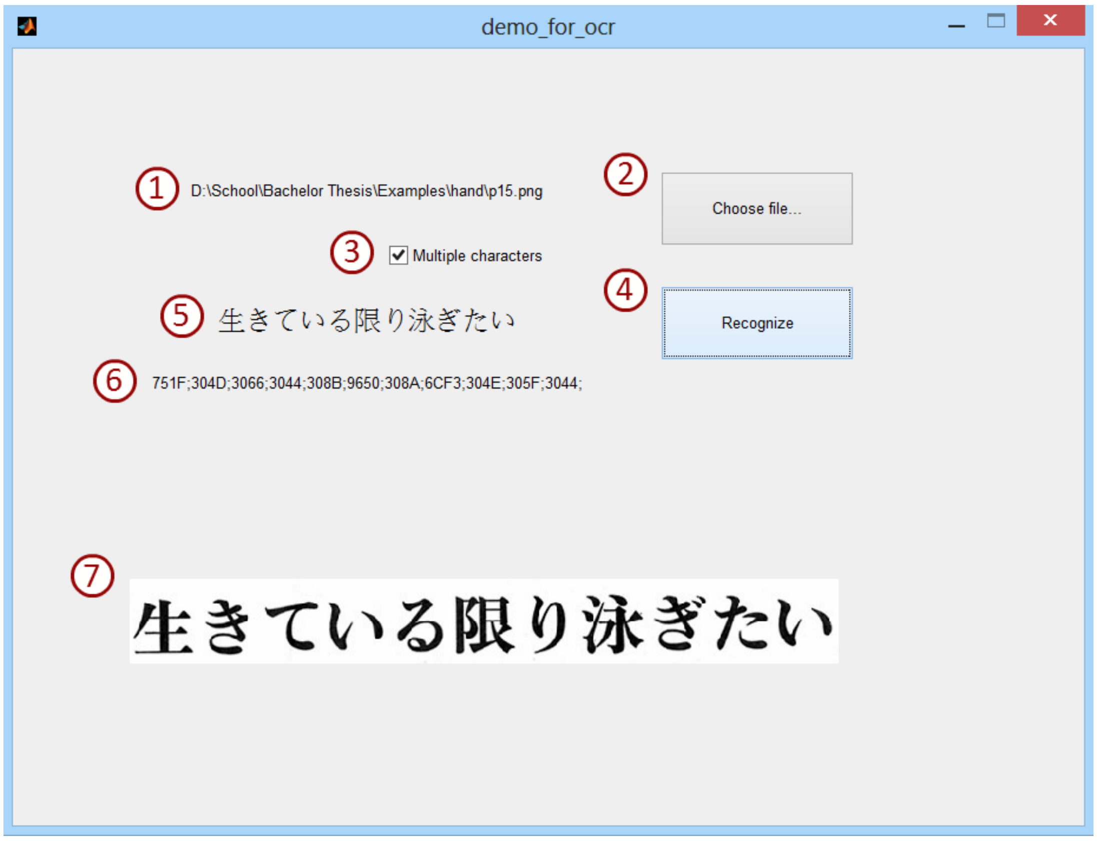


Figure E.1: *Screenshot of the demo application in Matlab.*