Heidelberg University
Faculty of Modern Languages
Department of Computational Linguistics

B.A. Thesis

Building and Improving an OCR Classifier for Republican Chinese Newspaper Text

Konstantin Henke
December 7, 2021

Im Neuenheimer Feld 681
69120 Heidelberg
henke@cl.uni-heidelberg.de

Supervisors: Prof. Dr. Anette Frank, Matthias Arnold (M.A.)
Assessors: Prof. Dr. Anette Frank, Prof. Dr. Barbara Mittler

Contents

List of Figures

List of Tables

Abstract

This work presents methods and results of an initial step towards full text extraction from a Republican Chinese newspaper. My basis is a small fraction of the image corpus for which text ground truth exists. I introduce a character segmentation method which produces over 90,000 labeled images of single characters. Then I pre-train a GoogLeNet classifier as an OCR model on character images extracted from font files and randomly augmented on the fly, whereafter I fine-tune it on the previously segmented character images. I show that the pre-training step is able to increase OCR accuracy from 95.49% to 96.95% on the test set and finally, how post-processing using a masked language model corrects up to 16% of remaining errors, increasing accuracy on the test set to 97.44%.

Zusammenfassung (German Abstract)

Diese Arbeit präsentiert Methoden und Ergebnisse eines ersten Schrittes zur Extraktion des Volltextes aus einer chinesischen Zeitung aus republikanischer Zeit. Als Basis fungiert ein kleiner Anteil des Bildkorpus, für den Textannotationen bestehen. Ich stelle eine Segmentierungsmethode vor, die reichlich 90,000 gelabelte Bilder einzelner Schriftzeichen produziert. Dann verwende ich ein GoogLeNet, um einen Klassifizierer auf künstlich aus Font-Dateien erzeugten und randomisiert augmentierten Schriftzeichenbildern vorzutrainieren, wonach Finetuning auf den echten Schriftzeichenbildern vorgenommen wird. Ich zeige, dass das Pre-Training die Korrektklassifikationsrate auf dem Testset von 95,49% auf 96,85% erhöht, und zuletzt, wie mittels eines maskierten Sprachmodells übrige Fehler korrigiert werden können, was einer weiteren Erhöhung der Korrektheit auf 97,44% und der Korrektur von reichlich 16% der OCR-Fehler entspricht.

1. Introduction

With the emergence of large-scale archives of digitized historical documents in the past decades, there has been a growing interest in efficient automatic information extraction. Among various kinds of documents, historical newspapers have been of particular interest as a representation of an era's zeitgeist—both from a content-driven and a linguistically motivated perspective.

Following two projects aimed at collecting Chinese tabloids (小報 *Xiǎobào*) and Chinese women's magazines, the Centre for Asian and Transcultural Studies (CATS) at Heidelberg University has been collecting a large amount of Republican Chinese newspapers, mainly consisting of image scans. As part of these collection projects, the platform "Early Chinese Periodicals Online" (ECPO)[1] was set up with the goal of providing free and location-independent access to more than 300,000 digital images of historical Chinese newspapers and additional metadata (Arnold and Hessel, 2019; Sung et al., 2014). It should be noted that digitized historical resources do not only help answer existing questions, but often also put up new ones: ECPO constantly poses new research opportunities, such as a new author and title index that will soon allow researches to obtain entirely new perspectives on interaction between historical agents that might not have been associated with each other in previous historical research.

Collections similar to ECPO include the NewsEye project[2] funded by various universities and national libraries from Germany, Austria, France and Finland[3]; "Chronicling America"[4], a fully searchable site put up by the United States' National Digital Newspaper Program; the impresso project[5] funded by the Swiss National Science Foundation; and VD16, VD17 and VD18, a German national bibliography collection which laid the basis for the OCR-D project[6].

One of the newspapers accessible on the ECPO project website is the 晶報 *Jīngbào*,

[1] https://uni-heidelberg.de/ecpo
[2] https://www.newseye.eu
[3] https://cordis.europa.eu/project/id/770299
[4] https://chroniclingamerica.loc.gov/
[5] https://impresso-project.ch/
[6] https://ocr-d.de/en/about, a full text transformation project coordinated by the Berlin-Brandenburg Academy of Sciences and Humanities, the Herzog-August Library Wolfenbüttel, the Berlin State Library and the Karlsruhe Institute of Technology.

"The Crystal"[7], with 9,385 scans of mostly double pages from March 3, 1919 to May 23, 1940. While the browser interface allows for improved navigation between issues and pages and provides essential metadata, this still only permits limited access to the actual textual information conserved in the image data. One entire month's issues have been annotated by native speakers using double-keying, a method that requires considerable financial and time expenditure. On the other hand, automated full text extraction is fast and more affordable by several orders of magnitude, yet non-trivial due to the newspaper's complex layout with text blocks strongly varying in position, shape, size and order of reading, surrounded by differently-sized headings, advertisements and marginalia. Apart from page segmentation, there is a need for a strong OCR model trained to deal with the recognition of characters printed in *Jīngbào's* particular font, featuring varying degrees of brightness and contrast due to inconsistent printing and scan quality as well as antiquated glyph variations and obsolete variant characters.

The following chapters and sections are structured as follows: Chapter 2 and 3 will give the reader the necessary knowledge to understand the basics of Chinese writing and printing as well as how Chinese character recognition has been addressed in related work. Chapter 4 will present the methods employed in this work and particularly go into great detail about the following contributions:

- character image segmentation and automatic labeling using existing ground truth,
- generation of additional synthetic OCR training data from Chinese fonts,
- and OCR post-processing (error correction using Bidirectional Encoder Representations from Transformers (BERT)).

Chapter 5 shows the concrete experiments done using these methods and discusses the obtained results. Finally, Chapter 6 presents a conclusion and an outlook on possible future work.

The source code for all implementation involved in this thesis can be found on GitLab at https://gitlab.com/konstantinhenke/bachelor-thesis.

[7] https://uni-heidelberg.de/ecpo/publications.php?magid=1

2. Chinese Writing, Fonts and Character Encoding

In order for the reader to be able to follow along during the next chapters, it is imperative to have a basic understanding of how written Chinese models the vernacular language and functions as a morphosyllabic system. Furthermore, this chapter will also touch upon Chinese character encoding in information technology, which is needed for understanding how the OCR classifier works.

2.1. Written Vernacular Chinese

When in non-scientific contexts one casually speaks about "Chinese" as a language, one usually refers to one of the two following concepts: 1. the entirety of the Sinitic languages spoken in Greater China as a branch of the Sino-Tibetan language family, including all its varieties and dialects, or 2. Mandarin Chinese, the most-spoken Sinitic language whose Beijing dialect was standardized and adopted as the national language of the People's Republic of China. The written language used throughout the later publishing years of the *Jīngbào* is one that closely mirrors the vernacular Mandarin dialects and thus mostly adheres to this national standard.[1] With the notable exception of Cantonese (mainly spoken in Hong Kong), most other Sinitic languages do not have a unitary standardized writing system[2], and while there is ongoing research on the representation of writing in other Chinese varieties in the *Jīngbào*, this work will solely focus on written vernacular Mandarin.

[1] Before the *May 4th Movement* in 1919 (and, to some extend, even later), most formal writing still used Classical Chinese, which is based on Old Chinese syntax. Starting from the Qin dynasty (221 BC), spoken Chinese evolved away from its written counterpart to such an extent that Classical Chinese is now largely unintelligible to modern Chinese speakers not educated in it. However, Classical Chinese reading (not writing) is an essential part of compulsory primary and secondary education in China, Hong Kong and Taiwan, and even nowadays a certain amount of fixed classical expressions are used throughout formal writing. Most notably, 成語 *chéngyǔ*, four-character classical idioms, are found in everyday use—both written and spoken.

[2] However, efforts are made for many Chinese languages such as Taiwanese Hokkien, a variety of Southern Min spoken in Taiwan (cf. also Lin (1999) and Lua and Iunn (2012) (in Chinese)). The Taiwanese Ministry of Education has published a standard character set for recommended use in Hokkien writing, titled 臺灣閩南語推薦用字 *(Taiwanese Southern Min Recommended Characters)*.

2.2. Chinese Characters

2.2.1. Characters as Morphemes

Chinese—with Mandarin in particular—features a writing system based on an extraordinarily extensive set of characters. Wilkinson (2018) calls it morphosyllabic, meaning

$$1 \text{ spoken syllable} = 1 \text{ character} = 1 \text{ morpheme},$$

where what we understand as a "word" is usually comprised of one or two (in some cases more) morphemes. The most frequently used pronouns, verbs and prepositions are usually monosyllabic, like 我 *wǒ* "I"/"me" or 去 *qù* "to go", while most nouns are disyllabic and—contrary to many multisyllabic words in European languages—often make sense as compound words joining the meaning of their component syllables:

- 眼鏡 *yǎnjìng* "glasses" = 眼 *yǎn* "eye" + 鏡 *jìng* "lens/glass";
- 手機 *shǒujī* "mobile phone" = 手 *shǒu* "hand" + 機 *jī* "machine";
- 冰箱 *bīngxiāng* "fridge" = 冰 *bīng* "ice" + 箱 *xiāng* "case/trunk/box".

This feature will be relevant for Section 3.4.

2.2.2. Characters as Components of Other Characters

To understand the challenge that comes with applying OCR systems to Chinese text, one also has to be aware of the low visual intra-class variance of the characters. This arises from the fact that character components reoccur in multiple—sometimes hundreds of—characters, especially in so-called phono-semantic compounds (ibid., ch. 2.3):

Looking at the character 箱 *xiāng* "box" again, it can be decomposed as ⺮ + 相. The former component is an abbreviated version of 竹 *zhú* "bamboo", which hints at the meaning of 箱 *xiāng* "box". The latter (相) is also pronounced *xiāng* and hints at the pronunciation of 箱 *xiāng* "box". This way of adding meaning components (like ⺮) to a character (like 相) to create a new character with different meaning but similar or equal pronunciation (like 箱) is the fundamental concept to how the vast majority of Chinese characters are composed. As a consequence, many characters share the same components, especially those within one phonetic series (a set of characters created from the same phonetic component), like 箱 *xiāng*, 廂 *xiāng*, 霜 *shuāng*, 想 *xiǎng*, 葙 *xiāng*, etc. Furthermore, this phenomenon can be of recursive nature, as in 礴 *bó*, which is composed of a semantic 石 and a phonetic 薄 *bó/báo*, which itself features a semantic ⺾ and a phonetic 溥 *pǔ/bó*, which again is 氵 + phonetic 尃 *fū*, which is phonetic 甫 *fǔ* + semantic 寸 (cf. e.g. Zhengzhang (2003)).

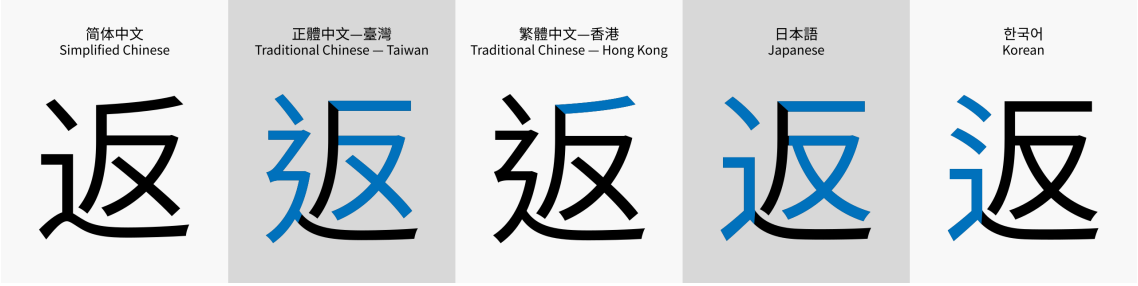


Figure 2.1.: Regional glyph variations (allographs) of the grapheme 返[4]

While it is difficult to exactly specify the percentage of phono-semantic compounds in all of Chinese characters, it is obvious that the vast majority of characters consist of reoccurring parts—no matter if in a phonetic, semantic or ideographic function (Wilkinson, 2018, Ch. 2.3). This results in a high visual confusability and proposes a big challenge for Chinese OCR systems, in contrast to systems processing Latin-alphabet-based documents.

2.2.3. Unicode and Fonts

With the rise of machine-processed text arose the need for suitable character encoding. In alphabetic writing, this involves nothing other than the minor step of assigning a number (usually called "code point") to every letter, even though occasionally, the visually same glyph is mapped to multiple code points across different alphabets, such as LATIN CAPITAL LETTER A (U+0041), CYRILLIC CAPITAL LETTER A (U+0410) and GREEK CAPITAL LETTER ALPHA (U+0391) which in most fonts should look exactly the same: A, A, A. That is, these are not the same *graphemes*—abstract functional units of writing—although the *glyphs*—their visible surface forms—look the same. Opposingly, "ɑ" would usually be perceived as the same grapheme as "a"[3], though these are two different glyphs. Different glyphs representing the same grapheme are also called allographs.

Unicode treats characters based on the principle of assigning code points to graphemes, not to glyphs; hence the creator of a font must decide upon the exact placement, angle, thickness etc. of every stroke when designing a glyph for a certain code point. This poses a problem when dealing with Chinese characters: Fig. 2.1 shows an example of variations in the glyph representations of the grapheme 返 across various regions. In the course of the so-called *Han unification* (Allen et al., 2012), all of the allographs in Fig. 2.1 have been assigned the same code point, leaving it up to the font designer to choose a standard to follow, and at the same time making it impossible for anyone to

[3]Usually known from handwritten text or italic printing (where a → *a*), since "ɑ" is not commonly seen in non-italic printing and has its own meaning when used as a character of the IPA.
[4]Source: https://w.wiki/3s9f

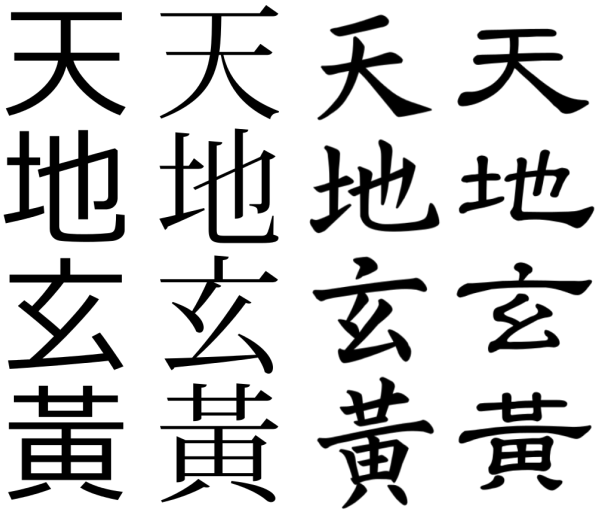


Figure 2.2.: The most common Chinese font styles (left to right):
黑體 *hēitǐ* (sans-serif / gothic);
明體 *míngtǐ* (Ming) a.k.a. 宋體 *sòngtǐ* (Song);
楷書 *kǎishū* (regular script);
隸書 *lìshū* (clerical script)

type a specific variant without the right font at hand. Some of the unified[5] variants are quite considerable in their structural dissimilarity, such as 直 (PRC/ROC standard) vs. 直 (Japanese standard) which are both encoded at U+76F4. Inconsistently, their descendants 值 and 値 were each given their own code point (U+503C and U+5024, respectively).[6] At the time of writing, Unicode provides code points for 93,779 characters (Table 2.1).

As for font styles, most of Chinese text elements are covered by three main styles (see Fig. 2.2): Sans-serif and Ming/Song can be seen as the direct equivalent of sans-serif and serif fonts known for Latin-based typography, where the former is more commonly seen on websites and (in variations) on posters etc. with shorter textual elements, while the latter is used for printing longer continuous texts as found in books and newspapers. Regular script is closer to handwritten character shapes, but still strictly standardized in terms of stroke length and position, hence it is largely used in language teaching material. It is also commonly seen as an equivalent to italic printing in Latin-based texts, as simply "obliquing" Chinese characters is a rather obvious typographic faux pas.

A fourth style, the clerical script, originates from Han-dynasty calligraphy, but since it is still highly legible to modern readers, it can still be found wherever some kind of artistic flavor or antique and classical appeal is strived for.

All of the printing in the *Jīngbào*'s text blocks is done in a Song font.

[5] = considered the same abstract grapheme and thus encoded at the same code point
[6] Hence, I have to used two different fonts to type out 直 vs. 直 but can use a single font for 值 and 値 as long as it provides glyphs for both code points.

Block range	Block name	number of characters
U+2E80⋯U+2EFF	CJK Radicals Supplement	115
U+2F00⋯U+2FDF	Kangxi Radicals	214
U+3400⋯U+4DBF	CJK Unified Ideographs Extension A	6,592
U+4DC0⋯U+4DFF	Yijing Hexagram Symbols	64
U+4E00⋯U+9FFF	CJK Unified Ideographs	20,989
U+20000⋯U+2A6DF	CJK Unified Ideographs Extension B	42,718
U+2A700⋯U+2B73F	CJK Unified Ideographs Extension C	4,149
U+2B740⋯U+2B81F	CJK Unified Ideographs Extension D	222
U+2B820⋯U+2CEAF	CJK Unified Ideographs Extension E	5,762
U+2CEB0⋯U+2EBEF	CJK Unified Ideographs Extension F	7,473
U+2F800⋯U+2FA1F	CJK Compatibility Ideographs Supplement	542
U+30000⋯U+3134F	CJK Unified Ideographs Extension G	4,939
		Σ 93,779

Table 2.1.: CJK (Chinese, Japanese, Korean) characters in Unicode

2.2.4. Traditional vs. Simplified Chinese

There are two standardized character sets for modern written Chinese: Traditional and simplified characters. The latter were promoted by the People's Republic of China's government for official use in printing, writing and education since the 1960s under the expectation of increasing literacy in the population. They are now in official use in the PRC, Malaysia and Singapore, whereas traditional Chinese remains in use in Taiwan, Hong Kong and Macau (Chu et al., 2012). Simplification was done with the ultimate goal of reducing written strokes, which was approached by various methods:

1. modifying existing cursive shapes for use in printing, e.g. 書 → 书, 樂 → 乐;
2. merging homophones, e.g. 復/複/覆/复 → 复 (all pronounced *fù*);
3. omitting entire components, e.g. 廣 → 广, 飛 → 飞, 習 → 习, 滅 → 灭;
4. replacing phonetic components by others that have less strokes: 鄰 *lín* → 邻 *lín* (with the phonetic components 粦 *lín* → 令 *lìng*);
5. re-adopting obsolete ancient variants or adopting variants already in use:
   從 → 从, 眾 → 众, 網 → 网, 與 → 与, 卻 → 却;
   etc.

Point 5 is particularly relevant to working with the *Jīngbào*: The parts of the image corpus I will use for the methods described in Chapter 4 are from April 1939 and hence

all printed before the government introduced the simplified character set, however some characters are already printed in a form later adopted during the simplification. For instance, the *Jīngbào* prints 幫, 强, 温 and 却 while the traditional characters used in the modern Taiwanese standard are 幫, 強, 溫 and 卻, respectively (cf. Table 4.5 on 30).

The absolute number of traditional characters that were simplified is hard to tell, as simplification is applied recursively: 馬 ("horse") is simplified to 马, which is then further conducted at component level in 媽, 碼, 騎, 駛, 驗[7] etc. There is, however, a considerable number of trivially simplifiable characters not in modern use like 驜[8], for which Unicode only provides one code point for the traditional form. This is to say, recursive simplification is not done for every traditional character encoded in Unicode, and as soon as one leaves the boundaries of Unicode, Chinese characters become very difficult to quantify. To still provide a reference point on the impact of character simplification: Around 29% of the characters in the ground truth section presented in Chapter 4 have a different form in the two character sets, not excluding multiple occurrences of the same character.

[7] i.e. in simplified Chinese, 妈, 码, 骑, 驶, 验 etc. are used
[8] According to zdic.net, a name of a horse. Presumably from some pre-modern work of literature.

3. Related Work

While at some point in the future, the ultimate goal for historical document processing will be to create a single pipeline that takes any document image as an input and is able to output all content information in a suitable format such as PAGE-XML (Pletschacher and Antonacopoulos, 2010), at the moment most research is still focused on producing efficient approaches for smaller substeps. This chapter is dedicated to giving exemplary insight into this.

3.1. Document Image Segmentation

Even though this thesis focuses solely on the character segmentation and recognition steps following below, one should not neglect the necessity for accurate page segmentation techniques. A survey presented by Eskenazi et al. (2017) gives a comprehensive overview on approaches to these challenges. Generally, one can differentiate between techniques that work top-down (starting from the entire page, segmenting it into its components) and those that work bottom-up (finding components that belong together to aggregate them). Their paper presents a detailed collection of both classical and deep learning algorithms, where the former often achieve surprisingly competitive results, although they rely on strict assumptions about various layout elements. Within the realm of the ECPO project, the historical nature of the document scans poses additional challenges. However, first bottom-up segmentation experiments using morphology-based methods as described by Liu et al. (2010) yield acceptable results.[1]

On the other hand, methods involving neural networks obviously require a sufficient amount of accordingly annotated data but allow for less strict layout assumptions, as shown in Chen et al. (2017), or the *dhSegment* tool (Oliveira et al., 2018), who treat page segmentation as a pixel labeling problem. Recently, more tools like *eynollah*[2] (part of the *Qurator* project (Rehm et al., 2020)) allow for straightforward training and even out-of-the-box usage. As for ECPO, first results using *dhSegment* look promising as well.[3]

[1] https://github.com/exc-asia-and-europe/ecpo-full-text/wiki/Finding-and-Connecting-Separators
[2] https://github.com/qurator-spk/eynollah
[3] https://github.com/exc-asia-and-europe/ecpo-segment

3.2. Chinese Character Detection and Segmentation

Once a document image is segmented into smaller structurally connected units, the next step is to detect the textual elements. One has to be aware of the fact that especially for Chinese characters, detection/segmentation and recognition are trivially separable steps, so it's reasonable to seek independent optimization. While Chinese characters can generally appear in all different kinds of fonts and styles (Yuan et al., 2018), in printed text, character shapes are never connected to each other and their squared appearance often yields an implicit grid layout. This allows for classical non-neural segmentation techniques using projection profiles (Fan et al., 1998; Lin et al., 2001). Mei et al. (2013) also build on projection profiles and leverage their use for single lines with connected component analysis, as do Xu et al. (2017) as well as Van Phan et al. (2011) for Vietnamese chữ Nôm[4] characters. In addition to character segmentation for grid-layout text blocks, projection profiles can also be used for skew detection and correction (Li et al., 2007).

It is out of question, though, that writing style, skew, noise or other issues concerning document image quality oftentimes produce layout situations that hamper these rule-based approaches, even to a degree that renders them completely unusable, e.g. for unconstrained handwriting or even calligraphy. Fortunately, thanks to deep learning in computer vision, object detection has overcome the problem of fixed hyperparameters. Generally speaking, U-Net (Ronneberger et al., 2015) has become popular for segmenting text areas and object detection models like the anchor-based YOLO (Redmon and Farhadi, 2018) to find the exact bounding boxes. Yang et al. (2018) employ what they call a recognition guided proposal network (RGPN) after segmenting single columns using projection profiles to propose regions of characters, and another CNN-based detection network to obtain their precise bounding boxes. Tang et al. (2020) propose HRCenterNet[5], an anchorless object detection network.

Finally, there do exist all-in-one solutions like *Tesseract*[6] (Smith, 2007) that apart from line finding also provide OCR functionality including language-model-based correction algorithms. (Ma et al., 2020) also present a framework that can jointly conduct layout detection, character detection and recognition.

3.3. Chinese Character Recognition

This section is about the actual OCR step. Depending on whether one deals with printed or handwritten Chinese characters, academic literature also often refers to the recog-

[4] a logographic writing system based on Chinese characters formerly used for the Vietnamese language
[5] https://github.com/Tverous/HRCenterNet
[6] http://code.google.com/p/tesseract-ocr

nition step as PCCR (printed Chinese character recognition) and HCCR (handwritten Chinese character recognition). While for obvious reasons the former is more relevant for dealing with the *Jīngbào* image data, the latter has received more attention throughout the past few decades (cf. related work in Melnyk et al., 2020), mainly due to the fact that individual handwriting styles pose a greater challenge to OCR techniques than the rather low variability in standard fonts used in printing. Zhong et al. (2015a) also bring up the fact that there exist more database resources for HCCR (e.g. Liu et al., 2011). Regardless of this, most methods presented in either field are generally applicable to both.

3.3.1. Early Work on PCCR

Probably the earliest approach to PCCR is the one of Casey and Nagy (1966), who employ simple template matching techniques. One of the authors later published a "Twenty-five Year Retrospective" on the field (Nagy, 1988) in which he describes the need for "special templates to detect radicals for subgrouping", referring to the component-assembled nature of Chinese characters. Considering reoccurring components to recognize Chinese characters intuitively does seem reasonable, however with the use of artificial neural networks (NN), it becomes more straightforward to simply regard characters as independent output classes – regardless of any shared components.[7] Xu and Ding (1992) were among the early pioneers to employ NN-based techniques for PCCR, followed by other approaches using traditional feedforward NNs (Khawaja et al., 2006a,b). Dai et al. (2007) give a great overview on the early stages of Chinese OCR before the use of CNNs).

3.3.2. Convolutional Neural Networks

Convolutional neural networks (CNNs) are inspired by the receptive fields found in animal vision and try to emulate the neurons in animals' visual cortices (Fukushima and Miyake, 1982). Consequently, they have been used with great success in many different subfields of computer vision, specifically image recognition and classification. CNNs date back to LeCun et al. (1989), who were the first to train the weights of a convolutional kernel (Fig. 3.1) using the backpropagation algorithm[8] known from traditional NNs. LeCun et al.'s use case also happened to be OCR, specifically digit recognition: Fig. 3.2 illustrates the digit image as an input, the convolutional layers at the bottom and the ten output units (one for every digit) at the top.

[7] There does exists more recent work concentrating on component structure on a sub-character level (He and Schomaker, 2018; Wang et al., 2017; Zhang et al., 2018).

[8] Independently discovered multiple times, but usually attributed to Rumelhart et al. (1986).

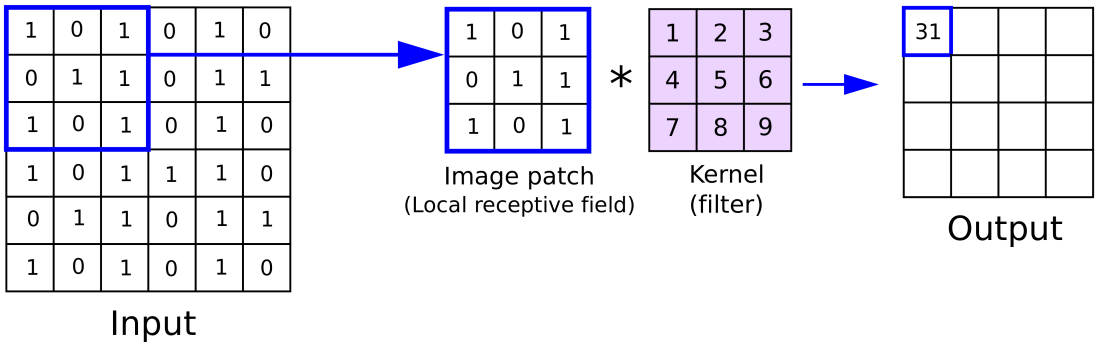


Figure 3.1.: Example for a convolutional filter (purple) with arbitrary numbers (1–9). A CNN optimizes the values itself through automated learning. Image as presented by Reynolds (2019).

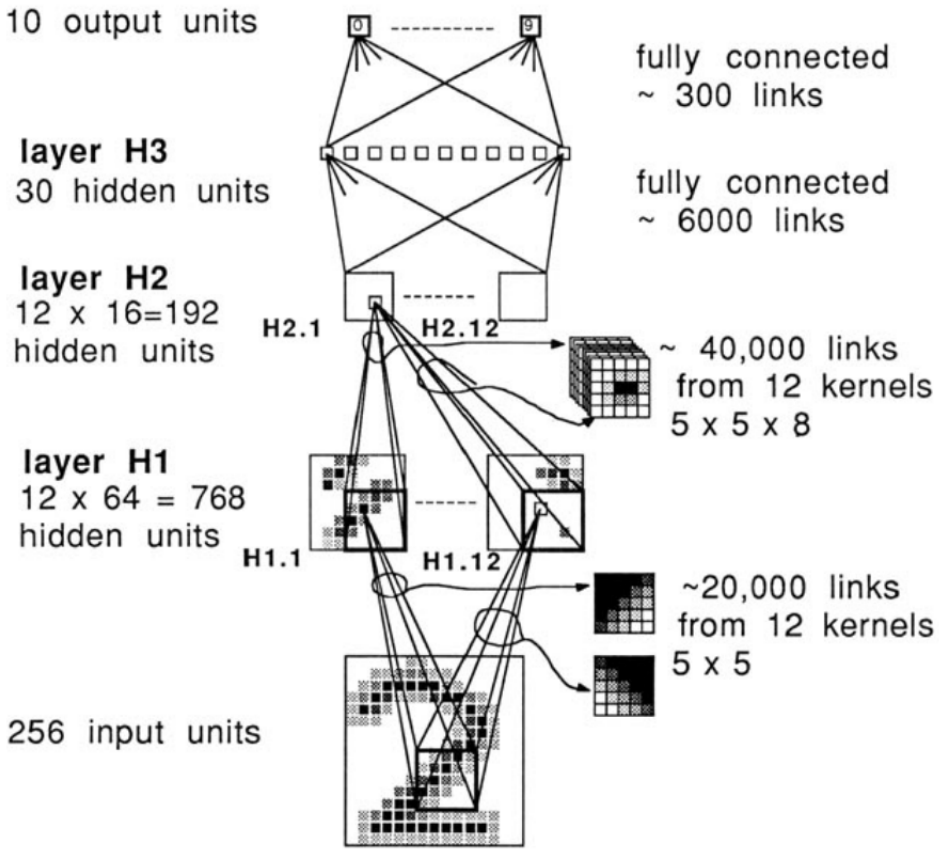


Figure 3.2.: The CNN architecture presented by LeCun et al. (1989)

Throughout the following decades, the basic principle of learning multiple kernels per layer and thus reducing dimensions until a fully connected layer predicts the output class largely stayed the same, though networks became deeper and added more attributes such as max pooling layers, ReLU activation and dropout, as seen e.g. in AlexNet (Krizhevsky et al., 2012). AlexNet won the 2012 *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) (Russakovsky et al., 2015), and in the following years, any other architectures continued to be outperformed by deeper and more sophisticated CNNs, such as ZFNet (Zeiler and Fergus, 2014), an improved version of AlexNet and winner of ILSVRC 2013, and GoogLeNet (Szegedy et al., 2014), winner of ILSVRC 2014.

Following their great success in image recognition, these CNNs have subsequently been employed in OCR, e.g. both Zhong et al. (2015b) and Xu et al. (2017) use a (slightly modified) GoogLeNet with great success. Motivated by these approaches and the demonstration of GoogLeNet's superiority in Chinese OCR as demonstrated in Yuan et al. (2018), this thesis will rely on a GoogLeNet as well (cf. Section 4.6.3). The ILSVRC did continue until 2017, and ever more powerful NNs have been presented in recent years for various vision tasks such as image classification and object detection. New state-of-the-art results on ImageNet are achived using transformer architectures known from NLP (Dosovitskiy et al., 2021; Zhai et al., 2021), which however is far beyond the scope of this thesis.

3.3.3. Artificial Image Generation and Image Augmentation

In machine learning, the training set is desired to independently represent the input and output space ideally in identical distribution to the test set. This results in the need to generate additional data if the training set would else not be of sufficient size or desired distribution, and the creators of competition-winning classifiers like AlexNet and GoogleNet all employed methods to artificially increase the number of training samples.

Chinese OCR generally faces a problem of data shortage due to the fact that a large number of input samples and their corresponding label in the output space for each of some thousand output classes are needed in order to model the target distribution. While suitable datasets for HCCR (Liu et al., 2011) exist, it is often cumbersome to obtain this amount of properly annotated printed character image data. This is where two techniques come into play:

1. producing entirely synthetic training images from scratch to increase the *size* of the training set;
2. augmenting existing training images to obtain a more *diverse* training set.

Augmentations often include geometric transformations, which themselves can be di-

vided into affine transformation (preserving lines and parallelism but not necessarily distances and angles, such as translations, rotations, shearing, etc.) and elastic transformations or non-linear distortions (not generally preserving lines and parallelism). One of the significant improvements in classifiers trained for handwritten digit recognition on the MNIST dataset has been obtained by the addition of elastic deformations (Simard et al., 2003) to already existing affine transformations (Wong et al., 2016).

In related work, the two above steps are usually applied jointly: de Campos et al. (2009) synthesize 62992 character images for the English alphabet from 254 fonts in 4 different styles and Jaderberg et al. (2014) produce images of entire English words using an elaborate generation procedure. With regard to Chinese, Ren et al. (2016) generate character images from 32 fonts and present their entire augmentation pipeline:

1. random selection of character and background color,
2. adding borders and/or shadows,
3. distorting the characters using projective transformations,
4. adding scene background patches to imitate real-word reflections,
5. adding Gaussian noise and Gaussian blur with random intensity.

Following these approaches, Xu et al. (2017) design a synthetic character engine that starts from clean images extracted from 28 fonts and then adds either random noise or erosion and blur. Zhong et al. (2015a) also employ non-linear transformations, demonstrating their effectiveness in increasing CNN performance and arguing that affine transformations are mostly unsuitable for Chinese characters.

3.4. OCR Post-Processing for Error Correction

Since predicting the right one among thousands of Chinese characters based on context-less pixel information is a notoriously difficult task, the output of any Chinese OCR model is likely to be subject to a considerable number of errors. Hence, there is a need for OCR post-processing. Languages with alphabetic writing systems allow for correction methods fundamentally different from what is needed for Chinese; for instance, Kissos and Dershowitz (2016) propose a method for misspelled words in Arabic OCR: On a word-level, they find correction candidates using Levenshtein distance, then train a classifier which decides whether or not to replace the OCR output with the highest ranked correction candidate with regard to OCR confidence, term frequency and dictionary features. Similar approaches are imaginable for English or other Indo-European languages with alphabetic writing systems.

Even though Chinese writing is fundamentally different, word-level approaches have not been entirely out of question for it, especially before powerful language models have come into existence: Tseng (2002) presents an algorithm that simply clusters confusing pairs (pairs of frequently seen two- or three-character sequences that only differ in one character) together and assumes the one with higher document frequency as correct.

Ultimately, the problem of correcting an OCR model's wrong character predictions can be generalized to Chinese spell checking (CSC), which over the last years has often been addressed by simply having a language model predict the right character from a previously obtained candidate set. This is reasonable due to the fact that *visually* similar characters often only differ in one of their components (cf. also Section 2.2.2), which means that characters in the candidate list are particularly unlikely to be *semantically* similar and thus—intuitively—easy to be corrected by a language model that takes context into account. Generally however, it is not trivial to find a good heuristic for

a) deciding which characters are likely to need correction at all and
b) deciding which candidates to have the language model choose from.

For example, Zhuang et al. (2004) use an n-gram model and an LSA[9] language model, then address problem a) by taking all characters into consideration and problem b) by generating a candidate list of visually similar characters as follows:

1. Record $n(F, C)$ in the training corpus. $n(F, C)$ is the times that F is the first choice in the candidate list produced by the OCR engine while the correct ("gold") character is C.
2. Calculate the confusion probability $P(C|F) = n(F, C)/n(F)$ where $n(F)$ is the times that F is the first choice in the candidate list.
3. Sort all possible C according to the probability.
4. Save the first M possible characters as the similar characters of F, and save their confusion probabilities.

Apart from that, various other approaches have been proposed: Zhuang and Zhu (2005) use the OCR model's candidate distance instead of a fixed k to both decide which characters to correct (a) and to reduce the search space (b). Then they combine various n-gram models to find the most likely candidate. Similarly, Wang and Liu (2019) manually set an OCR confidence threshold of 95% (a) and have the language model choose the most likely candidate among the top 5 OCR candidates (b).

CSC is however not only needed for OCR post-processing (with a search space of *visually* similar characters), but also for post-processing of e.g. Chinese text typed on computers with a search space of *phonetically* similar characters (as Chinese character input

[9] Short for "latent semantic analysis", for further information cf. Zhuang et al.'s work mentioned above.

methods are usually based on character pronunciation) and eventually most methods are applicable to both. Hong et al. (2019) generate the search space (b) from databases of both visually and phonetically similar characters and use a masked language model similar to BERT (Devlin et al., 2019) for correction. Yang et al. (2019) demonstrate a technique for character correction in speech recognition: They train a Bi-LSTM model operating on single characters to identify erroneous characters (a) and then replace them with the most likely candidate from a list of homophones (b). The top candidate is predicted by another Bi-LSTM model.

While the fact that most Chinese words are disyllabic (cf. Section 2.2.1) allows for efficient use of n-gram models as shown above, there are plenty of situations where switching single characters (especially those that aren't part of a disyllabic word) will still result in a meaningful and grammatical sentence, e.g.

- 他不喜歡你 "he doesn't like you",
- 他也喜歡你 "he also likes you",
- 他們喜歡你 "they like you",

where only the second character is changed (不 "not", 也 "also", 們 (plural marker)). Relying on masked language models such as BERT trained on single characters as tokens might therefore yield superior performance, as more context is taken into account. Hence, in this work I will employ a BERT language model for OCR post-processing.

4. Implementation

This chapter will give insight into the concrete methods I implemented in this work. As the focus of this thesis lies in segmenting single characters to train and post-process an OCR classifier, all the algorithms below are proposed under the premise that automatic text block segmentation has already been completed. All of the image processing methods described in the following sections of this chapter are implemented using the OpenCV library[1] for Python.

4.1. Text Block Segmentation

In order to emulate the segmentation results, several hundred text blocks as seen in Fig. 4.1 are manually cropped from the 40 issues of April 1939 which text ground truth exists for.[2] The corresponding section of the ground truth is manually extracted and assigned the same index as the text block image. Several crops are sorted out due to inconsistent layout (see next two sections) or bad scan quality. Table 4.1 gives an overview over the remaining data after a rough 50–25–25 split for training, validation and test.

Figure 4.1.: Examples of text blocks manually cropped from the *Jīngbào*

[1] cf. https://pypi.org/project/opencv-python/ and https://opencv.org/
[2] https://github.com/exc-asia-and-europe/ecpo/tree/master/jingbao/1939/04

	total	training	validation	test
number of crops	840	426	183	231
total number of characters	92,039	47,986	21,676	22,377
number of unique characters	3,045	2,797	2,074	2,187

Table 4.1.: Statistics on the ground truth sections assigned to the text block crops

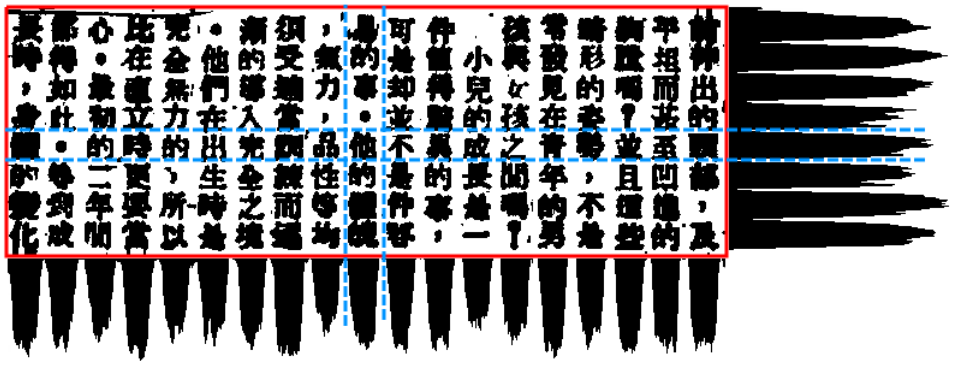


Figure 4.2.: Projection profiles after binarization using a 125px kernel

4.2. Deskewing

Due to Chinese characters being printed into near-squares, it is common to find that resulting text blocks feature an implicit grid usable for segmentation (as can be seen in Fig. 4.1). Deviation from this grid is oftentimes subject to the need for more characters to be squeezed into one column, the desire to not have new columns start with certain punctuation marks, the need to fill available space, or simply inaccurate printing. In order for the method described below to work, any text blocks that don't adhere to the grid layout for any of the above reasons are manually sorted out or not cropped in the first place.

To perform deskewing, I employ adaptive binarization (Fig. 4.2; cf. also Section 4.4) and then calculate horizontal and vertical projection profiles similar to Fan et al. (1998). Roughly following Li et al. (2007), I subsequently aim to find a rotation angle α with $\alpha \in [-2.0°, -1.5°, \ldots, 2.0°]$ such that image rotation by α maximizes the criterion S:

$$S = \sum_{i}^{w-1} (c_{i+1} - c_i)^2 + \sum_{j}^{h-1} (l_{i+1} - l_i)^2 \tag{4.1}$$

where w and h are the width and height of the image, c_i is the number of black pixels in the i-th column (= the corresponding value of the vertical projection profile) and, analogously, l_j the number of black pixels in the j-th line. S grows for bigger differences between adjacent values and will thus be maximized for an α that yields an extremal projection profile, which is the case if the text block is fully derotated.

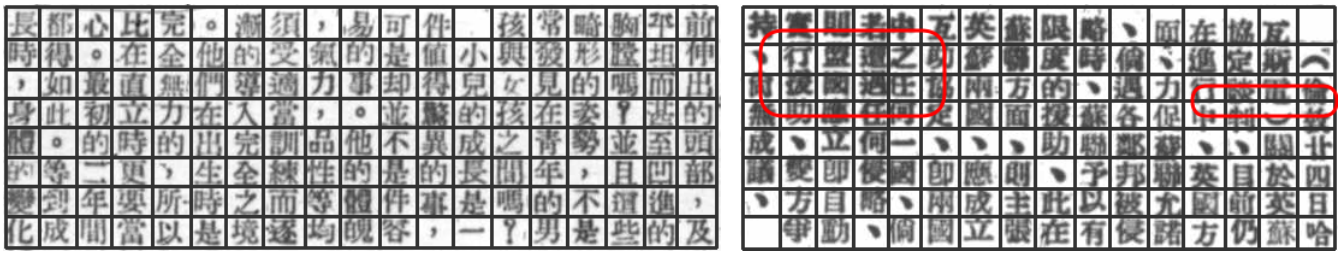


Figure 4.3.: Separators generated using only global projection profiles

Figure 4.4.: Separators generated using only local projection profiles for columns

Figure 4.5.: Separators generated using hybrid approach combining global and local projection profiles

4.3. Character Segmentation

After deskewing, I cut the original (= gray-scale, non-binarized) text block image into single character images along separators defined by the following heuristic:

1. Use the valleys of the vertical projection profile to define separators between the columns.
2. Use the valleys of the horizontal (global) projection profile to define separators between the lines.
3. For every column, produce another (local) projection profile.
4. If a local separator defined by 3. lies within 7px distance of a global separator defined by 2., discard the global separator and only use the local separator; else only use the global separator.

The positions of the valleys are obtained by `scipy.signal.find_peaks`[3] (using a min-

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html

imum distance of 22 (1.), 20 (2.) and 14 (3.)). The motivation for treating column segmentation separately is the wider margins between them, making segmentation using projection profiles easier, as can also be seen in Yang et al. (2018). Figs. 4.3–4.5 demonstrate the failures of using only global (Fig. 4.3) or only local projection profiles (Fig. 4.4) to segment single columns. Problematic cases are marked in red, local improvements in green. All problematic cases are solved by the hybrid approach (step 4. above, Fig. 4.5).

Subsequently, the resulting fields can be easily mapped to the ground truth text (1 field = 1 character). Any indentation has to be manually marked in the annotation in order for the mapped characters to shift to the right position within the column. Further, it must be noted that the method entirely relies on correct ground truth annotation. Missing lines are easily detected, as this will cause the number of detected columns in the image to surpass the number of lines in the corresponding annotation. Missing or extra characters within a line are not automatically detected and would cause the character assignment to shift by one slot (disastrous!), so annotations have to be double-checked, e.g. by confirming that every line corresponding to a single crop is of the same length. Wrong or swapped characters that don't affect the line length (e.g. phonetic typos such as 莫 *mò* instead of 墨 *mò*) are hard to be detected without careful proof-reading of the entire ground truth. Hence they will, if existent, invariably lead to lower accuracy.

4.4. Binarization vs. Contrast Enhancing

For noisy input data it is challenging to identify foreground pixels (usually any pixels that belong to the to-be-OCR'ed characters) and background pixels (any other pixels, e.g. the document background or any noise). Image binarization is the method used for this identification, meaning that ideally, all foreground pixels are assigned the value 0 (black) and all background pixels the value 255 (white). A significant amount of research has sought to improve binarization directly, as for high-resolution images and less complex character sets such as the Latin alphabet binarized input largely improves OCR results (Gupta et al., 2007). The *International Conference on Document Analysis and Recognition* (ICDAR) and the *International Conference on Frontiers in Handwriting Recognition* (ICFHR) even hold an annual document image binarization competition (DIBCO)[4], presenting increasingly elaborate methods. The next few paragraphs will however present more straightforward approaches and critically evaluate their usefulness within the realm of this work.

The most intuitive approach to binarization, global thresholding, simply does the following: Given a grey-scale image, i.e. one whose pixel values are of one channel (single numbers), usually between 0 and 255, find a threshold t such that setting every pixel

[4]For the most recent DIBCO, cf. Pratikakis et al. (2019) and https://vc.ee.duth.gr/dibco2019/.

to 0 or 255 depending on whether its value is greater or smaller than t means for all foreground pixels to turn black and all background pixels turn white. Formally, for an image A with

$$A = (a_{ij})_{\substack{i=1,\dots,I;\\ j=1,\dots,J}} = \begin{bmatrix} a_{11} & \cdots & a_{1J} \\ \vdots & \ddots & \\ a_{I1} & & a_{IJ} \end{bmatrix}, \tag{4.2}$$

we obtain the values b_{ij} of the binary image B through

$$b_{ij} = \begin{cases} 0 & \text{if } a_{ij} < t \\ 255 & \text{otherwise} \end{cases}. \tag{4.3}$$

Unfortunately, contrast and brightness may vary considerably within the same document, sometimes to a degree that makes it impossible to find a single value t that reliably separates foreground and background in the entire image, even with advanced threshold selection techniques like the well-known Otsu's method (Otsu, 1979) which is included in most modern image processing libraries.

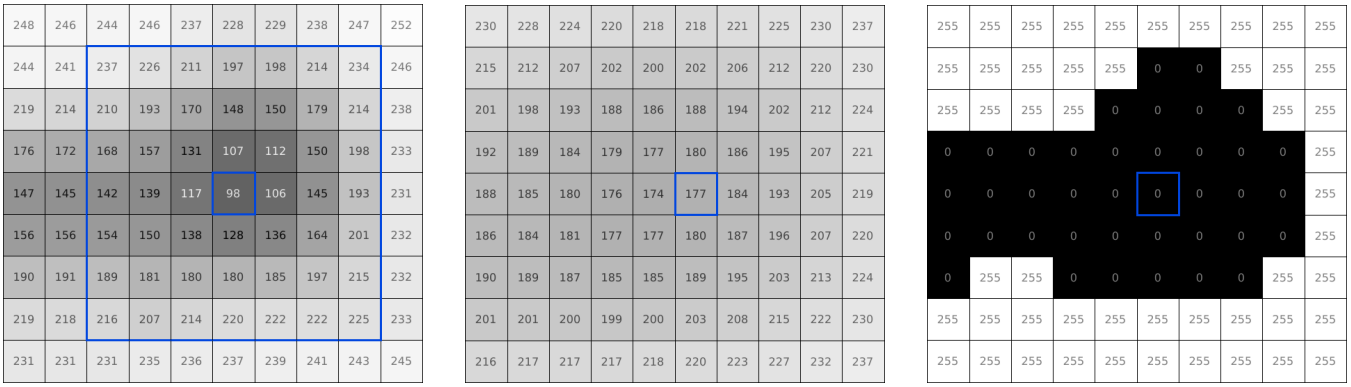
Hence, other than global thresholding, more advanced techniques have arisen to face this challenge. One of these is adaptive mean thresholding: Instead of using one global t, the threshold is set separately for every pixel as the average of the pixel values in its proximity (cf. Fig. 4.6). We thus obtain a threshold matrix $\bar{T} = (\bar{t}^k_{ij})$ whose entries are the moving averages within a squared kernel of size k around every pixel a_{ij} in an image A as defined in Equation 4.2. Formally, these threshold values are computed by

$$\bar{t}^k_{ij} = \frac{\sum_{p=p_{min}}^{p_{max}} \sum_{q=q_{min}}^{q_{max}} a_{pq}}{(p_{max} - p_{min}) * (q_{max} - q_{min})} \quad \text{with} \quad \begin{array}{c} k'=\frac{k-1}{2}; \\ p_{min}=max(1,i-k'); \\ q_{min}=max(1,j-k'); \\ p_{max}=min(i+k',I); \\ q_{max}=min(j+k',J) \end{array} \tag{4.4}$$

The kernel (the outer blue box in Fig. 4.6 (a)) is cut off at the edges of A if necessary, hence the need to specify $p_{min}, q_{min}, p_{max}, q_{max}$. This means for the values b'_{ij} of the adaptively thresholded binary image B':

$$b'_{ij} = \begin{cases} 0 & \text{if } a_{ij} < \bar{t}^k_{ij} \\ 255 & \text{otherwise} \end{cases}. \tag{4.5}$$

Fig. 4.6 shows an example computation: Assuming at the position $i = 17, j = 42$ of an image A we find the pixel $a_{17,42} = 98$ and set $k = 7$ (a): The average of the surrounding kernel $\bar{t}^7_{17,42} \approx 177$ (b) and finally the thresholding step $a_{17,42} = 98 < 117 \approx \bar{t}^7_{17,42} \implies b_{17,42} = 0$ (c).



(a) A: original image and 7px kernel (b) $\bar{T}$: averages for $k = 7$ (c) B': binary image

Figure 4.6.: Adaptive thresholding: 98 is lower than 177, the average of its surrounding 7px kernel, and is thus set to 0.

As with choosing a suitable t in global thresholding, the difficulty for adaptive mean thresholding lies in setting k. Larger values for k will make sure the background stays white, but due to higher values all over $\bar{T}$, the result B' is darker and might even cause foreground elements to "clump together" (Fig. 4.7 (a)). As k gets lower, individual foreground elements appear cleaner and more detailed, but wherever the kernel fits between them covering only background areas, some background pixels will be set to 0. This effect grows stronger the lower k is chosen (Fig. 4.7 (b–d)).

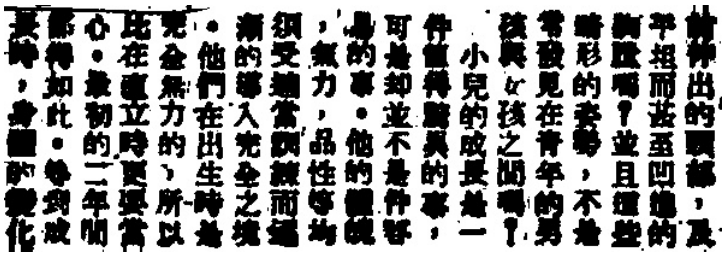
In order to counter this effect, I propose a method that ensures most background pixels stay white: Assuming that across the entire image, the number of background pixels is greater than the number of foreground pixels, pixels brighter than the median m of all pixel values are marked as part of the background and will always be set to white *after* the mean thresholding computations, no matter what the mean thresholding would originally suggest for the marked pixels to be set to (Fig. 4.7 (e)):

$$b''_{ij} = \begin{cases} 0 & \text{if } a_{ij} < \bar{t}^k_{ij} \text{ and } a_{ij} < m \\ 255 & \text{otherwise} \end{cases} \tag{4.6}$$

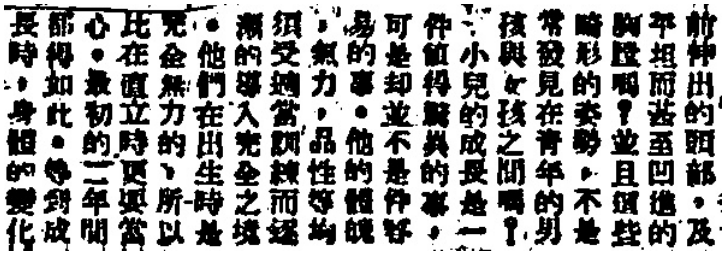
While using the globally computed median works well for the small text block crops in this thesis, it intuitively seems reasonable to use a ($\bar{k}$-sized; $\bar{k} \gg k$) running kernel for the median thresholding as soon as images get bigger and more likely to suffer from larger variance in contrast and brightness.

Finally, after manual evaluation I deemed

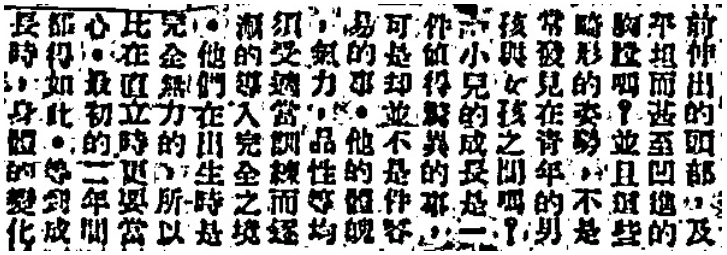
- adaptive mean binarization with $k = 125$ most suitable for deskewing, as the projection profile is more stable across the entire image for "clumpier" binarization results,



(a) $k = 125$px

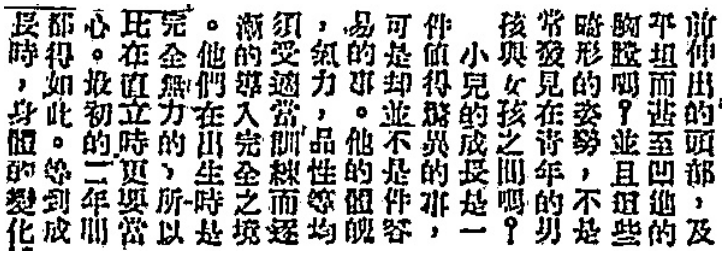


(b) $k = 25$px



(c) $k = 15$px

(d) $k = 7$px



(e) $k = 7$px; median thresholding for background preservation

Figure 4.7.: Adaptive mean thresholding using different kernel sizes (a–d) and additional median thresholding for background preservation at small kernel sizes (e)

- adaptive mean binarization with a low kernel size ($k = 7$) combined with additional median thresholding (cf. Fig 4.7 (e)) most suitable for separator finding as described in Section 4.3, as the resulting projection profile is more fine-grained since characters don't grow together as much, leaving enough spaces to find possible separators within single columns.

There is, however, always a certain amount of information loss involved with binarization (Yousefi et al., 2015), and while binarization is certainly inevitable to create projection profiles as needed for deskewing and character segmentation, the low resolution at character level makes good OCR results unlikely after binarization in my particular case. Even in Fig 4.7 (e) binarization renders many characters unreadable to the human eye, as close strokes in characters like 體 in the left-most column will invariably become all-white or all-black patches. In order to preserve a maximum of pixel information but still allow for normalized character images of similar brightness distribution, I further propose a partial thresholding and contrast enhancing method applied to the resulting character images for the OCR step:

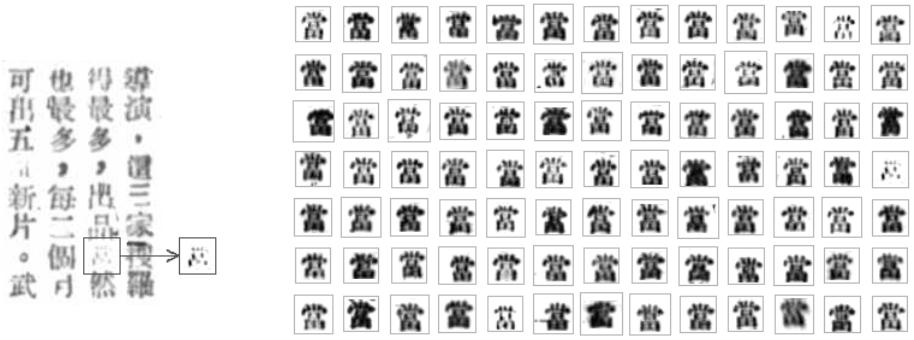
1. Globally (for the whole crop): Employ partial adaptive mean thresholding: Every pixel whose gray-scale value is larger (= brighter) than the average of a surrounding 7x7-kernel is set to 255 (white). Separately, every pixel whose value is greater than the median of the image (called threshold below) is assumed to be a background pixel and also set to 255. Every other pixel keeps its gray-scale value:

$$b'''_{ij} = \begin{cases} 0 & \text{if } a_{ij} < \bar{t}^k_{ij} \text{ and } a_{ij} < m \\ a_{ij} & \text{otherwise} \end{cases} \tag{4.7}$$

2. Locally (after cropping B''' along the separators into rectangles containing one character each): Ignoring white pixels, linearly re-scale pixel values from $[c_{min}, m]$ to $[0, 255]$, where c_{min} refers to the darkest pixel in C, a single character image. This allows even for very lightly printed characters to appear darker and have the decisive features more strongly separated from the background, cf. Fig. 4.8 (a). Thus, we eventually obtain a fully-processed character image $C' = (c'_{ij})$ through:

$$c'_{ij} = \frac{c_{ij} - c_{min}}{m - c_{min}} \in [0, 255] \text{ (after rounding)} \tag{4.8}$$

Furthermore, since the CNN described in Section 4.6.3 will need squared images as input, I add white padding to transform the rectangular character images into squares without distorting them. This results in slight size differences, as can be seen in Fig. 4.8 (b).



(a) Contrast enhancing (b) Segmented samples of the character 當

Figure 4.8.: Results after character segmentation and contrast enhancing

4.5. Character Image Generation and Augmentation

The training data automatically extracted by the method described in Section 4.3 is very limited in size – partly due to lack of more annotations, partly due to the effort of creating the crops and finding the corresponding section in the text ground truth. Tables 4.2 and 4.3 give an idea about the effective size of the training set: Table 4.2 shows the number of characters that have at least x training samples, put into perspective by Table 4.3, which shows how many of the most frequently used characters cumulatively amount for what percentage of a representative corpus. Assuming that, say, $x = 10$ samples of every character are needed during training in order for the CNN to predict it correctly at all, then Table 4.2 suggests there are not enough characters in the training data to later achieve a top-1 accuracy of over 95% ($900 < 1389$). Of course, there is no way to specify an exact minimum definitely needed for x, as it would largely depend on

- the number of target classes: more classes $\implies$ higher confusability $\implies$ need for more samples;
- intra-class variance: larger variance $\implies$ harder to generalize $\implies$ need for more samples;
- inter-class variance: lower variance due to e.g. low image resolution or blurry quality $\implies$ higher confusability $\implies$ need for more samples;

and other parameters, including the fact that some characters are harder to recognize than others. Ren et al. (2016) address this question with the following statement:

> "To train a CNN to recognize all the Chinese characters used nowadays, which is counted more than 10,000, the size of training dataset is demanded to be millions. Even for the most common used 1,200 Chinese characters, the size of training dataset is also demanded to be several hundreds of thousands."

x	number of characters that appear at least x times
1	2,797
2	2,124
3	1,752
⋮	⋮
10	900
⋮	⋮
20	527
⋮	⋮
50	209
⋮	⋮
100	70

Table 4.2.: Overview over number of training samples after character image extraction

index	character	cumulative perc. (%)
1	的	3.804
2	是	5.666
3	不	7.313
4	我	8.816
5	一	10.296
⋮	⋮	⋮
28	也	25.365
⋮	⋮	⋮
129	實	50.078
⋮	⋮	⋮
408	黃	75.005
⋮	⋮	⋮
928	毛	90.008
⋮	⋮	⋮
1,389	瞭	95.001
⋮	⋮	⋮
2,558	削	99.000

Table 4.3.: Cumulative character frequency in a corpus presented by Tsai (1996)

In other words, they demand that the training set be at least two orders of magnitude larger than the number of target classes. While at 47,986 training samples my extracted training set only reaches one order of magnitude in that respect (cf. also Section 4.6.2), intra-class variance is rather low as all the characters are printed in the same font (see Fig. 4.8), which might to some degree make up for the shortage in training data.

Apart from that, when training a NN on a fixed set of samples, there is generally a risk of overfitting: As the training proceeds over multiple epochs, the NN "memorizes" the very details of the training data distribution, eventually negatively affecting performance on a validation or test set not seen during training, an effect commonly referred to as "lack of generalization". Overfitting tends to be more problematic for smaller training sets (Perez and Wang, 2017), as more training epochs are needed until convergence is reached, thus the same samples are seen more often.

Among methods like regularization, dropout and batch normalization, one way to combat overfitting commonly seen in computer vision is data augmentation: By making randomized transformations to the training images on the fly (i.e. during, not before training), a CNN can be fed with any desired or needed amount of additional input. It goes without saying that these transformations should emulate the training data as well as possible, i.e. ideally represent the same distribution of features.

Taking all of the above into account, and motivated by related work on synthetic data generation and augmentation for PCCR (Ren et al., 2016; Xu et al., 2017; Zhong et al., 2015a), I develop a method to generate additional character image data capable of emulating the distribution of real character images. One fundamental set of operations is used from the field of mathematical morphology. This term summarizes a set of algorithms and techniques in image processing, with the main operations being dilation, erosion, opening and closing. Their origins lead back to the works of Georges Matheron and Jean Serra (Haas et al., 1967), but a more comprehensible description of the four main operations can be found in Haralick et al. (1987). Without going into too much detail, one can imagine these operations on binary images as follows:

- **Dilation**: Make every black pixel adjascent to a white pixel white as well.
- **Erosion**: Make every white pixel adjascent to a black pixel black as well.
- **Opening**: Erosion followed by dilation.
- **Closing**: Dilation followed by erosion.

In binary images, these algorithms are designed (and named) for white content pixels on black background pixels. In my case (black characters on white background), the terms are swapped pairwise.

After manually setting hyperparameters (see below) for the randomized elements in the algorithm below, an arbitrary number of character images can be sampled. Fig. 4.10 (on page 29) juxtaposes synthetic and real character images. Padding is added to the former in a way that it is randomly distributed to the top, bottom, left and right, such that after all images are resized to the same size, the small size variations (as visible in Fig. 4.8 (b)) are also emulated. Randomization hyperparameters are set under manual evaluation of the resulting output, which is naturally desired to have the largest similarity to the real-life images possible. The resulting parameters are as follows:

Let s denote the size of the image, where $s = \text{width} = \text{height}$. Let $\in_R$ denote random sampling of a single number and $\in_{R=n}$ random sampling of n numbers. Now set:

$$
\begin{aligned}
m &\in_R \{1, 2, 3\} \\
k &\in_R \{8, 9, \ldots, 17\} \\
b &\in_R \{-100, -99, \ldots, +50\}
\end{aligned}
$$

Then use these hyperparameters to augment each glyph image using the following procedure (Fig. 4.9):

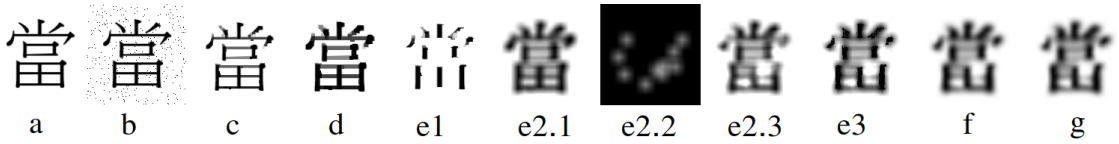


Figure 4.9.: Steps to creating the synthetic character images

a. Extract PNG images of a predefined set of glyphs from a traditional Chinese Song-style font (cf. Sections 2.2.3 and 2.2.4).
b. Add random noise ("peppering"): Sample $X, Y \in_{R=\lceil 0.01*m*s^2 \rceil} \{0, 1, ..., s\}$, then turn all pixels $(x_1, y_1), (x_2, y_2), ...$ (with $x_i \in X, y_i \in Y$) to black (0).
c. Use morphological opening to enlarge noise pixels, grow them to close black pixels (other noise or the actual character), and closing to remove useless noise again. Employ m iterations for both opening and closing. The kernel size is $(2, 2)$.
d. Use morphological erosion to thicken lines (kernel size $(3, 3)$).
e. Emphasize vertical lines while blurring and staining the remaining parts:
    1. Extract vertical elements of a certain minimum length using dilation with a vertical kernel of size $(7, 1)$.
    2. Separately apply the following to d:
        1. Further conduct erosion $(3, 3)$ and blur $(10, 10)$.
        2. Generate random patches using the following algorithm:
            A. Create a new image of size (s, s) (same size as the character image).
            B. Set the number of patches $n_p = 10$, size $s_p = 12$ and brightness $b_p = 150$.
            C. Sample $X, Y \in_{R=n_p} \{0, 1, ..., s\}$.
            D. Draw rectangles with brightness b_p: For pixels $(x_1, y_1), (x_2, y_2), ...$ (with $x_i \in X, y_i \in Y$), the upper left corner is (x_i, y_i), the lower right corner is $(x_i + s_p, y_i + s_p)$.
            E. Blur the image (kernel size $(15, 15)$).
        3. Add the patches to the image: Increase the pixel values of the original image by the values obtained from the steps A–E above (component-wise addition with cut-off at 255).
    3. Join the result and the previously extracted vertical lines back together using bitwise AND between e1 and e2.3.
f. Increase/decrease the brightness by b (not done in the above example image). Blur the image once more (kernel size (k, k)). Linearly rescale the resulting grey-scale values to $[0, 255]$, just like the real-life images.
g. Apply randomized elastic distortion as described in Simard et al. (2003).

(a) Synthetic character images sampled using the algorithm presented in Fig. 4.9

(b) "Real" character images extracted from the image scans.

Figure 4.10.: Comparison between synthetic and real character images.

4.6. Character Recognition

4.6.1. Unicode and Fonts

Both Xu et al. (2017) and Ren et al. (2016) generate synthetic character images from a rather large selection of fonts (the former use 28 and the latter 32 fonts). This is reasonable in so far as their test sets are more diverse, thus requiring better generalization across the larger intra-class variance. Within this thesis, images of only one specific font style need to be recognized, but one has to be aware that there are still considerable differences between individual traditional Chinese Song fonts, just like in Latin-script serif fonts. These differences are characterized by

1. slight variation in stroke thickness, character size and height-width ratio;
2. position and rotation of strokes due to regional variations of the same character.

1. can be addressed by simply training on images generated from multiple fonts, hoping that the augmentations will help the NN generalize enough to make up for these differences. As for 2., there are more striking differences in the way certain characters are supposed to be shaped according to different countries' standards and across historical development. Table 4.4 presents a selection of various character shapes (allographs) that are mapped to the same code point in Unicode (with the exception of columns 1 and 3). The variants true to the one printed in the *Jīngbào* are black, the others are greyed out. For the detailed explanation on graphemes, glyphs and Unicode see Section 2.2.3. Finally, I settle with using the four fonts shown in Table 4.4 for training.

[5] https://data.gov.tw/dataset/5961
[6] http://fonts.jp/hanazono/
[7] https://github.com/adobe-fonts/source-han-serif
[8] https://github.com/ichitenfont/I.Ming

Unicode point → / Font name ↓	1 U+9752 U+9751	2 U+8ACB	3 U+70BA U+7232	4 U+722D	5 U+76CA	6 U+7336	7 U+7576	8 U+66FE	9 U+5E73	10 U+4EE4	11 U+7D05	12 U+9019	13 U+798F
TW-Sung[5]	青 青	請	為 爲	爭	益	猶	當	曾	平	令	紅	這	福
HanaMin A[6]	青 青	請	為 爲	爭	益	猶	當	曾	平	令	紅	這	福
SourceHanSerif JP[7]	青 青	請	為 爲	爭	益	猶	當	曾	平	令	紅	這	福
I.Ming[8]	青 青	請	爲 爲	爭	益	猶	當	曾	平	令	紅	這	福
Jīngbào													

Table 4.4.: Glyphs as designed in four different Song fonts for several Unicode points

4.6.2. Target classes

It is not trivial to decide how many and precisely which of Unicode's 93,779 CJK characters an OCR system should be able to recognize. Xu et al. (2017) opt for 3,755 Chinese characters (presumably the 3,755 Chinese characters of the first level of GB2312-80[9]), as well as Zhong et al. (2015a). The same set of 3,755 characters is also commonly found in HCCR (Melnyk et al., 2020). GB2312-80 is established for simplified Chinese characters of contemporary use, though, whereas the *Jīngbào* (1) features traditional Chinese and (2) may contain a certain number of characters no longer (or less) commonly used today, e.g. those appearing in personal and place names frequently used at the time of printing. Hence, I settle with a union of

1. those of the 4,199 unique characters in the ground truth that are not encoded in or beyond the Unicode block *CJK Unified Ideographs Extension B* (starting with U+20000, cf. Table 2.1)[10] and
2. the most common 4,000 characters as in Tsai (1996).

modern variant	*Jīngbào* variant
值 (U+503C)	值 (U+5024)
偽 (U+507D)	僞 (U+50DE)
即 (U+5373)	卽 (U+537D)
卻 (U+537B)	却 (U+5374)
啟 (U+555F)	啓 (U+5553)
回 (U+56DE)	囘 (U+56D8)
夠 (U+5920)	够 (U+591F)
幫 (U+5E6B)	幇 (U+5E47)
強 (U+5F37)	强 (U+5F3A)
既 (U+65E2)	旣 (U+65E3)
款 (U+6B3E)	欵 (U+6B35)
汙 (U+6C59)	汚 (U+6C5A)
污 (U+6C61)	汚 (U+6C5A)
清 (U+6E05)	淸 (U+6DF8)
溫 (U+6EAB)	温 (U+6E29)
為 (U+70BA)	爲 (U+7232)
真 (U+771F)	眞 (U+771E)
眾 (U+773E)	衆 (U+8846)
鎮 (U+93AE)	鎭 (U+93AD)
青 (U+9752)	靑 (U+9751)

Table 4.5.: Variants (modern vs. *Jīngbào*)

[9] cf. the *Chinese ideogram coded character set for information interchange (basic set)* by the Standardization Administration of the People's Republic of China (1980)

[10] Other than that, there are only two instances of characters in *CJK Unified Ideographs Extension A*, namely one occurrence of each 嬗 (U+372C) and 嗀 (U+35D6). All other characters not in *Extension B* are within the 20,992 characters of the block *CJK Unified Ideographs*.

Furthermore, in some cases the scans show certain variant characters (異體字 *yìtǐzì*) with own code points (like in columns 1 and 3 of Table 4.4) whose modern traditional Chinese equivalent has to be manually excluded (see Table 4.5). Finally, this approach yields 4,806 classes for the OCR model to recognize.

4.6.3. Neural Network Architecture

As previously explained in Section 3.3.2, I use a GoogLeNet as proposed by Szegedy et al. (2014). The original GoogLeNet implementation in PyTorch[11], a library commonly used for machine learning with Python, however assumes RGB-images as an input, and consequently the first convolutional layer is designed for 3 input channels.[12] Since all of the *Jīngbào*'s scans are grey-scale images, I modify this layer to accept images with only 1 input channel. Furthermore, the input images need to be resized to the required input dimension of 224×224 pixels. The last layer is a fully connected layer outputting the logits for the 4,806 classes.

4.7. OCR Output Correction

In Section 3.4, I explained how for OCR error correction, one needs to find a heuristic for

a) deciding which characters are likely to need correction at all and
b) deciding which candidates to have the language model choose from.

The section referred to above presents different approaches to a) and b) across related work in the field. In this thesis, I will address these issues as follows:

a) In order not to "mis-correct" correct OCR predictions, I aim to separate right from wrong predictions by setting a threshold t for the difference in the confidence score of the top 1 and top 2 OCR candidate. This follows the assumption that wrong OCR predictions are likely to be of lower confidence (for experiments confirming this assumption cf. Section 5.4). Since applying the softmax to the logit outputs of the OCR model results in very high confidence scores for most predictions (> 0.9999), I employ the log-softmax instead. In fact, computing the difference between the *log-softmax scores* of the top 1 and the top 2 candidate is equivalent to computing the difference between the *raw logits*. This is because for an output vector $x \in \mathbb{R}^n$

[11] cf. https://pypi.org/project/torch/ and https://pytorch.org/
[12] for the implementation of Szegedy et al.'s architecture in `torchvision.models` see https://pytorch.org/vision/stable/models.html#id23

$(n = 4806)$ containing the logits for a single prediction, the log-softmax simply lowers every entry x_i by a value $log\left(\sum_j e^{x_j}\right)$ which is constant for all i:

$$
\begin{aligned}
&log(softmax(x_i)) - log(softmax(x_k)) \\
&\quad = log\left(\frac{e^{x_i}}{\sum_j e^{x_j}}\right) - log\left(\frac{e^{x_k}}{\sum_j e^{x_j}}\right) && (4.9)\\
&\quad = x_i - log\left(\sum_j e^{x_j}\right) - \left(x_k - log\left(\sum_j e^{x_j}\right)\right) && (4.10)\\
&\quad = x_i - x_k && (4.11)
\end{aligned}
$$

In short: Given a character image, the OCR model will produce a list of predictions. Let x be the vector containing the prediction logits in descending order, then the language model will be employed for correcting all characters where $x_1 - x_2 < t$. Hence, this approach is somewhat similar to the work of Zhuang and Zhu (2005).

b) The search space for the language model to predict the right character from will be set as the top k predictions of the OCR model. Section 3.4 argued for the suitability of a masked language model like BERT. To test this hypothesis, I test the performance of a pretrained BERT model[13] (trained on single traditional Chinese characters as tokens) on a fill-mask task: For every character in the ground truth text, mask and predict it. Results are shown in Table 4.6. As can be seen, in more than half of the cases the BERT model predicts the correct candidate in the top position and in about 4 of 5 cases it is among the top 10 predictions. I thus assume that, provided enough correct context, a BERT model could reliably improve the OCR output if the search space is suitably restricted. Just like with t, Section 5.4 in the next chapter will present experiments on choosing k.[14] Finally, it has to be noted that the variants shown in Table 4.5 have to be replaced back to the modern ones, which are those the BERT model was most likely trained on.

n	1	2	3	4	5	6	7	8	9	10	$\cdots$	20
$p(n)$	56.83	67.44	72.37	75.32	77.43	78.87	80.14	81.15	82.00	82.71	$\cdots$	82.71

Table 4.6.: $p(n) :=$ perc. (%) of masked tokens whose gold label was among the top n candidates predicted by the BERT model

[13] https://huggingface.co/ckiplab/bert-base-chinese, provided by CKIP (https://ckip.iis.sinica.edu.tw/) at the Academia Sinica in Taiwan.

[14] Note that here and in the next chapter, t and k are unrelated to binarization thresholds (t) and kernel sizes (k) defined for the methods in Section 4.4.

5. Experiments and Discussion

Training on the images generated as described in Section 4.5 means training on data randomly augmented on the fly. As ever new images are generated with a large number of parameters for randomization, it is highly unlikely for the CNN to see the same input image twice. On the other hand, during training on the fixed set of previously extracted "real-life" character images, the CNN sees the same set of images in every epoch. As a consequence, convergence is estimated to be considerably slower for training on the synthetic data.

Furthermore, no matter how well the synthetic images emulate real-life data, they are not likely to surpass it in terms of distribution congruence with the validation set. Consequently, training on real-life images is likely to lead to a higher accuracy than training on *only* synthetic data.

From these two assumptions I derive the strategy to carry out extensive pre-training on synthetic data first, and to fine-tune the network on the real-life character images after. I conduct experiments with and without pre-training before passing the best model on to post-processing using BERT. Finally, using the optimal hyperparameters I evaluate the best models on the test set and discuss results.

5.1. Pre-training on Synthetic Data

I use a batch size of 4 and set up training using stochastic gradient descent as an optimizer with a fixed learning rate of 0.001 and a momentum of 0.5. One epoch equals training on those of the 4,806 classes that the individual fonts provide a glyph for. The fonts lack glyphs for between 9 (TW-Sung) and 128 (SourceHanSerif JP) of the classes, which I consider negligible as the missing characters can be safely assumed to be rare ones and the upper bound for the resulting accuracy loss during pre-training is only $128/4806 = 2.66\%$.

Every image is augmented using the method described on page 28 and then passed to the CNN. After every epoch, the network is evaluated on the validation set and saved with the current parameters. It is then trained for as many epochs as needed until no improvement is made over 50 epochs. The model with the highest validation accuracy

is kept, all other checkpoints are discarded. The CNN is trained on an Nvidia GTX 1080 Ti. For the pre-training, one epoch takes roughly 4.2 minutes, amounting to between 24 and 48 hours for one model to finish training.

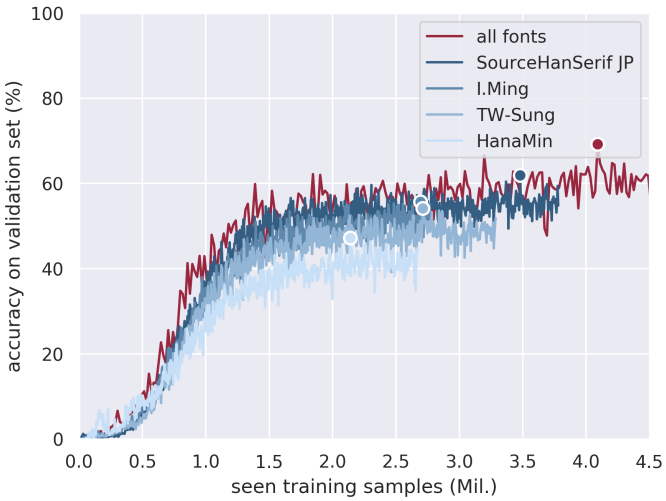
Fig. 5.1 (a) shows the validation accuracies during training on the individual fonts vs. on the entire set of glyph images from all fonts. The following observations can be made:

1. The font that performs best individually, SourceHanSerif JP, is not the one that seems the closest to the real-life data in terms of character variants (arguably I.Ming, cf. Table 4.4). This suggests that contrary to intuition, the rather limited number of variants with deviant stroke positions (cf. 2. in Section 4.6.1) is less important for emulating the real-life characters than the general structure of the entire font's glyphs (stroke width and length, character height/width ratio, size of certain components in relation to character size etc.; 1. in Section 4.6.1).
2. Training on all fonts combined yields slight improvement over training on only the best performing font (SourceHanSerif JP). Higher peaks after some epochs yield considerably higher accuracies. These "lucky catches" are most likely based on the strong randomization (and thus variability) of the training set.
3. As assumed, the validation accuracy barely decreases after converging, suggesting that there is not a lot of overfitting. Longer training times might reveal a more pronounced decline, which was not tested for practical and resource reasons.

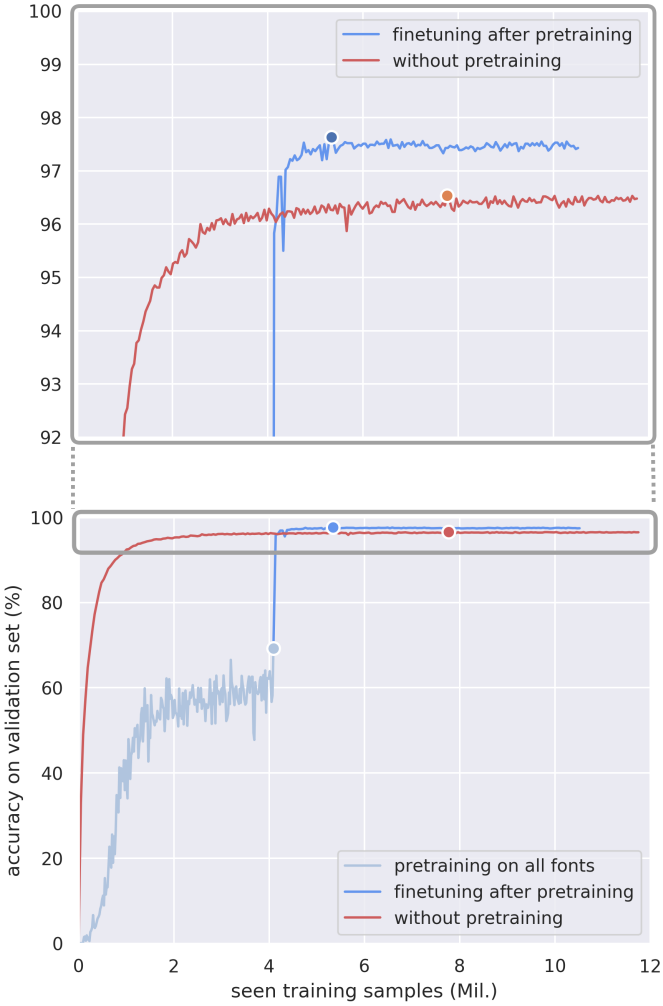
5.2. Fine-tuning on Extracted Character Images

Using the same training settings (batch size, optimizer, learning rate, momentum) as for pre-training, I continue training on the set of 47,986 characters—once from scratch, once continuing from the maximum accuracy model yielded during pre-training on all fonts (the red dot in Fig. 5.1 (a), which is equivalent to the light blue dot in Fig. 5.1 (b)). This time, the training images are natural and not augmented, hence they are seen repeatedly in every epoch. As a consequence of differing epoch sizes, epochs are not a suitable measure of training time, which is why Fig. 5.1 instead refers to the number of seen training samples for comparability. Fig. 5.1 (b) allows for the following observations:

1. With pre-training, the model achieves higher accuracy. This is presumably due to the greater amount of diverse training data leading to better generalization even on the specific font in the validation data and justifies the motivation and method presented in Section 4.5.



(a) Pre-training on artificial image data



(b) Training on real-life data with and without pre-training

Figure 5.1.: Development of validation accuracies during training. Maxima are marked with a larger dot, for numerical values cf. Table 5.1.
"All fonts" in (a) is equivalent to "pretraining on all fonts" in (b).

	font name	$k=1$	2	3	4	5	6	7	8	9	10
synthetic	TW-Sung	54.40	64.86	69.51	72.19	74.18	75.65	76.79	77.76	78.57	79.30
	HanaMin A	47.69	59.45	64.87	68.10	70.42	72.17	73.46	74.60	75.54	76.22
	SourceHanSerif JP	**62.62**	**70.64**	**74.09**	**76.23**	**77.76**	**78.94**	**79.93**	**80.61**	**81.24**	**81.69**
	I.Ming	55.60	66.41	70.90	73.84	76.09	77.62	78.83	79.86	80.60	81.23
	$(*)$ all fonts	**69.73**	**78.30**	**81.68**	**83.65**	**84.99**	**86.06**	**86.87**	**87.49**	**87.97**	**88.46**
real	without pre-training	96.54	97.32	97.49	97.58	97.64	97.65	97.70	97.70	97.71	97.71
	after pre-tr. on $(*)$	**97.63**	**98.57**	**98.78**	**98.91**	**98.98**	**99.01**	**99.07**	**99.10**	**99.12**	**99.13**

Table 5.1.: Top-k accuracy of the OCR classifier on the validation set

2. The model converges faster after pre-training, even when taking the time spent on pre-training into account: Once the maximum is reached, no improvement is observable for more than 4 million seen samples. The model trained on real-life data from scratch however shows very slow convergence and new slight improvements (mostly by 0.01%) are reached over a long time.
3. During fine-tuning, the amplitude of accuracy values between epochs is visibly lower than during pre-training, where peaks and lows are deviating stronger from each other. This can be ascribed to the randomization which leads to a training set changing between epochs, in contrast to the static real-life training set.

5.3. OCR Results

Extending what can be seen in Fig. 5.1, Table 5.1 quantifies the entire results on the validation set. Most importantly, top k accuracy is presented for $k = 2, \ldots, 10$, too, which will be important for error correction using the BERT model. Generally, it is evident that accuracies are substantially lower when only training on synthetic data. This is supposedly due to the fact that the feature distribution will always be worse than that of real-life data which is naturally more similar to the validation set the model is evaluated on. As stated before, the best results are achieved when pre-training on randomly augmented glyph images of all four fonts and fine-tuning on the segmented real-life character images: Pre-training raises the top-1 validation accuracy from 96.54% to 97.63% which is equivalent to an error reduction of 31.5%. Final results on the test set will be presented in the next section.

As mentioned above, Table 5.1 also shows top k accuracy values for $k > 1$. Unsurprisingly, a higher k leads to a considerably higher accuracy for all of the models. This is because for incorrect OCR predictions, the correct candidate is often predicted in second or third place, as can be seen in the examples in Table 5.2. This intuitively justifies the approach described in the next section.

sample	top 10 candidates
	肯 貴 骨 片 督 昔 皆 貨 旨 嘗
	胎 貽 船 貼 晤 賠 始 販 斯 脂
	油 汕 別 遇 洲 勃 海 前 西 効
	棒 梭 慘 核 稼 橡 桂 俸 梓 控
	蒲 蓮 薄 通 滯 謝 浦 逝 蕩 鼎
	數 歎 歌 欸 歡 欽 教 歉 默 歔

sample	top 10 candidates
	開 閱 聞 則 閔 題 圍 四 期 閾
	廖 麼 慶 廳 鑒 豐 麼 應 禦 農
	買 貲 貨 質 賞 贊 貸 貿 賢 實
	盛 懿 驚 盤 璇 離 鑑 蓬 蹤 蘇
	換 撲 模 摟 瑛 摸 漢 搜 樓 揍
	申 甲 中 串 弔 早 冉 里 叩 官

Table 5.2.: Examples of wrong OCR predictions

5.4. OCR Error-Correction Using a BERT Model

Let x_1 and x_2 denote the logit scores of the top 2 candidates output by the OCR model. As introduced in the last chapter (Section 4.7), I now aim to find characters likely to have been predicted incorrectly by the OCR model by setting a threshold t for the difference between x_1 and x_2. Any OCR prediction where $x_1 - x_2 < t$ is treated as likely to be incorrect and is passed on to the correction step. This step works by having a pre-trained BERT model[1] re-predict the character from the top k OCR candidates. If $x_1 - x_2 > t$, the OCR prediction is assumed to be correct and will serve as part of the context necessary for the language model (LM). This section will deal with finding suitable values for t and k and present results.

First, I check the assumption that $x_1 - x_2$ is smaller if the top candidate is not correct ($\neq$ the gold label). Fig. 5.2 (a) and (b) show histograms of the values of $x_1 - x_2$ for every character in the validation set (bin size on the x-axis: 0.2). While the different scaling of the y-axis has to be taken into account, it is evident that the above assumption holds, which is congruent with the intuition that correct predictions are higher in confidence—or less formally, the model is "surer" about its prediction when it's correct than when it's wrong. Supporting this finding, manual evaluation revealed that among wrong predictions with high confidence there is a considerable amount of annotation errors—i.e. the model predicted the correct character but the prediction was classified as wrong due to an incorrect gold label. However, there are only 227 cases (1% of the validation set) where for a wrong prediction $x_1 - x_2 > 3$.

Depending on the threshold t, a differently sized proportion of the OCR output is re-evaluated by the LM. The extremal cases are trivial:

1. $t = 0 \implies$ The entire OCR output is assumed to be correct, the LM is never used.
2. $t \geq \max_{i,j}(x_i - x_j) \implies$ The entire OCR output is re-evaluated by the LM:
   a) $k = 1 \implies$ The LM only gets one candidate to choose from, equivalent to 1.
   b) $k = |V|$ (k is equal to the LM's vocabulary size) $\implies$ Every character is re-evaluated, the final accuracy would be somewhere near $p(1)$ in Table 4.6.

Systematically testing different values of t and k with $t \in [0, 0.5, \ldots, 10]$ and $k \in [0, 1, \ldots, 18]$, I obtain the results shown in Fig. 5.2 (c). For comparison, the original OCR accuracy of 97.63% is marked as a horizontal line. It is evident that within certain intervals of t and k, the LM indeed lowers the error on the validation set, i.e. post-processing is successful. Optimal values for t and k are obtained by the following observations:

[1] cf. the footnote on page 32

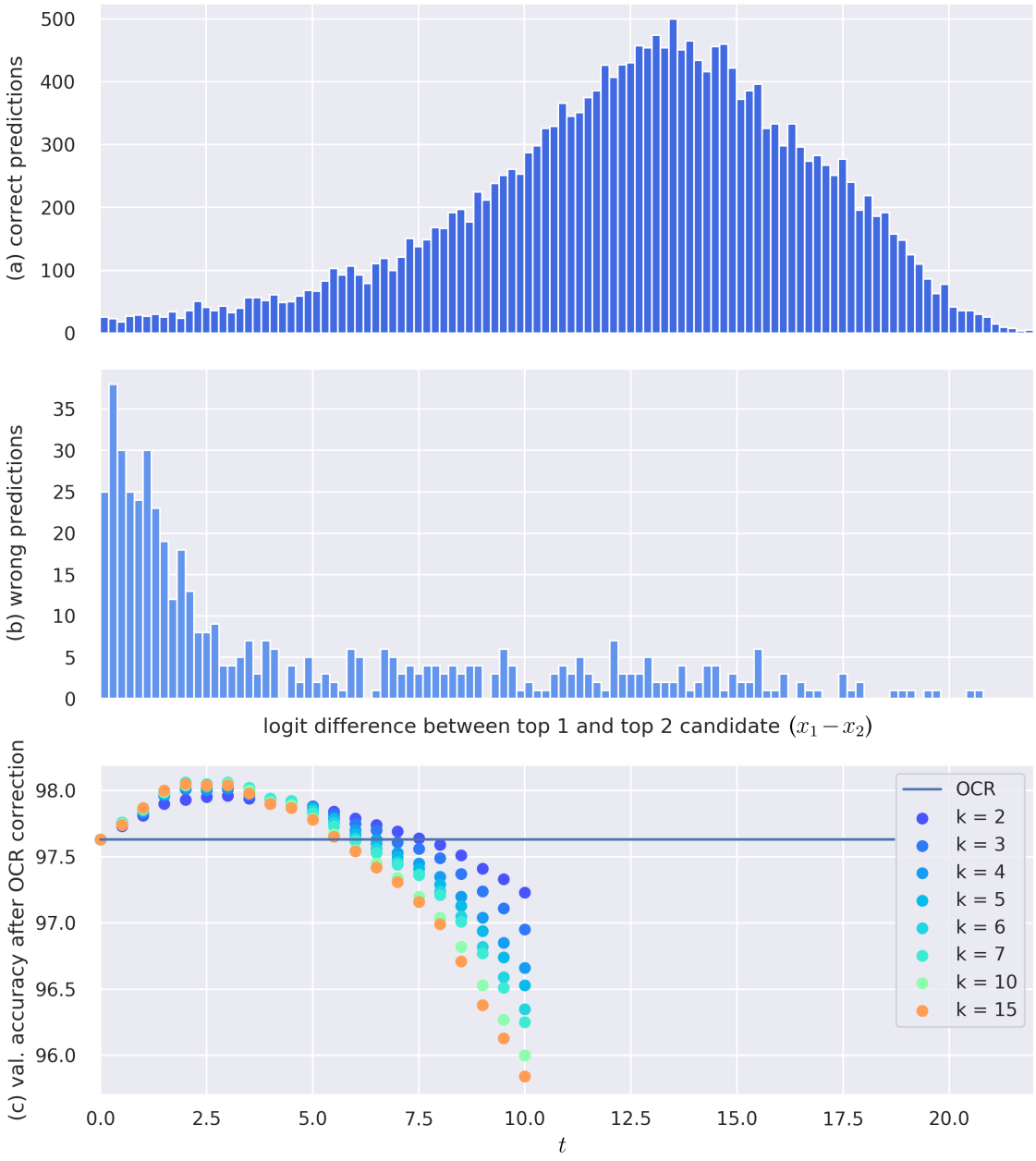


Figure 5.2.: Values for $x_1 - x_2$ for right (a) and wrong (b) predictions and accuracy on validation set for different t after LM-correction (c) (numeric values up to $k = 18$ along with $k = |V|$ can be found in Appendix A.1).

t: Independent of k, the best performance is consistently obtained for $2.0 \leq t \leq 3.0$, which is in line with the observations in Fig. 5.2 (a) and (b) where this is the region that intuitively best separates correct from incorrect predictions. As within this interval there is no clear tendency of a specific maximum (cf. also Appendix A.1), I settle with the midpoint of $\underline{t = 2.5}$.

k: The best performance is obtained for $\underline{k = 7}$, albeit by a very slight margin over any $3 \leq k \leq 18$. As for lower and higher k: At $k = 2$ the LM correction does improve performance over the basic OCR model, but while in wrong predictions the correct candidate is often in second place (cf. Table 5.2 on page 36), this misses out on the cases where it is further back. Within $3 \leq k \leq 18$, the LM reveals its full potential: The small number of visually similar characters are not likely to be semantically similar and thus easily identifiable by the context-aware LM. For $k > 7$, the top accuracy value ($2.0 \leq t \leq 3.0$) slowly decreases, presumably because given greater choice, the LM becomes ever more likely to find a semantically fitting (but incorrect) candidate, until more and more initially correct OCR predictions are "mis-corrected" and the overall accuracy drops below the initial OCR accuracy. At $k = |V|$, no improvement over the OCR model is made for any t (cf. A.1).

5.5. Final Results on the Test Set

Resulting from the observations described in the last section, the best model (as of Table 5.1) was evaluated on the test set using $t = 2.5$ and $k = 7$. Of the 22,377 images in the test set, 96.95% were correctly identified by the OCR model. 767 characters were passed to the LM for re-evaluation (see paragraph below), whereafter a total accuracy of 97.44% is achieved. As becomes apparent in Table 5.3, these values are lower than on the validation set, where for $t = 2.5, k = 7$ accuracies of 97.63% and 98.05% were attained, respectively. Presumably, the crops in the test set happened to originate from less clearly printed sections of the newspaper. Nevertheless, the overall correction rate

	val. set	test set
only OCR w/o pre-training	96.54	95.49
only OCR w/ pre-training	97.63	96.95
OCR w/ pre-training + LM	**98.05**	**97.44**

Table 5.3.: Classification accuracy (%) on validation and test set

		after applying LM: right	after applying LM: wrong
OCR result:	right	292	82
	wrong	174	54+165*

Table 5.4.: Confusion table for the LM correction step (*54 left unchanged, 165 changed to another wrong character)

of the LM step is consistent between validation and test set, with error reduction rates of 18.1% (validation set) and 16.1% (test set).

Finally, Table 5.4 gives additional insight into what happens at the LM correction step: Of the OCR output passed to the LM, correct and incorrect characters nearly make up one half each (374 right, 393 wrong). After applying the LM, this ration shifts to about 60–40 (466 correct, 311 incorrect). Precisely, 44% of wrong OCR output passed to the LM is corrected, but also 22% of the characters correctly predicted by the OCR model are changed to incorrect ones. Though ultimately, positive changes outweigh negative ones, the absolute amount of correct OCR output "botched" by the LM leaves room for improvement.

6. Conclusion and Outlook

This thesis investigated an approach to building a character-level OCR classifier and reducing errors in the output by using a pre-trained BERT language model. In the course of the experiments, I have shown how pre-training on large amounts of synthesized image data and LM post-processing considerably reduce the classification error, and how to set hyperparameters for the latter. The results allow for optimistic outlooks on extracting the full text from the *Jīngbào* and similar newspapers, as they will significantly reduce human annotation work (at 97.5% accuracy, only one out of 40 characters is wrong). There is, however, still an urgent need for efficient segmentation methods on higher layout levels which this work did not address. Existing NN-based architectures such as *eynollah* presented in the related work section might solve this remaining problem in the future. Apart from that, future challenges also include:

- the fact that the grid layout is not generally given throughout the entire *Jīngbào* or other newspapers, hence NN-based character detection such as the *HRCenterNet* (cf. Section 3.2) will be necessary;
- the fact that other parts of the *Jīngbào* and other newspapers use different fonts that even vary within the same structurally connected segments, such as heading fonts vs. text fonts in article blocks or different fonts in advertisements and marginalia;
- possible improvements in LM-based error correction, e.g. by increasing search space beyond the top k candidate list of the OCR model, considering that it does not generally output all visually similar characters in question;
- the question if, in the case of totally absent training data, synthetic data alone would be able to provide enough correct output that post-processing using LMs is effective. Table 5.1 yields 88.46% top 10 accuracy when training on only synthetic data, but a masked LM like BERT might not have enough *correct* context for reliable mask prediction. Chapter 3 of this work however presents other methods that–in combination with BERT–may still be able to identify a sufficient number of correct candidates.

Acknowledgements

Throughout the months I have spent working on this thesis, several people have been particularly helpful to me, answering questions, sharing knowledge or providing additional insight into their own work.

First of all, I would like to thank Matthias Arnold, who always took his time when I needed his advice. As a supervisor—also in the ECPO project—he introduced me to a fascinating niche in the field of digital humanities and was always of great help answering my questions concerning newspaper digitization projects and Chinese printing. Not only did he provide me with relevant scientific work, but he also invited me to attend several conferences on related topics. It was through him that I gained a perspective on the larger picture that this thesis will eventually form a part of.

Secondly, I want to mention Prof. Dr. Anette Frank, who provided me with the computational linguist's view on the topic, which was particularly relevant for the language model correction step. She is always full of spontaneous ideas for interesting approaches.

Finally, I want to express my thanks to all the student assistants who have been working hours over hours to create and review the text ground truth for the *Jīngbào*, as well as to Julia Baum for her fantastic proofreading skills and her hints on the subtleties of mathematical notation and the English language.

A. Appendix

A.1. Numeric Values for Fig. 5.2 (c)

$k = 2$:		$k = 3$:		$k = 4$:		$k = 5$:		$k = 6$:		$k = 7$:	
t	v. a.*	t	v. a.*	t	v. a.*	t	v. a.*	t	v. a.*	t	v. a.*
0.0	97.63	0.0	97.63	0.0	97.63	0.0	97.63	0.0	97.63	0.0	97.63
0.5	97.73	0.5	97.75	0.5	97.75	0.5	97.76	0.5	97.76	0.5	97.76
1.0	97.81	1.0	97.83	1.0	97.84	1.0	97.85	1.0	97.85	1.0	97.86
1.5	97.90	1.5	97.96	1.5	97.97	1.5	97.98	1.5	97.98	1.5	97.99
2.0	97.93	2.0	98.01	2.0	98.02	2.0	98.04	2.0	98.04	2.0	98.06
2.5	97.95	2.5	98.00	2.5	98.01	2.5	98.03	2.5	98.03	2.5	98.05
3.0	97.96	3.0	98.01	3.0	98.02	3.0	98.04	3.0	98.05	3.0	98.06
3.5	97.94	3.5	97.98	3.5	97.97	3.5	98.00	3.5	98.01	3.5	98.02
4.0	97.90	4.0	97.93	4.0	97.92	4.0	97.93	4.0	97.94	4.0	97.94
4.5	97.90	4.5	97.92	4.5	97.90	4.5	97.91	4.5	97.92	4.5	97.92
5.0	97.88	5.0	97.88	5.0	97.86	5.0	97.87	5.0	97.85	5.0	97.83
5.5	97.84	5.5	97.82	5.5	97.79	5.5	97.79	5.5	97.76	5.5	97.73
6.0	97.79	6.0	97.75	6.0	97.70	6.0	97.67	6.0	97.64	6.0	97.62
6.5	97.74	6.5	97.70	6.5	97.63	6.5	97.60	6.5	97.56	6.5	97.53
7.0	97.69	7.0	97.61	7.0	97.53	7.0	97.50	7.0	97.46	7.0	97.44
7.5	97.64	7.5	97.56	7.5	97.45	7.5	97.41	7.5	97.37	7.5	97.36
8.0	97.59	8.0	97.49	8.0	97.35	8.0	97.29	8.0	97.24	8.0	97.21
8.5	97.51	8.5	97.37	8.5	97.20	8.5	97.13	8.5	97.05	8.5	97.01
9.0	97.41	9.0	97.24	9.0	97.04	9.0	96.94	9.0	96.82	9.0	96.77
9.5	97.33	9.5	97.11	9.5	96.85	9.5	96.74	9.5	96.59	9.5	96.51
10.0	97.23	10.0	96.95	10.0	96.66	10.0	96.53	10.0	96.35	10.0	96.25

*v. a. = validation accuracy

$k = 8$:		$k = 9$:		$k = 10$:		$k = 11$:		$k = 12$:		$k = 13$:	
t	v. a.*	t	v. a.*	t	v. a.*	t	v. a.*	t	v. a.*	t	v. a.*
0.0	97.63	0.0	97.63	0.0	97.63	0.0	97.63	0.0	97.63	0.0	97.63
0.5	97.74	0.5	97.75	0.5	97.75	0.5	97.74	0.5	97.74	0.5	97.75
1.0	97.85	1.0	97.86	1.0	97.86	1.0	97.85	1.0	97.87	1.0	97.87
1.5	97.98	1.5	97.99	1.5	97.99	1.5	97.97	1.5	97.98	1.5	97.98
2.0	98.04	2.0	98.05	2.0	98.04	2.0	98.03	2.0	98.03	2.0	98.03
2.5	98.04	2.5	98.05	2.5	98.04	2.5	98.03	2.5	98.04	2.5	98.03
3.0	98.05	3.0	98.06	3.0	98.05	3.0	98.03	3.0	98.04	3.0	98.03
3.5	98.00	3.5	98.01	3.5	98.00	3.5	97.98	3.5	97.99	3.5	97.98
4.0	97.92	4.0	97.92	4.0	97.91	4.0	97.90	4.0	97.91	4.0	97.90
4.5	97.90	4.5	97.89	4.5	97.89	4.5	97.87	4.5	97.88	4.5	97.87
5.0	97.81	5.0	97.80	5.0	97.79	5.0	97.77	5.0	97.79	5.0	97.77
5.5	97.70	5.5	97.68	5.5	97.67	5.5	97.65	5.5	97.66	5.5	97.65
6.0	97.58	6.0	97.57	6.0	97.55	6.0	97.53	6.0	97.55	6.0	97.54
6.5	97.49	6.5	97.49	6.5	97.45	6.5	97.43	6.5	97.45	6.5	97.43
7.0	97.40	7.0	97.38	7.0	97.34	7.0	97.32	7.0	97.35	7.0	97.31
7.5	97.30	7.5	97.26	7.5	97.20	7.5	97.18	7.5	97.20	7.5	97.16
8.0	97.15	8.0	97.11	8.0	97.04	8.0	97.01	8.0	97.03	8.0	97.00
8.5	96.95	8.5	96.91	8.5	96.82	8.5	96.78	8.5	96.80	8.5	96.74
9.0	96.66	9.0	96.61	9.0	96.53	9.0	96.47	9.0	96.50	9.0	96.41
9.5	96.41	9.5	96.36	9.5	96.27	9.5	96.20	9.5	96.21	9.5	96.15
10.0	96.16	10.0	96.11	10.0	96.00	10.0	95.92	10.0	95.91	10.0	95.85

$k = 14$:		$k = 15$:		$k = 16$:		$k = 17$:		$k = 18$:		$k = \|V\|$:	
t	v. a.*	t	v. a.*	t	v. a.*	t	v. a.*	t	v. a.*	t	v. a.*
0.0	97.63	0.0	97.63	0.0	97.63	0.0	97.63	0.0	97.63	0.0	97.63
0.5	97.74	0.5	97.74	0.5	97.74	0.5	97.74	0.5	97.74	0.5	97.62
1.0	97.88	1.0	97.87	1.0	97.87	1.0	97.88	1.0	97.86	1.0	97.55
1.5	98.00	1.5	98.00	1.5	98.00	1.5	98.00	1.5	97.99	1.5	97.49
2.0	98.05	2.0	98.05	2.0	98.04	2.0	98.04	2.0	98.01	2.0	97.35
2.5	98.04	2.5	98.04	2.5	98.04	2.5	98.03	2.5	98.01	2.5	97.16
3.0	98.03	3.0	98.04	3.0	98.03	3.0	98.03	3.0	98.00	3.0	96.95
3.5	97.98	3.5	97.98	3.5	97.98	3.5	97.97	3.5	97.95	3.5	96.73
4.0	97.90	4.0	97.90	4.0	97.91	4.0	97.90	4.0	97.89	4.0	96.45
4.5	97.87	4.5	97.87	4.5	97.88	4.5	97.87	4.5	97.85	4.5	96.19
5.0	97.78	5.0	97.78	5.0	97.79	5.0	97.78	5.0	97.76	5.0	95.81
5.5	97.65	5.5	97.65	5.5	97.66	5.5	97.65	5.5	97.62	5.5	95.34
6.0	97.54	6.0	97.54	6.0	97.55	6.0	97.53	6.0	97.51	6.0	94.82
6.5	97.43	6.5	97.42	6.5	97.43	6.5	97.42	6.5	97.39	6.5	94.34
7.0	97.31	7.0	97.31	7.0	97.31	7.0	97.30	7.0	97.27	7.0	93.74
7.5	97.17	7.5	97.16	7.5	97.16	7.5	97.14	7.5	97.12	7.5	92.94
8.0	96.99	8.0	96.99	8.0	96.99	8.0	96.97	8.0	96.94	8.0	92.18
8.5	96.72	8.5	96.71	8.5	96.71	8.5	96.68	8.5	96.64	8.5	91.09
9.0	96.40	9.0	96.38	9.0	96.38	9.0	96.36	9.0	96.32	9.0	89.99
9.5	96.15	9.5	96.13	9.5	96.14	9.5	96.11	9.5	96.07	9.5	88.82
10.0	95.87	10.0	95.84	10.0	95.85	10.0	95.81	10.0	95.74	10.0	87.59

Bibliography

Allen, Julie D, Deborah Anderson, Joe Becker, Richard Cook, Mark Davis, Peter Edberg, Michael Everson, Asmus Freytag, Laurentiu Iancu, Richard Ishida, et al. (2012). "The Unicode Standard". In: *Mountain view, CA*, pp. 660–664.

Arnold, Matthias and Lena Hessel (2019). "Transforming data silos into knowledge: Early Chinese Periodicals Online (ECPO)". In: *E-Science-Tage 2019: Data to Knowledge, Heidelberg: heiBOOKS, 2020*. DOI: 10.11588/heibooks.598.c8420.

Casey, Richard and George Nagy (1966). "Recognition of printed Chinese characters". In: *IEEE Transactions on Electronic Computers* 1, pp. 91–101. DOI: 10.1109/PGEC.1966.264379.

Chen, Kai, Mathias Seuret, Jean Hennebert, and Rolf Ingold (2017). "Convolutional Neural Networks for Page Segmentation of Historical Document Images". In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 01, pp. 965–970. DOI: 10.1109/ICDAR.2017.161.

Chu, Chenhui, Toshiaki Nakazawa, and Sadao Kurohashi (May 2012). "Chinese Characters Mapping Table of Japanese, Traditional Chinese and Simplified Chinese". In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Istanbul, Turkey: European Language Resources Association (ELRA), pp. 2149–2152. URL: https://aclanthology.org/L12-1140/.

Dai, Ruwei, Chenglin Liu, and Baihua Xiao (2007). "Chinese character recognition: history, status and prospects". In: *Frontiers of Computer Science in China* 1.2, pp. 126–136. DOI: 10.1007/s11704-007-0012-5. URL: https://www.researchgate.net/profile/Cheng-Lin-Liu-4/publication/220412438.

de Campos, Teófilo E., Bodla Rakesh Babu, and Manik Varma (2009). "Character Recognition in Natural Images". In: *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications - Volume 2: VISAPP, (VISIGRAPP 2009)*. INSTICC. SciTePress, pp. 273–280. DOI: 10.5220/0001770102730280. URL: https://www.scitepress.org/Papers/2009/17701/17701.pdf.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv: 2010.11929.

Eskenazi, Sébastien, Petra Gomez-Krämer, and Jean-Marc Ogier (2017). "A comprehensive survey of mostly textual document segmentation algorithms since 2008". In: *Pattern Recognition* 64, pp. 1–14. DOI: 10.1016/j.patcog.2016.10.023.

Fan, Kuo-Chin, Liang-Shen Wang, and Yin-Tien Tu (1998). "Classification of machine-printed and handwritten texts using character block layout variance". In: *Pattern Recognition* 31.9, pp. 1275–1284. DOI: 10.1016/S0031-3203(97)00143-X.

Fukushima, Kunihiko and Sei Miyake (1982). "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, pp. 267–285. DOI: doi:10.1007/BF00344251.

Gupta, Maya R, Nathaniel P Jacobson, and Eric K Garcia (2007). "OCR binarization and image pre-processing for searching historical documents". In: *Pattern Recognition* 40.2, pp. 389–397. DOI: 10.1016/j.patcog.2006.04.043.

Haas, A, G Matheron, and J Serra (1967). "Morphologie mathématique et granulométries en place". In: *Annales des mines*. Vol. 11. 736-753, pp. 7–3.

Haralick, Robert M, Stanley R Sternberg, and Xinhua Zhuang (1987). "Image analysis using mathematical morphology". In: *IEEE transactions on pattern analysis and machine intelligence* 4, pp. 532–550. DOI: 10.1109/TPAMI.1987.4767941.

He, Sheng and Lambert Schomaker (2018). *Open Set Chinese Character Recognition using Multi-typed Attributes*. arXiv: 1808.08993.

Hong, Yuzhong, Xianguo Yu, Neng He, Nan Liu, and Junhui Liu (Nov. 2019). "FASPell: A Fast, Adaptable, Simple, Powerful Chinese Spell Checker Based On DAE-Decoder Paradigm". In: *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*. Hong Kong, China: Association for Computational Linguistics, pp. 160–169. DOI: 10.18653/v1/D19-5522. URL: https://aclanthology.org/D19-5522.

Jaderberg, Max, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman (2014). *Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition*. arXiv: 1406.2227.

Khawaja, Attaullah, AF Chando, Alataf Rajpar, and Ali Raza Jafri (2006a). "Classification of printed Chinese characters by using neural network". In: *Proc. of the 5th WSEAS International Conference on Signal Processing, Istanbul, Turkey, May*. Citeseer, pp. 27–29. URL: http://www.wseas.us/e-library/conferences/2006istanbul/papers/521-144.pdf.

Khawaja, Attaullah, Shen Tingzhi, Noor Mohammad Memon, and Altaf Rajpar (2006b). "Recognition of printed Chinese characters by using Neural Network". In: *2006 IEEE International Multitopic Conference*. IEEE, pp. 169–172. DOI: 10.1109/INMIC.2006.358156.

Kissos, Ido and Nachum Dershowitz (2016). "OCR error correction using character correction and feature-based word classification". In: *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*. IEEE, pp. 198–203. DOI: 10.1109/DAS.2016.44.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25, pp. 1097–1105. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

LeCun, Yann, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

Li, Shutao, Qinghua Shen, and Jun Sun (2007). "Skew detection using wavelet decomposition and projection profile analysis". In: *Pattern recognition letters* 28.5, pp. 555–562. DOI: 10.1016/j.patrec.2006.10.002.

Lin, Alvin (1999). *Writing Taiwanese*. URL: http://www.sino-platonic.org/complete/spp089_taiwanese.pdf.

Lin, Chi-Fang, Yu-Fan Fang, and Yau-Tarng Juang (2001). "Chinese text distinction and font identification by recognizing most frequently used characters". In: *Image and Vision Computing* 19.6, pp. 329–338. DOI: 10.1016/S0262-8856(00)00082-2.

Liu, Cheng-Lin, Fei Yin, Da-Han Wang, and Qiu-Feng Wang (2011). "CASIA online and offline Chinese handwriting databases". In: *2011 International Conference on Document Analysis and Recognition*. IEEE, pp. 37–41. DOI: 10.1109/ICDAR.2011.17.

Liu, Zongyi, Hanning Zhou, and Ning Yang (2010). "Semi-supervised learning for text-line detection". In: *Pattern recognition letters* 31.11, pp. 1260–1273. DOI: 10.1016/j.patrec.2010.03.015.

Lua, Siok-ling and Un-gian Iunn (2012). "On the Recommended Han Characters for Taiwan Southern Min Promulgated by Taiwan's Ministry of Education". In: 海翁台語文學. DOI: 10.29919/wtl.201204.0001.

Ma, Weihong, Hesuo Zhang, Lianwen Jin, Sihang Wu, Jiapeng Wang, and Yongpan Wang (2020). "Joint Layout Analysis, Character Detection and Recognition for Historical Document Digitization". In: *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, pp. 31–36. DOI: 10.1109/ICFHR2020.2020.00017.

Mei, Yuan, Xinhui Wang, and Jin Wang (2013). "A Chinese character segmentation algorithm for complicated printed documents". In: *International Journal of Signal Processing, Image Processing and Pattern Recognition* 6.3, pp. 91–100. URL: http://article.nadiapub.com/IJSIP/vol6_no3/8.pdf.

Melnyk, Pavlo, Zhiqiang You, and Keqin Li (2020). "A high-performance CNN method for offline handwritten Chinese character recognition and visualization". In: *soft computing* 24.11, pp. 7977–7987. DOI: 10.1007/s00500-019-04083-3.

Nagy, G (1988). "Chinese character recognition: a twenty-five-year retrospective." In: *ICPR*, pp. 163–167. DOI: 10.1109/ICPR.1988.28196.

Oliveira, Sofia Ares, Benoit Seguin, and Frederic Kaplan (2018). "dhSegment: A generic deep-learning approach for document segmentation". In: *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, pp. 7–12. DOI: 10.1109/ICFHR-2018.2018.00011.

Otsu, Nobuyuki (1979). "A threshold selection method from gray-level histograms". In: *IEEE transactions on systems, man, and cybernetics* 9.1, pp. 62–66. DOI: 10.1109/TSMC.1979.4310076.

Perez, Luis and Jason Wang (2017). *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. arXiv: 1712.04621.

Pletschacher, Stefan and Apostolos Antonacopoulos (2010). "The page (page analysis and ground-truth elements) format framework". In: *2010 20th International Conference on Pattern Recognition*. IEEE, pp. 257–260. DOI: 10.1109/ICPR.2010.72.

Pratikakis, Ioannis, Konstantinos Zagoris, Xenofon Karagiannis, Lazaros Tsochatzidis, Tanmoy Mondal, and Isabelle Marthot-Santaniello (2019). "ICDAR 2019 Competition on Document Image Binarization (DIBCO 2019)". In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1547–1556. DOI: 10.1109/ICDAR.2019.00249.

Redmon, Joseph and Ali Farhadi (2018). *YOLOv3: An Incremental Improvement*. arXiv: 1804.02767.

Rehm, Georg, Peter Bourgonje, Stefanie Hegele, Florian Kintzel, Julián Moreno Schneider, Malte Ostendorff, Karolina Zaczynska, Armin Berger, Stefan Grill, Sören Räuchle, et al. (2020). *QURATOR: innovative technologies for content and data curation*. arXiv: 2004.12195.

Ren, Xiaohang, Kai Chen, and Jun Sun (2016). *A CNN Based Scene Chinese Text Recognition Algorithm With Synthetic Data Engine*. arXiv: 1604.01891.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical*

image computing and computer-assisted intervention. Springer, pp. 234–241. DOI: 10.1007/978-3-319-24574-4.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, pp. 533–536.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. (2015). "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Simard, Patrice Y, David Steinkraus, John C Platt, et al. (2003). "Best practices for convolutional neural networks applied to visual document analysis." In: *Icdar*. Vol. 3. 2003. URL: https://www.researchgate.net/publication/2880624.

Smith, Ray (2007). "An overview of the Tesseract OCR engine". In: *Ninth international conference on document analysis and recognition (ICDAR 2007)*. Vol. 2. IEEE, pp. 629–633. DOI: 10.1109/ICDAR.2007.4376991.

Standardization Administration of the People's Republic of China (1980). 信息交换用汉字编码字符集—基本集 *(Chinese ideogram coded character set for information interchange (basic set))*. URL: https://archive.org/details/GB2312-1980.

Sung, Doris, Liying Sun, and Matthias Arnold (2014). "The Birth of a Database of Historical Periodicals: Chinese Women's Magazines in the Late Qing and Early Republican Period". In: *Tulsa Studies in Women's Literature* 33.2, pp. 227–237. URL: https://www.jstor.org/stable/43653333.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2014). *Going Deeper with Convolutions*. arXiv: 1409.4842.

Tang, Chia-Wei, Chao-Lin Liu, and Po-Sen Chiu (2020). "HRCenterNet: An Anchorless Approach to Chinese Character Segmentation in Historical Documents". In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 1924–1930. DOI: 10.1109/BigData50022.2020.9378051.

Tseng, Yuen-Hsien (2002). "Error correction in a chinese ocr test collection". In: *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 429–430. DOI: 10.1145/564376.564478.

Van Phan, Truyen, Bilan Zhu, and Masaki Nakagawa (2011). "Development of Nom character segmentation for collecting patterns from historical document pages". In: *Proceedings of the 2011 Workshop on Historical Document Imaging and Processing*, pp. 133–139. DOI: 10.1145/2037342.2037365.

Wang, Hsiang-An and Pin-Ting Liu (2019). "Towards a Higher Accuracy of Optical Character Recognition of Chinese Rare Books in Making Use of Text Model". In: *Proceed-*

ings of the 3rd International Conference on Digital Access to Textual Cultural Heritage, pp. 15–18. DOI: 10.1145/3322905.3322922. URL: https://dl.acm.org/doi/pdf/10.1145/3322905.3322922.

Wang, Tie-Qiang, Fei Yin, and Cheng-Lin Liu (2017). "Radical-based Chinese character recognition via multi-labeled learning of deep residual networks". In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 1. IEEE, pp. 579–584. DOI: 10.1109/ICDAR.2017.100.

Wilkinson, Endymion Porter (2018). *Chinese history: A new manual*. Harvard University Asia Center Cambridge, MA. DOI: 10.1017/S0021911820000224.

Wong, Sebastien C, Adam Gatt, Victor Stamatescu, and Mark D McDonnell (2016). "Understanding data augmentation for classification: when to warp?" In: *2016 international conference on digital image computing: techniques and applications (DICTA)*. IEEE, pp. 1–6. DOI: 10.1109/DICTA.2016.7797091.

Xu, Ning and Xiaoqing Ding (1992). "Printed Chinese character recognition via the cooperative block neural networks". In: *[1992] Proceedings of the IEEE International Symposium on Industrial Electronics*. IEEE, pp. 231–235. DOI: 10.1109/ISIE.1992.279580.

Xu, Xin, Jun Zhou, Hong Zhang, and Xiaowei Fu (2017). "Chinese characters recognition from screen-rendered images using inception deep learning architecture". In: *Pacific Rim Conference on Multimedia*. Springer, pp. 722–732. DOI: 10.1007/978-3-319-77380-3_69. URL: https://www.researchgate.net/publication/325056786.

Yang, Hailin, Lianwen Jin, Weiguo Huang, Zhaoyang Yang, Songxuan Lai, and Jifeng Sun (2018). "Dense and tight detection of chinese characters in historical documents: Datasets and a recognition guided detector". In: *IEEE Access* 6, pp. 30174–30183. DOI: 10.1109/ACCESS.2018.2840218.

Yang, Li, Ying Li, Jin Wang, and Zhuo Tang (2019). "Post text processing of chinese speech recognition based on bidirectional LSTM networks and CRF". In: *Electronics* 8.11, p. 1248. DOI: 10.3390/electronics8111248.

Yousefi, Mohammad Reza, Mohammad Reza Soheili, Thomas M Breuel, Ehsanollah Kabir, and Didier Stricker (2015). "Binarization-free OCR for historical documents using LSTM networks". In: *2015 13th international conference on document analysis and recognition (ICDAR)*. IEEE, pp. 1121–1125. DOI: 10.1109/ICDAR.2015.7333935.

Yuan, Tai-Ling, Zhe Zhu, Kun Xu, Cheng-Jun Li, and Shi-Min Hu (2018). *Chinese Text in the Wild*. arXiv: 1803.00085.

Zeiler, Matthew D and Rob Fergus (2014). "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer, pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53.

Zhai, Xiaohua, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer (2021). *Scaling Vision Transformers*. arXiv: 2106.04560.

Zhang, Jianshu, Yixing Zhu, Jun Du, and Lirong Dai (2018). "Radical analysis network for zero-shot learning in printed Chinese character recognition". In: *2018 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, pp. 1–6. DOI: 10.1109/ICME.2018.8486456.

Zhengzhang, Shangfang (郑张尚芳) (2003). 上古音系 *(Old Chinese Phonology)*. Shanghai Educational Publishing House. ISBN: 978-7-5320-9244-4.

Zhong, Zhuoyao, Lianwen Jin, and Ziyong Feng (2015a). "Multi-font printed Chinese character recognition using multi-pooling convolutional neural network". In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, pp. 96–100. DOI: 10.1109/ICDAR.2015.7333733.

Zhong, Zhuoyao, Lianwen Jin, and Zecheng Xie (2015b). "High performance offline handwritten chinese character recognition using googlenet and directional feature maps". In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, pp. 846–850. DOI: 10.1109/ICDAR.2015.7333881.

Zhuang, Li, Ta Bao, Xioyan Zhu, Chunheng Wang, and Satoshi Naoi (2004). "A Chinese OCR spelling check approach based on statistical language models". In: *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*. Vol. 5. IEEE, pp. 4727–4732. DOI: 10.1109/ICSMC.2004.1401278.

Zhuang, Li and Xiaoyan Zhu (2005). "An OCR post-processing approach based on multi-knowledge". In: *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, pp. 346–352. DOI: 10.1007/11552413_50. URL: https://www.researchgate.net/profile/Xiaoyan-Zhu-3/publication/221018980.

Online Resources

Reynolds, Anh H. (2019). *Convolutional Neural Networks (CNNs)*. URL: https://anhreynolds.com/blogs/cnn.html (visited on 08/12/2020).

Tsai, Chih-Hao (1996). *Frequency and stroke counts of Chinese characters*. URL: http://technology.chtsai.org/charfreq/ (visited on 08/26/2020).