2018-05-01

Subword Spotting and Its Applications

Brian Lafayette Davis
Brigham Young University

Subword Spotting and Its Applications

Brian Lafayette Davis

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

William Barrett, Chair
Thomas Sederberg
Daniel Zappala

Department of Computer Science

Brigham Young University

ABSTRACT

Subword Spotting and Its Applications

Brian Lafayette Davis
Department of Computer Science, BYU
Master of Science

We propose subword spotting, a generalization of word spotting where the search is for groups of characters within words. We present a method for performing subword spotting based on state-of-the-art word spotting techniques and evaluate its performance at three granularitires (unigrams, bigrams and trigrams) on two datasets.

We demonstrate three applications of subword spotting, though others may exist. The first is assisting human transcribers identify unrecognized characters by locating them in other words. The second is searching for suffixes directly in word images (suffix spotting). And the third is computer assisted transcription (semi-automated transcription). We investigate several variations of computer assisted transcription using subword spotting, but none achieve transcription speeds above manual transcription. We investigate the causes.

Keywords: subword, spotting, CAT, semi-automated, handwriting, n-gram, character

ACKNOWLEDGMENTS

I would like to thank my wife Sarah for her support of, and patience with, my endeavors. Her editing skills are also responsible for eradicating many errors in this document.

I would like to extend thanks to Curtis Wigington, Seth Stewart, Chris Tensmyer and Bill Barrett, who, among other assistance to my work, provided use timing data for my semi-autometed transcription system used in its simulations.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Most data recorded by humans in the past, and much even today, is in handwritten documents. It is highly desirable for many areas of research for this information to be easily accessible, either being searchable through automatic means or having the data digitized in some form that is easily manipulated by computers.

The digitization of these documents is a large task. Obviously not all recorded data is important, but even within a specific domain, the number of relevant handwritten documents can be far too large to be manually transcribed to digital text (that is, a human reads the document and then types the contents into a computer). Within the domain of family history research, for example, there are billions of handwritten documents which have been photographed as digital images and more are being captured every day. The transcription of these lags far behind, and the gap is growing. It is expensive to have the contents of these documents manually typed by people.

A solution to the problem of transcribing documents that has been worked on for several decades is automated handwriting recognition. The basic formulation is that given an image of handwriting (either a character, word, line, paragraph or page), automatically produce the text of its handwritten content. The state-of-the art is primarily focused on the transcription of text lines and paragraphs through artificial neural networks, particularly recursive neural networks (RNN) [25, 26, 35]. These methods work relatively well, particularly in single author scenarios. However, they do require large training sets and would require human correction in some difficult applications where human-level accuracy is required.

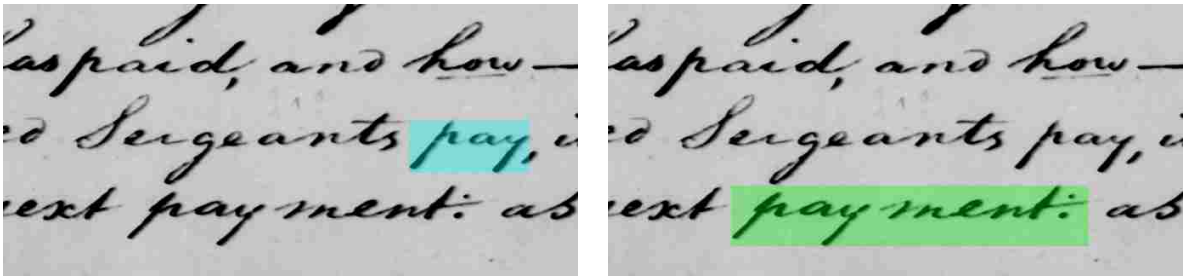


Figure 1.1: Examples of word spotting for ‘pay’ and ‘payment’.

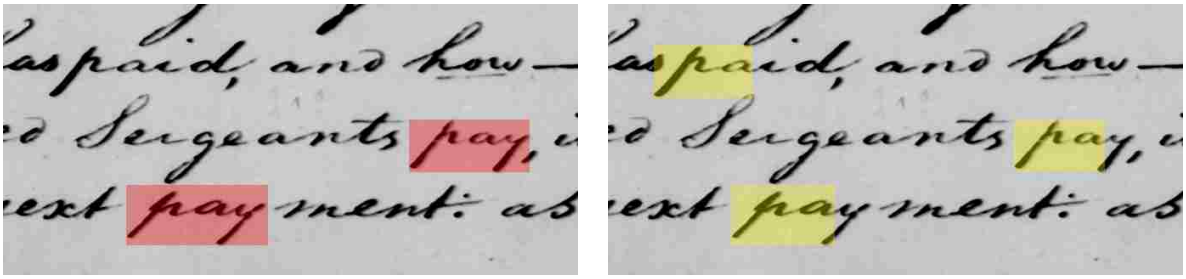


Figure 1.2: Examples of subword spotting for the character trigram ‘pay’ (left, red) and bigram ‘pa’ (right, yellow).

Computer assisted transcription (CAT), or semi-automated transcription, is an approach to transcription which begins with the understanding that human input will be needed to achieve the desired accuracy and aims to merge computed and human effort in an effective manner. These methods have generated less interest recently due to the large gains deep convolutional neural networks have given automatic transcription.

However, transcription is not always needed; if search is the only utility desired an alternative solution is word spotting [14]. In word spotting, the goal is to make a collection of document images searchable without transcription. The search results are based on visual features the system extracts. The result of word spotting is the location, and potentially bounding box, of each instance of the query in the corpus (see Figure 1.1).

In this work, subword spotting is explored. Where word spotting finds words matching a query, subword spotting relaxes this to finding any instances of the query, even within a word. As seen in Figure 1.2 (left, red), an additional instance of “pay” is found compared to Figure 1.1. And in Figure 1.2 (right, yellow), an additional instance of “pa” is found, which turns out to be a different form of the word“pay.”

Figure 1.3: Cropped examples of the characters “e” and “i” (excluding dots), on the left and right respectively, from a single author. Without context the characters are practically indistinguishable.

1.1 Why Subword Spotting?

The motivation for exploring subword spotting comes from the observation that techniques for handwriting recognition have tried many different primitives, or fundamental units, for recognition. One can look at pixels at the lowest level, and then move onto graphemes or substrokes [13], strokes, characters, subwords (or character n-grams), words and sentences. With current neural networks, the primitives are learned within the stacked convolutional layers, and the network then recognizes characters based on these. Previous CAT systems such as [5] and [37] use whole words as units for recognition.

An individual primitive has two desirable properties: distinctiveness and frequency. Without distinctiveness it will be very difficult to discriminate between different primitive instances. Frequency is valuable as it allows us to get more out of the effort required to discriminate.

Subwords provide an interesting middle ground between individual characters and complete words. Individual characters can be very similar to one another (i.e. not distinctive). For example, in cursive an “i” missing it’s dot frequently looks like an “e”, as seen in Figure 1.3, making discriminating between characters in isolation very difficult. On the other hand, words are generally much more distinctive than characters. However, while individual characters occur many times throughout a corpus, a given word may appear less frequently.

In the extreme, words such as names, may occur only once. However, subwords, such as "th", "ed", "ion", "ing", etc., occur very frequently (in English) as they are used in many words. And while being more frequent than words, subwords also provide more distinctiveness and discernibility than individual characters.

Important use cases for subword spotting include searches (1) where a root is desired to be searched, such as querying "pay" wanting to find instances of "payment", "payments", "prepay", etc., (2) where only part of the desired word, such as a name, is known, such as "Chr.", (3) where there are character(s) which a human transcriber cannot initially recognize, but if instances elsewhere in the document in familiar words could be found in context, would become discernible to the transcriber.

An additional appeal of subword spotting lies in computer assisted transcription (CAT). In character-level recognition schemes, a lexicon is frequently used to correct erroneous character predictions if other characters in the word are transcribed correctly. Similarly a lexicon can also be used to fill in untranscribed characters, assuming significant other characters are transcribed. This is useful in a transcription by iteratively spotting subwords. While the spotting results may not cover every character, they can cover a sufficient number of characters for each particular word.

In Figure 1.4 we see an example of the word "payment", where "pa" and "men" have been spotted; this covers 71% of the characters in the word. However, if we estimate that there are 1-2 characters between "pa" and "men" and estimate 1-2 characters after "men" the regular expression `pa..?men..?` matches only "pavement", "pavements", "payment", and "payments" in our large lexicon (described in Chapter 3). This is a particularly potent tool with longer words, which typically may require more effort to automatically transcribe, but tend to be more distinctive from a lexical perspective.

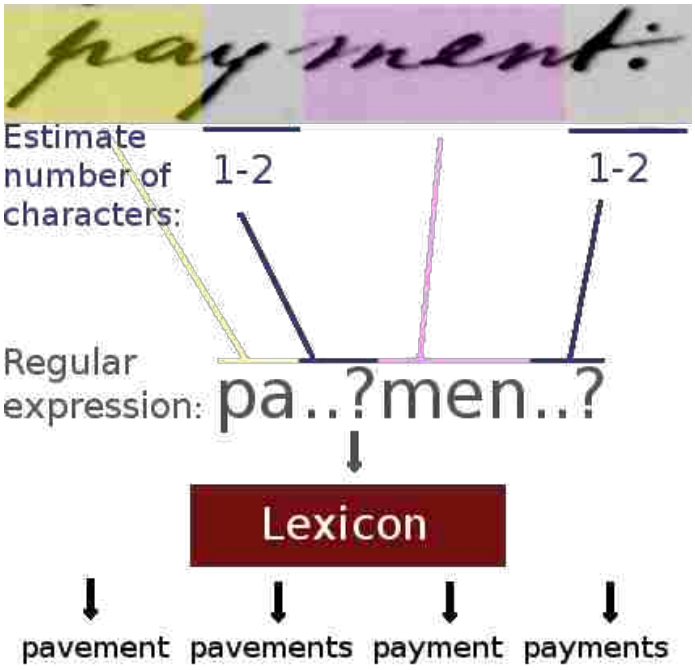


Figure 1.4: An example of a word having "pa" and "men" spotted in it. A regular expression representing this, `pa..?men..?`, yields only a few matches from our lexicon, including the correct one: "pavement", "pavements", "**payment**", and "payments."

1.2 Main Contributions

The primary scope of this work is to provide an exploration of the performance of subword spotting, extending state-of-the-art word spotting methods. We show that reasonable levels of mean average precision (mAP) can be achieved for character tri-, bi-, and unigram spotting. We demonstrate the variability in performance between different character n-grams (referred to simply as n-grams hereafter).

Additionally, we demonstrate several applications of subword spotting: user directed searching to aid in human transcription, suffix searching/spotting, and CAT using word completion from aggregated subword spottings.

Chapter 2

Related Work

2.1 Word Spotting

Word spotting was first proposed as an alternative to transcribing a corpus. Rather than transcribing the document image so standard text searches can be run, the document is searched using the images themselves, either with a keyword string or a keyword image (exemplar image). In the past, techniques were distinguished by which search pattern they used. There are two primary approaches to featurizing the images when word spotting is performed: holistic features, that capture information about a whole image (word), and local or sequential features [22]. Holistic features have one description for an entire word image (such as a bag-of-visual-words [29]), whereas local features have a descriptor for a small portion, or window, of a word image.

Most early work with local features share the common theme of taking features from vertical slice windows (usually only one or a handful of pixels wide); Figure 2.1 shows an example of this process. They compare these to the features extracted from an exemplar using either dynamic time warping or hidden Markov models (HMMs), relying on a prior line segmentation. The variation between the methods largely lies in the features used. Some use small square windows to allow segmentation-free word spotting using a sliding window [24].

Rodrıguez et al. [22] proposed a simple histogram gradient feature that demonstrated improved performance when compared with (1) the popular profile and transition-based features of Rath and Manmantha [19], (2) the gradient and transition-focused features of Marti and Bunke [16] and (3) the simple histogram features of Bunke et al. [4]. They also

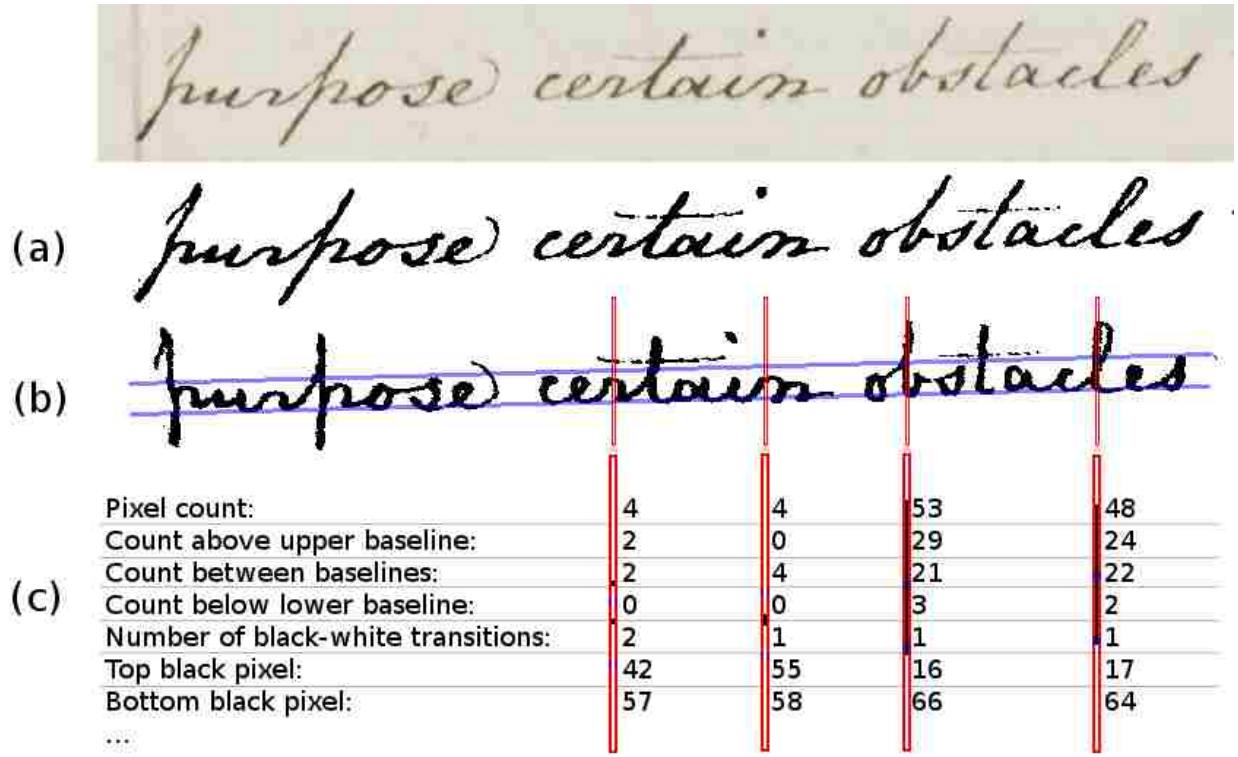


Figure 2.1: Example of how vertical slice window features are extracted. Typically most features are extracted on a binarized image (a). Deskewing the image (b) plays an important role as the vertical slices are very sensitive to skew. Many typical features extracted (c) are pixel counts; in this example we show count features dependent on baselines (blue).

showed that a HMM worked better than dynamic time warping, for their features. Their method was not an exemplar-based approach.

Aldavert et al. [1] and Almazan et al. [3] have presented superior word spotting methods that rely on heuristic descriptions. Aldavert et al. used the well known bag-of-visual-words method, including recent improvements from the computer vision community. This is a very simple, yet effective, exemplar spotting approach. Aldavert et al. use Fischer vectors, which are similar to bag-of-visual-words, with spatial pyramids in addition to a special pyramidal histogram of characters (PHOC). PHOC vectors are a fixed length vector which describe a word; it encodes which characters are present and roughly their location. If your alphabet is length N, the first N values of the vector indicate if each character is present in the word (Is 'a' in the word? Is 'b' in the word?). The next N values indicate

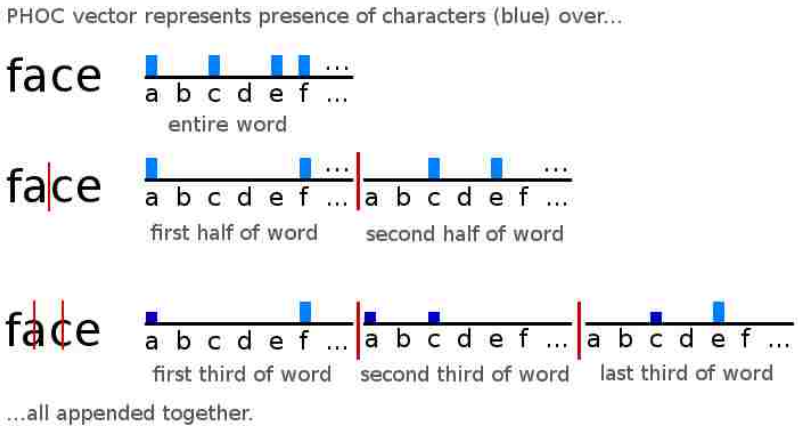


Figure 2.2: Example of a three-level PHOC vector for the word "face." The final vector is all levels appended together. Note that partial values are given when characters are split over bins.

if each character is present in the first half of the word, the N values after that indicating presence in the second half of the word (this is the rough location encoding). This is carried out to three levels. In the original PHOC a set of character bigrams are also used on the first two levels, however, we do not use these. See Figure 2.2 for clarification. The PHOC vector and pyramidal bag-of-visual-words are used to find a new space in training in which both strings and word images can be embedded. This allows it to perform both string and exemplar queries as well as hybrid queries which yield excellent results.

Sudholt and Fink [30, 31] expanded on [3] by training a deep convolutional neural network to produce the PHOC vector from a word image. This, and following improvements [12, 21], have yielded the state-of-the-art for segmentation-based spotting. We base our spotting method on [31].

There are also techniques for segmentation-free word spotting, where spotting is performed on a full page without any segmentation information [36]. While we used word image segmentation in our work to narrow its scope, we note that it could be applied, with some adjustments, in a segmentation-free setting.

In this thesis we are interested in spotting character n-grams (uni-, bi- and trigrams) rather than words. While this hasn't explicitly been done, we feel examining the performance

of other methods on short words (2 to 3 letters) is instructive. For Rothacker et al.'s [24] segmentation-free HMM based method, they report ∼46% and ∼55% mean Average Precision (mAP) for two- and three-lettered words respectively, a drop by ∼15% and ∼6% from the mAP for all words they tested (61%). Fischer et al. [6] report ∼70% and ∼83% mAP for two and three lettered words, compared to >90% mAP for words of length 5 or longer, for their line segmentation dependent, character HMM based method. While neither of these numbers are very promising, Almazan et al. [2] observe that sliding window approaches can frequently find false positives of short words inside other words (e.g. finding the word "the" inside the word "weather"). As this is precisely what we want to have happen in n-gram spotting (that is, we *want* to find groups of letters in the middle of words), we expect we should have more success spotting character n-grams than other methods in spotting short words, assuming we have good sliding windows. However, part of the poor accuracy in spotting short words is simply the fact that there is less information with which to discriminate.

2.2 Automatic Handwriting Recognition

The state-of-the-art for handwriting recognition relies heavily on recursive neural networks (RNNs) paired with convolutional neural networks (CNNs). The key component is the connectionist temporal classification (CTC) loss which allows the network to be trained given a line image and the ground truth text for that line without any alignment [8]. In a recent competition on historical German documents [26] (see Figure 2.3), the leading contestants

Figure 2.3: Example of lines extracted from the dataset of the ICDAR HTR competition [26].

used a bidirectional RNN on top of a CNN. The best results had word error rates of 19.1% and character error rates of 7.0%. This is impressively low given the diversity of the data, and deep learning methods are continuing to improve [18, 35].

2.3 Computer Assisted Transcription

There tends to be two primary CAT (Computer Assisted Transcription) methodologies: corrective and directive. Corrective CAT methods rely on handwriting recognition to do most of the work, and then users provide feedback correcting some errors of the recognition, where the gain comes from other errors being automatically corrected or being assisted so that corrections are easier to make.

In a directive CAT method, the user provides some of the initial input to the system and it uses this to transcribe more than what the user inputted. An active learning approach to handwriting transcription is similar to CAT, but is not concerned about achieving human level accuracy, just better accuracy, and thus requires less human involvement [27]. We will focus on CAT approaches.

Toselli et al. [33] have explored the realm of corrective CAT using the idea of user-verified prefixes. They use a fairly standard HMM recognition model as the backbone of their approach, and take advantage of the incremental nature of the Viterbi decoding algorithm. The recognition is done for a line of text and the user corrects the first error. The Viterbi decoding is then run again, but this time using the assumption that everything occurring before the correction is correct, and thus reusing the computation up to that point. They have also explored slight variations of the same approach that enable more fluent user input with a touchpad [34], mouse [23], or multimodal means [32], which speed up the transcription process by allowing more intuitive user interaction (Fig. 2.4). Their approach relies on a language model to make corrections on a line when a supervision is made. This means this method cannot be used to effectively transcribe documents containing non-sentence writing,

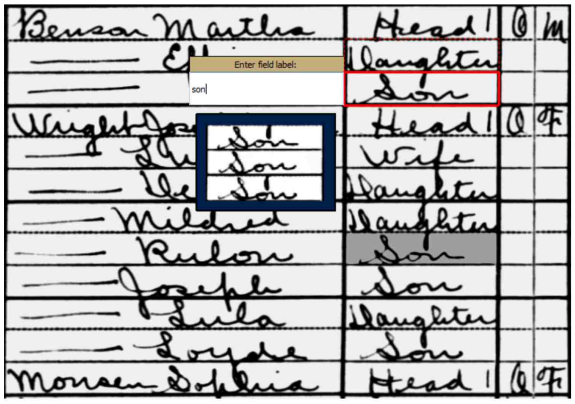
Figure 2.4: A screenshot of a demo of Toselli et al's multimodal CAT system. The red line is drawn by the user to indicate the need to insert a word into the automatically obtained transcription.

such as tables and lists. Serrano, et al. also have pursued a similar approach, where the user corrects the n words in which the recognition model had the least confidence [28].

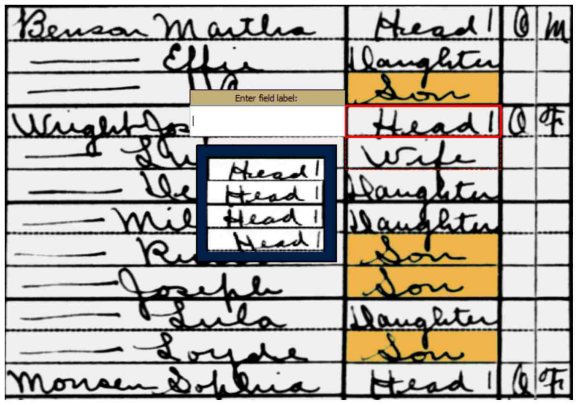
While it is useful to employ an automatic handwriting recognition method as part of a CAT approach to reduce the human burden, we cannot use the same type of recognition used by these previous CAT systems, where there is a reliance on the predictive capacity of a language model. Some documents are not comprised of only sentences, and thus cannot be transcribed by these methods (e.g. the name field of census documents, see Figure 2.5). However, many documents are structured such that a pattern can be learned to assist in transcription.

Robert Clawson [5][1] designed a CAT system for handwritten tabular documents, which have a clear pattern. His approach relies on simply finding matches in the document column of the current word image (red box in Figure 2.5(a). This is essentially query by example word spotting with user oversight. The matched words are assigned the same user-specified label. This provides an accurate CAT system where the user oversees all transcription. The user oversight of matches is accomplished by showing a list of matches to the user (with an adjustable threshold for sensitivity) from which the user removes the false-positive matches. The remaining all have the same label applied to them. This interface, discarding bad

[1]You can view a short demo and explanation of his approach at `https://www.youtube.com/watch?v=gqdVzEPnBEw`



(a) The red box indicates the current word. A small window shows the matching words that appear elsewhere in the column. The user can get rid of bad matches by either clicking on them or adjusting a threshold.



(b) The matched words, highlighted in yellow, are all assigned the same user-entered label.

Figure 2.5: Clawson's CAT system for tabular documents.

matches, leverages the human user's natural ability to quickly identify outliers. Our CAT attempts to also leverage this ability as well.

Zagoris et al. [37] [2] presented a CAT system that was also based on word spotting. In their system when the user is transcribing a word image, they are presented with the results of spotting that image (sorted according to rank). The user can then confirm these spottings by clicking on them, causing them to move to a separate list. When this is done, a relevance feedback loop is activated, which submits another word spotting query of the confirmed image. These spotting results are used to refine the ranked list, providing a better selection. Figure 2.6 shows a screen-shot of their demo system. We use the same strategy as [37] to guide the combination of our subword spotting results from multiple queries. Further details are given in Section 4.1.5.

Neudecker and Tzadok [17] presented a CAT system for historical printed documents that is very similar to the CAT system we present in Chapter 6. The system would first segment the individual characters of the documents and run an OCR engine on them. Those characters with low confidence would then be presented to a user for verification in a character

[2] A demo of their system is found at `http://vc.ee.duth.gr/ws/`

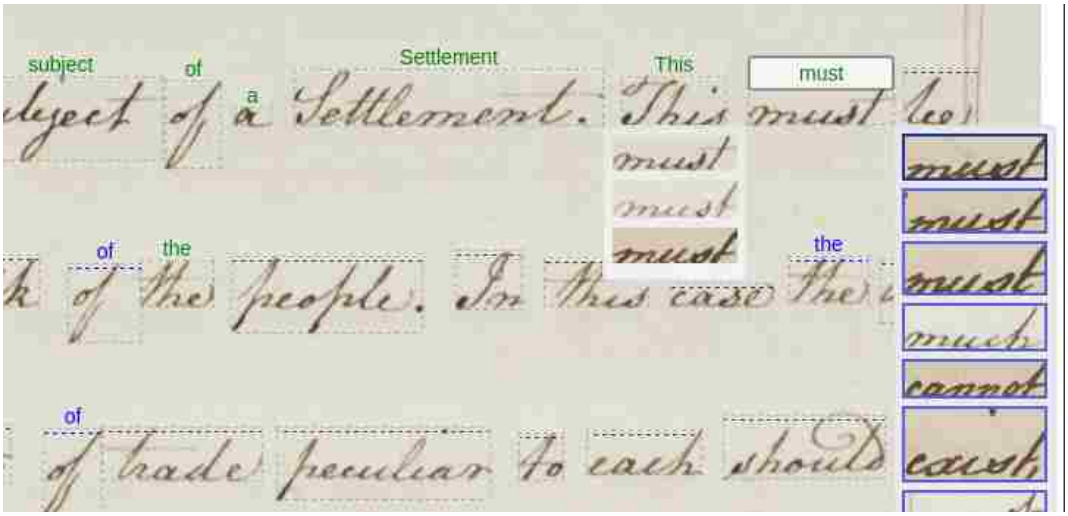


Figure 2.6: A screen-shot of a the CAT system of Zagoris et al. [37]. The green text indicates hand labeling, the blue text indicates automatic labeling. You can see to the right of the current word ("must") the current ranked list of spottings (blue boxes).

session. A single character session contains all the low-confidence character images classified to a single character. An example of their system's character session for the character "?" is given in Fig. 2.7. The user merely needs to select the incorrect classifications. Then in a word session, a word image is shown to the user with possible transcriptions for the word, from which they select the correct one. There are three key strengths of this system. One is that as long as the documents characters can be segmented, it can work. The second is that it formats all user tasks as selections, rather than typing, and they are quickly completed. This creates a much more enjoyable experience for the user. The third key strength is that it is highly parallelizable for crowd-sourced transcribing. This parallelism is achieved because all character sessions are independent of one another and all word sessions are independent of one another. Our system follows this method's pattern so it has flexibility of document types, simple user tasks and a parallelizable framework.

Retsinas et al. [20] expanded on [17] to reduce the amount of user input required by clustering character images together. By viewing an average of a character cluster (where the characters are overlapped), the user can then assign the whole cluster a character label or reject the cluster as being incoherent (i.e. the cluster contains multiple characters). See

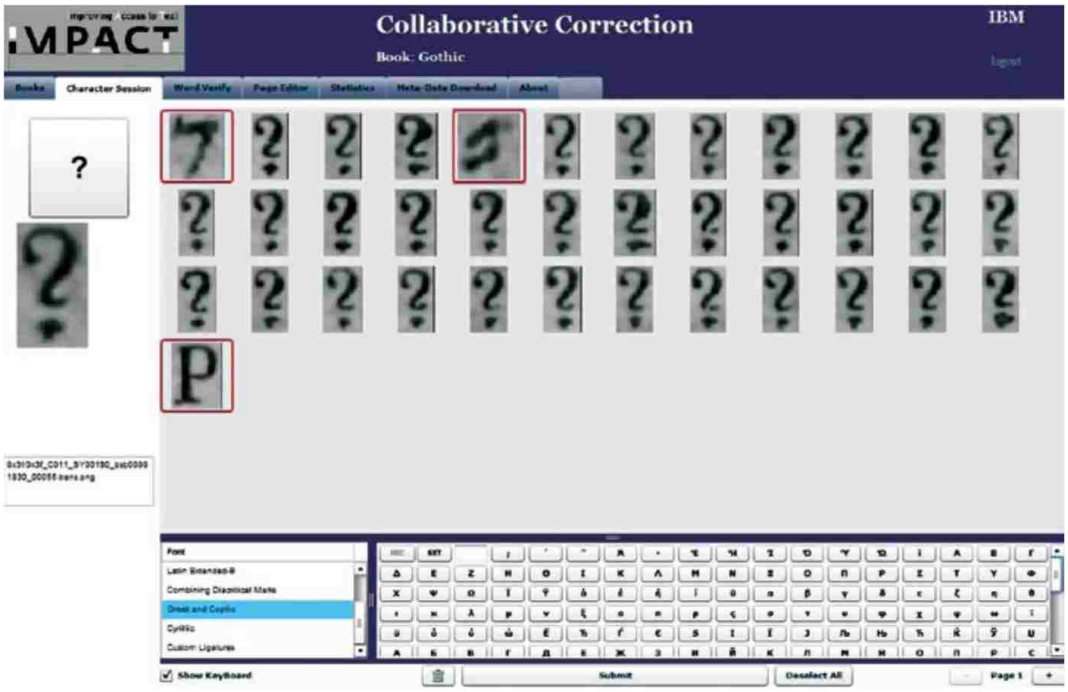


Figure 2.7: A screen-shot of a character session for "?" from Neudecker and Tzadok's CAT system [17], taken directly from their report [17]. Notice how easy it is for a user to simply click on the erroneous classifications.

Figure 2.8 for an example. Though requiring more thought from the user, this prevents the tediousness of examining all character images. We attempt to use clustering in our own CAT method, however we cannot perform the overlap due to the greater variance in handwriting. Instead we take the approach of [5] and simply group clusters together to help visual discrimination.

While we have not seen it done, it would be relatively easy to transfer the results of RNN/CTC-based automatic transcription method to CAT by extracting the top-N transcrip-

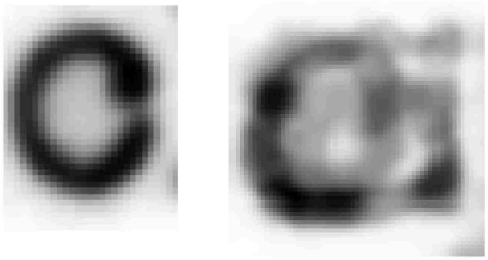


Figure 2.8: Examples of a good cluster ('c') and an incoherent cluster (multiple character classes) taken from [20].

tions from the network output. These could then be fed to a user for approval and quick correction for most errors. Given the strength of these automatic methods by themselves, we expect a CAT version of them would outperform the older CAT methods discussed in this section.

Chapter 3

Datasets

We use three datasets in our evaluations: the IAM handwriting dataset, the Bentham dataset, and a dataset we extracted from the U.S. 1930 census, which we refer to as Census Names. In this chapter we describe each of these datasets and additionally describe our lexicon.

3.1 IAM Dataset

The IAM handwriting dataset [15] is a dataset collected by having human subjects copy certain paragraphs of printed text into handwriting. They were instructed to keep lines well separated and the dataset is very clean as a result. It contains annotations to the word level. We used the provided test set split, the combine training and validation 2 splits as our training set, and the validation 1 set as our validation set. Example lines from the IAM dataset can be seen in Figure 3.1. The test set has 96.1% of its contents in our lexicon. These sets have the respective sizes and number of (exclusive) authors:

Training: 55106 words, 326 authors

Validation: 7089 words, 46 authors

Testing: 14600 words, 128 authors

3.2 Bentham Dataset

The Bentham dataset [7] is a collection of documents written by the 17th century philosopher Jeremy Bentham. While our data is based on the officially released ground truth labels and

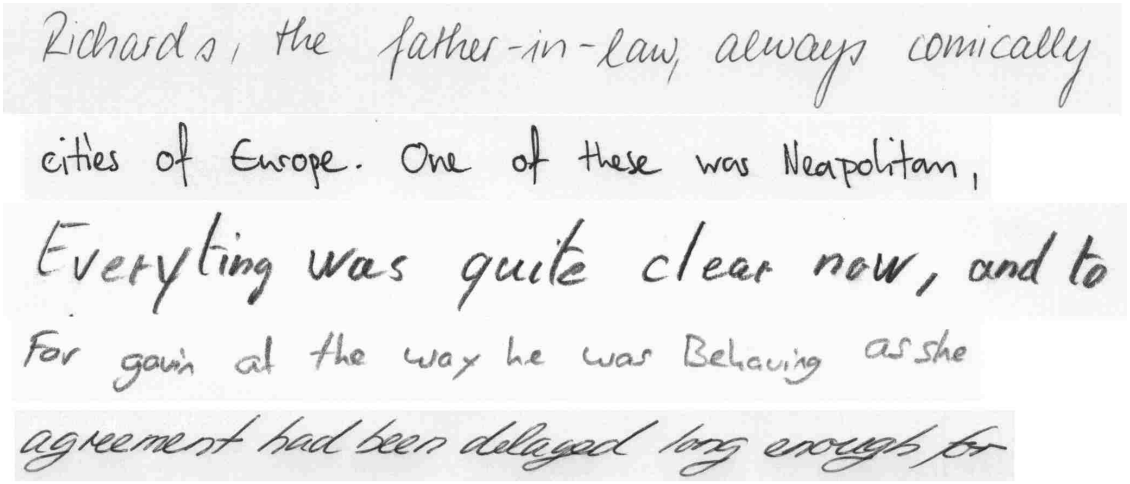


Figure 3.1: Examples of lines from the IAM dataset.

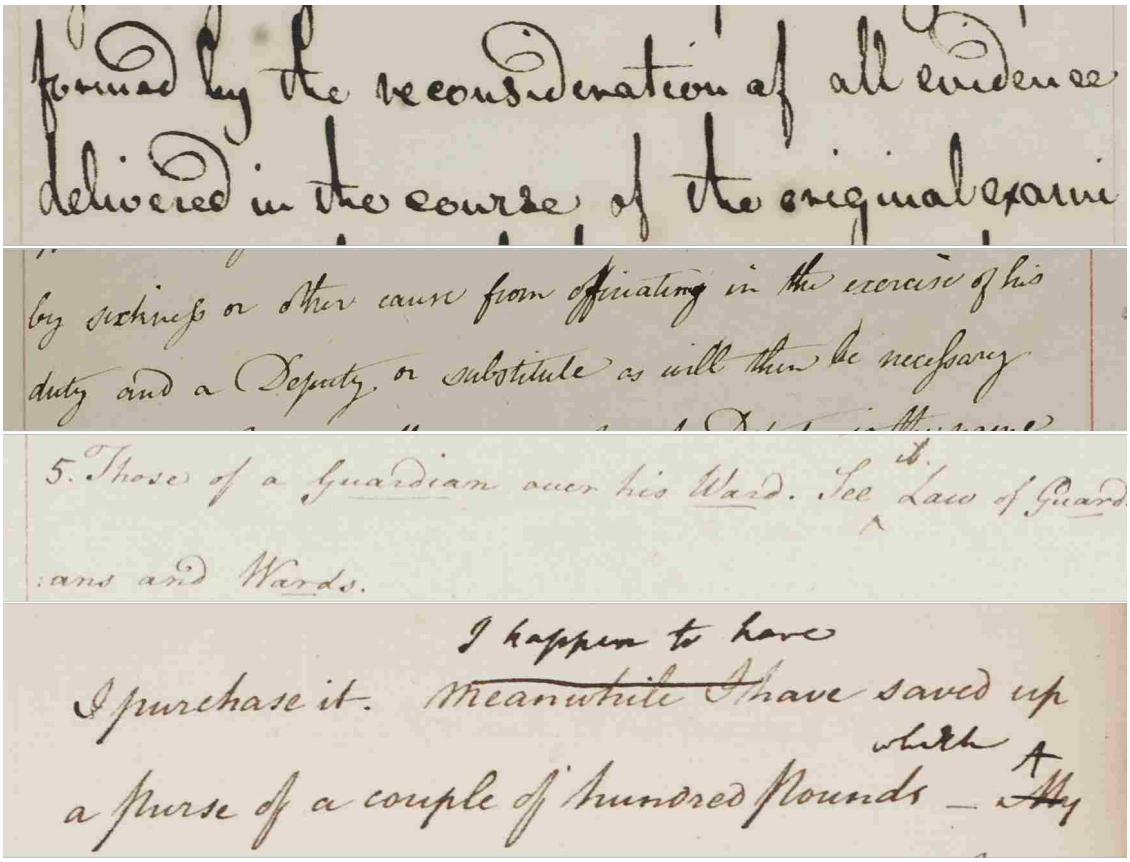


Figure 3.2: Excerpts from the Bentham dataset.

bounding boxes, we have corrected a plethora of errors. Our training, validation, and test sets are based on the official split, but are not identical. We ensured pages are exclusive to a single set. Example lines from the Bentham dataset can be seen in Figure 3.2. The test set has 94.9% of its contents in our lexicon These sets have the respective sizes:

Training: 8490 words

Validation: 1071 words

Testing: 860 words

3.3 Census Names Dataset

The Census Names dataset is an extraction of names from the United Stated of America 1930 census. Example lines from the Census Names dataset can be seen in Figure 3.4. FamilySearch provided us the ground truth index as well as a registration of the form images (rotation, scale and offset to align them). We locate the bounding boxes of the name fields by the following process:

1. Average the registered images (Figure 3.3 a).
2. Manually enhance the contrast of the resulting image (Figure 3.3 b).
3. Hand annotate the average image with the lines of the form, using straight lines (Figure 3.3 c).
4. For each form image perform a more refined registration
   (a) Globally move all the form lines together to maximize the summed inverse pixel intensity (darker is better) along the form lines. This is done by scanning 50 pixels in the four cardinal directions separately (Figure 3.3 d).
   (b) Do a more dense scan of all positions in a 7×7 neighborhood around the combined the best x and y positions from the separate horizontal and vertical scans (previous step).

(c) For each individual cell of the form, create a small (12 pixel) vertical or horizontal profile around each of the four lines forming the cell boundaries (Figure 3.3 e)

(d) Locally snap each cell wall to the darkest part of it's respective profile (Figure 3.3 f). This yields a very precise registration in most cases.

(e) Manually segment the words (last name, first name, middle initial/name) within each cell (Figure 3.3 g).

The test set has 83.7% of its contents in our lexicon These sets have the respective sizes:

Training: 6292 words

Validation: 718 words

Testing: 2130 words

3.4 Character Annotation

For the testing portion and a small validation set of the Bentham and Census Names datasets, we produced a character segmentation ground truth. We annotated the word segmentation ground truth by marking the point between characters, meaning character boundaries do not overlap, in addition to a tighter start and end boundary for the word (first and last letters), as seen in Figure 3.5.

3.5 Lexicon

For our CAT (computer assisted transcription) methods, we require a lexicon. We choose to use a large English lexicon as this would indicate the methods could generalize to other corpi. We obtained a lexicon from SIL International at `http://www-01.sil.org/linguistics/wordlists/english/` which contained 108,028 words (not including names). We also desired a large lexicon of names which would enable the Census Names dataset to be transcribed. FamilySearch provided us with data from the US 1940 census, which listed all names appearing in those records along with a count of how many times they occurred. We took all names

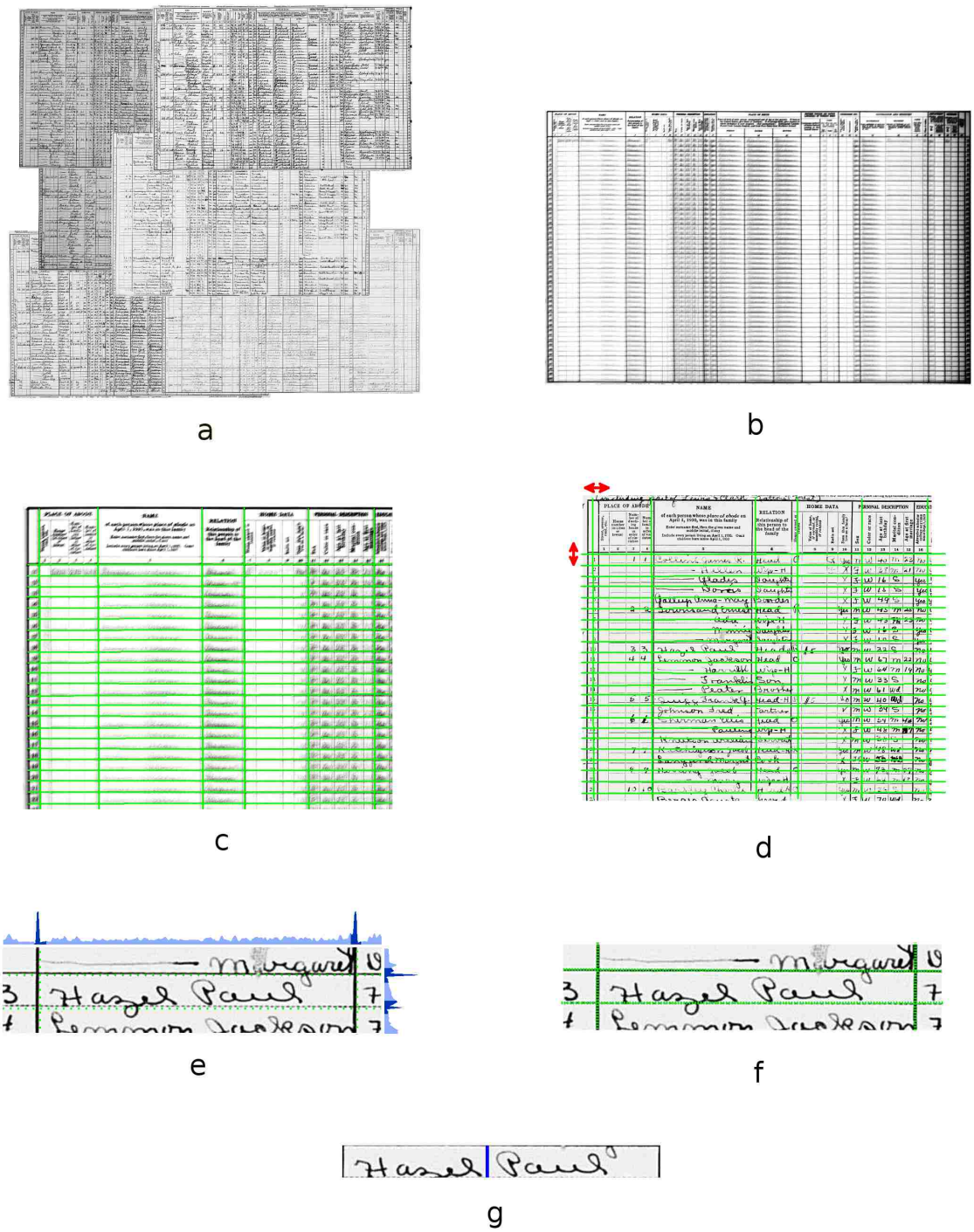


Figure 3.3: Process for extracting names from US 1930 census forms for Census Names data set. We begin with registered forms (a). Then we average the images together (b). The average image is used to mark form boundaries manually (c). The lines are registered to specific census image (d). At each cell a projection profile is computed around the cell boundaries (dark blue) (e). The cell boundaries are snapped to the histogram peaks (f). Word boundaries were manually annotated for each cell (g).

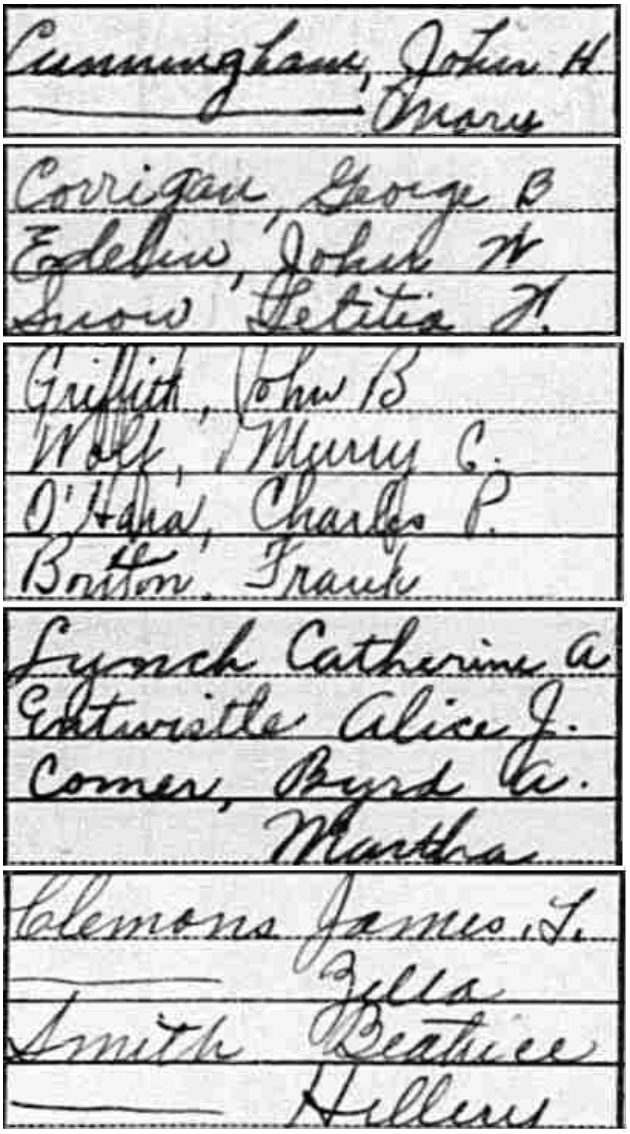


Figure 3.4: Excerpts from the Census Names dataset.

Figure 3.5: Example of hand annotated character segmentation.

occuring at least 1000 times, leaving us with a list of 6,939 unique names. Combining these two sets gave us a lexicon size of 114,968. We use the full lexicon for the Bentham dataset, but only the names portion for the Census Names dataset.

Chapter 4

Subword Spotting

Subword spotting is an extension of traditional word spotting where we allow instances to occur within words. We attempt to localize the spotting in the word to provide spatial information. In this work we focus on small subwords (1-3 characters), as most of the applications of subword spotting we explore in Chapters 5 and 6 use these. Figure 1.2 shows an example subword spotting.

In this chapter we describe our implementation of subword spotting and then evaluate many aspects of its performance, including full word, uni-, bi-, and trigram spotting as well as multi-query aggregation.

4.1 Implementation

We use a sliding window over word images with a word spotting CNN to perform subword spotting.

4.1.1 Architecture

Our subword spotting is built on the segmentation-based word spotting method PHOCNet [30, 31] which we adapted to perform a sliding window over the word images. [31] uses a deep convolutional network trained on word images with pyramidal histogram of characters (PHOC) [3] as the target vectors. This method can word spot using both query-by-string (QbS) and query-by-example (QbE), that is a query may be a text word or a word image. Searches are performed simply by comparing vector similarities, the query vector either being

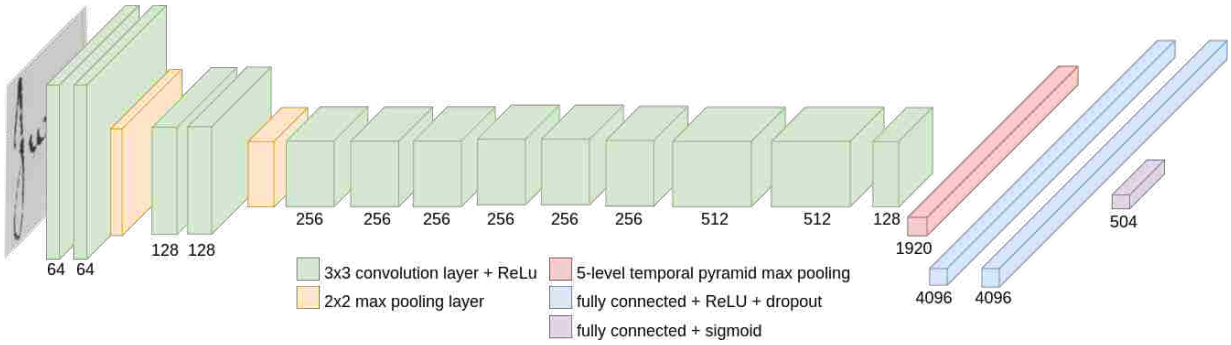


Figure 4.1: Network architecture for embedding images as PHOC vectors. The numbers beneath each layer represent the number of channels. As the network uses temporal pyramid pooling before the fully connected layers, it can accept images of any size. Our architecture differs from [31] only in the number of channels of the last convolutional layer.

a word's PHOC vector if searching with text or found from running its image through the network. See Section 2.1 for the details of PHOC vectors. We follow [31] and use a PHOC vector where the alphabet is case insensitive, includes digits (0-9), and does not include bigrams (unlike [3] which uses bigrams in the first two layers). For our specific application of subword spotting, we try adapting the PHOC vector in [31] so instead of having levels 2,3,4,5 we have 1,2,3. [31] was designed for words which require more spatial resolution than subwords. Our PHOC levels directly correspond with the fact that we are spotting uni-, bi- and trigrams. We refer to this as our "adapted PHOC" in the results.

Our network architecture varies slightly from [31] and is outlined in Figure 4.1. We note the reduction in the number of channels in the layer before the temporal pyramid pooling (TPP) and fully connected layers (from 512 to 128); this is to reduce the size of featurized images (which are saved after this layer). We used Sudholts's code available on GitHub (`https://github.com/ssudholt/phocnet`), which uses the Caffe framework [10].

TPP is a slight modification of spatial pyramid pooling (SPP) [9], which is a method of allowing a network with fully connected layers to handle images of arbitrary size. This is accomplished by performing a pooling (max in our implementation) over windows of the feature map (image). The first level is a window that encompasses the full image. The second level is four windows dividing the image into fourths (each the size of half the image width

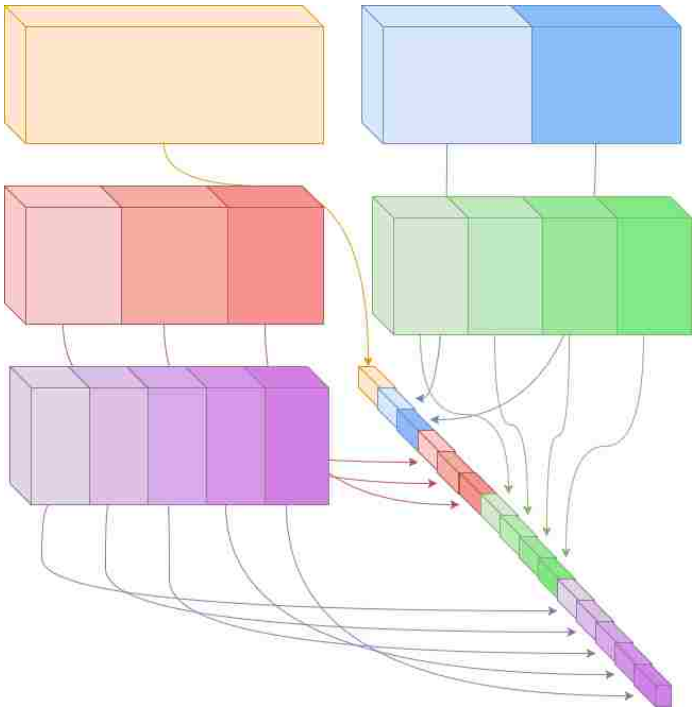


Figure 4.2: What temporal pyramid pooling (TPP) looks like visually. The same network features (large blocks) is divided into even horizontal windows of different counts (1,2,3,4,5 here). The result of pooling each of these windows (max pooling in our implementation) is appended together as an output vector. This is the red vector in Figure 4.1.

and half the image height). The proceeding levels follow a similar pattern. TPP uses the same idea, but only divides windows horizontally as vertical information is not useful once word images are featurized. We use 5 layers (dividing the image into 1, 2, 3, 4, and 5 even windows). Figure 4.2 shows what TPP looks like visually. We modify TPP slightly to handle the corner case of small images such that the windows are adjusted to be slightly overlapping at a particular level if a window had a width of zero at that level.

4.1.2 Training

Training follows a procedure based on [31] where word images are passed through to the network and the loss is computed against the ground truth PHOC using cross-entropy. We used a batch size of 10 words, a learning rate of 10^{-4}, a weight decay of 0.00005, and the Adam optimizer [11]. We train the model out 240,000 iterations, dropping the learning rate by a factor of 10 for the last 10,000 iterations. We note that the model does not appear to overfit, but improvements at this point in training are minimal.

We use the same data augmentation as [31], which consists of balancing the training set according to word occurances (by their text) and distorting duplicate instances with random affine transformations.

4.1.3 Determining Window Widths

Because the neural network takes in some context, it is difficult to measure what the optimal sliding window size should be, based on visual width. Additionally each n-gram is likely to have a different optimal window size. We determine the optimal window size for each n-gram as follows. We first compute the mAP for a range of window sizes (stepping 4 pixels), smoothing these results and taking the max. We smooth by averaging a window of 3 steps (average mAP from 3 window sizes), using 2 steps on the edges. For efficiency we cluster all the resulting window sizes (from each n-gram) into 10 clusters using k-means, assigning each

n-gram the cluster mean as its width. This allows us to pre-compute PHOC vectors of only 10 window sizes.

We note that refining the window size during spotting to maximize the score for a specific word image does not give improvements overall.

Our optimal sizes are reported in the Appendix in Table 9.1.

4.1.4 Running

To prepare for spotting, we save the output of the network before the TPP layer as a featurization of each word image. Then subwindows of this featurization can be passed to the latter part of the network (starting with TPP) to generate a PHOC vector. As part of the preprocessing we extract PHOC vectors for subwindows with a stride of 4 pixels (step of 1 in the featurized image as the network reduces images by a factor of 4) for the 10 sizes determined using the method in the previous section.

[31] uses a cosine similarity to compare PHOC vectors (both string and network generated). We used this, but also tested other comparison methods and found that for string generated PHOC vectors, using cross-entropy performs better. This is what the network is optimizing, and as the training labels are string generated PHOC vectors it follows that this should perform well. The query vectors for network generated vectors (QbE queries) are noisy and appear very different. We also found improvements by masking out irrelevant levels of the PHOC for the given n-gram being spotted. For unigram spotting we only use level 1, for bigrams levels 1,2 and for trigrams levels 1,3. This masked-cross-entropy comparison is the measure we use for QbS and we use cosine similarity for QbE.

4.1.5 Combining QbE Results

In one of our tests, we perform aggregation of QbE results by merging results from different queries. To do this we check for potentially overlapping spottings between two sets of results, where overlap is defined as when the predicted bounding boxes overlap by at least 20%. If

Method	IAM		Bentham		Census Names	
	QbS	QbE	QbS	QbE	QbS	QbE
Our network	0.892	0.833	0.926	0.971	0.758	0.858
Our network adapted PHOC	0.868	0.809	0.935	0.975	0.755	0.864
PHOCNet[30]	0.830	0.725	-	-	-	-
TPP-PHOCNet[31]	0.934	0.827	-	-	-	-

Table 4.1: mAP for full word spotting results, reported for both query-by-string (QbS) and query-by-example (QbE).

there is an overlap we discard the result with the worse score. This can trivially be repeated for multiple queries as we are essentially finding the max score. Merging by taking the better score follows what [37] found to be most effective when combining full word spotting results. As further justification, one can think of each n-gram or word as having several prototypical ways it can appear (e.g. an 'a' appearing with or without the tail on top, '$\mathtt{a}$' or '$\mathtt{\alpha}$'). By selecting the best score you select the score computed on the exemplar image with the prototype closest to the instance of interest's prototype.

4.2 Analysis of Subword Spotting

In this section we present results of testing the subword spotting method described above. As a baseline, we show our network's performance on full word spotting. We evaluate subword spotting of unigrams, bigrams and trigrams against our testing portions of the Bentham dataset and Census Names dataset.

4.2.1 Full Word Spotting

Full word spotting is performed as described in [30], where the neural network is used to generate a PHOC vector of each word image in the corpus. For QbS, each word (string) in the corpus's PHOC representation is used as a query. For QbE, each image which has at least one of image with the same label is used as a query. We compare using mean average precision (mAP). Our results are shown in Table 4.1 for the Bentham dataset, the Census Names dataset, and, for comparison with PHOCNet [30, 31], the IAM dataset.

It can be seen that we underperform compared to [31]. This is to be expected as we have reduced the number of parameters in the CNN model. The adapted PHOC vector performs variably when compared to the original, performing better in some cases. This is surprising as we have reduced its descriptive power for words longer than three characters. It may be that the reduced description was easier for our reduced network to learn.

As a note, it is not always meaningful to compare QbS scores to QbE scores, as QbS represents a mean over text queries, one for each possible word label, and QbE represents a mean over instance queries, one for each instance of a word in the dataset, meaning it is biased towards the performance of common words (or n-grams in later tests).

4.2.2 Subword Spotting

We use our sliding window method to evaluate spotting uni-, bi- and trigrams. We used our ground truth for the testing portion of the Bentham and Census Names datasets which have character boundaries. We considered a spotting to be correct if its window overlapped with the desired n-gram's boundaries by 0.5 of the smallest of the two's boundaries. We spotted each letter of the alphabet, the 100 most frequent bigrams (in the English language), and the 300 most frequent trigrams. Each of these n-grams were spotted using QbS if there was at least one instance of it in the testing set. All examples of these n-grams in the dataset were used as a QbE query if there were at least two instances of the n-gram in the test set.

Owing to the fact that a single word may have multiple instances of an n-gram, we align the spotting results to their nearest instance. If multiple spottings claim the same instance and are both overlapping enough to be considered a positive match, we only keep the one overlapping the most. If an n-gram instance is not claimed by any spotting, we add a pseudo-result with the maximum spotting score to the mAP calculations to ensure it properly reflects the missed n-gram.

Results can be seen in Table 4.2. Some qualitative results are show in Figures 4.3 and 4.4 for the Bentham and Census Names datasets respectively.

	Bentham		Census Names	
	QbS	QbE	QbS	QbE
Original PHOC using cosine similarity				
Unigram	0.654	0.511	0.480	0.340
Bigram	0.549	0.569	0.375	0.294
Trigram	0.522	0.571	0.271	0.285
Adapted PHOC using cross entropy				
Unigram	0.677	0.327	0.497	0.203
Bigram	0.682	0.426	0.402	0.179
Trigram	0.705	0.441	0.363	0.170

Table 4.2: mAP for subword spotting results on the Bentham and Census Names datasets, reported for query-by-string (QbS), and query-by-example (QbE).

	PHOC [31]			Adapted PHOC			
	Cosine	CE	Masked CE	Cosine	CE	Masked cosine	Masked CE
QbS							
Unigram	0.654	0.608	0.654	**0.678**	**0.677**	0.499	0.570
Bigram	0.549	0.641	0.679	0.643	**0.682**	0.571	0.663
Trigram	0.522	0.560	0.612	0.618	**0.705**	0.586	0.697
QbE							
Unigram	**0.511**	0.338	0.321	0.547	0.329	0.296	0.717
Bigram	**0.569**	0.429	0.425	0.465	0.426	0.435	0.605
Trigram	**0.571**	0.449	0.431	0.467	0.441	0.497	0.563

Table 4.3: mAP for subword spotting results on the Bentham dataset using cosine similarity and cross-entropy (CE) similarity and networks trained with the PHOC used in [31] and an adapted PHOC for subword spotting. In some experiments parts of the PHOC vectors were masked which were uninformative to the given task.

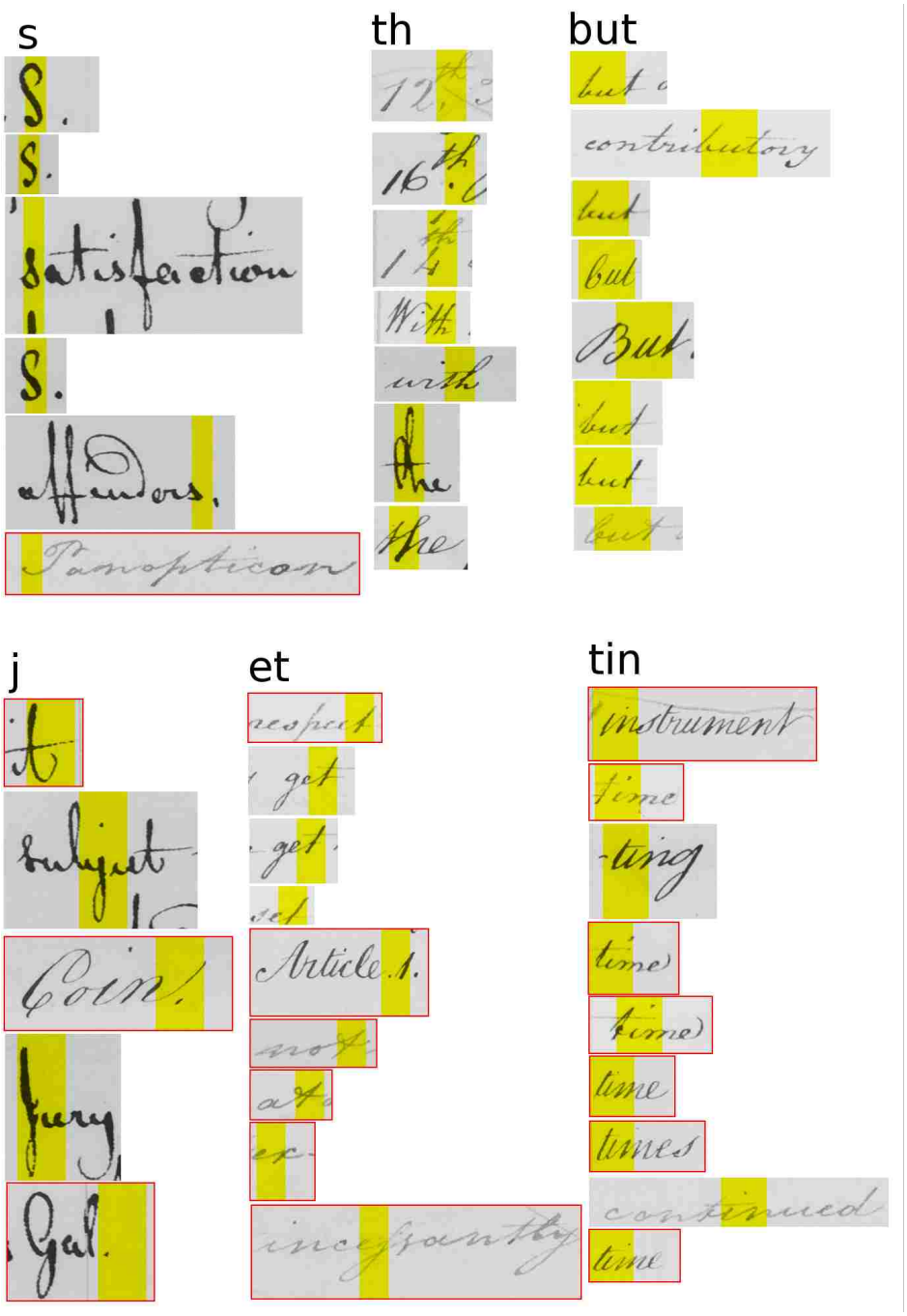


Figure 4.3: Qualitative results for QbS subword spotting on the Bentham Dataset. Spottings show the top results for the various n-gram queries. We selected some of the better n-grams ('s', 'th', 'but') and worse n-grams ('j', 'et', 'tin') by mAP. Red boxes indicate incorrect spottings.

Figure 4.4: Qualitative results for QbS subword spotting on the Census Names Dataset. Spottings show the top results for the various n-gram queries. We selected some of the better n-grams ('i', 'el', 'int') and worse n-grams ('v', 'ts', 'pre') by mAP. Red boxes indicate incorrect spottings.

In Table 4.3 we show a comparison of using cosine similarity and cross-entropy (CE) for PHOC vector comparison using both the original and adapted PHOC vectors. We also show the effect of masking the PHOC vector, where divisions of the word which do not divide evenly with the number of characters being spotted are masked out (e.g. division of thirds is masked for spotting bigrams). For QbS, using cross-entropy shows a large improvement over cosine similarity, but cosine similarity always out performs in QbE.

We note that for QbS the relative performance of unigrams, bigrams, and trigrams is reversed for the Census Names dataset. This occurs because of a common error where windows containing only part of the desired n-gram (e.g. a single character of a bigram) are scored highly. The problem becomes most aggravated when trying to spot n-grams with double letters. As these errors do not occur at all in unigrams, its mAP is higher. It can occur more frequently in the trigram case than bigram. We investigated vector comparison methods other than cosine similarity because of this problem, which is amplified by the fact that cosine-similarity ignores the network's predictions of characters not present in the query; they are multiplied by zero, though they do effect the overall normalization. Cross-entropy reduces the problem as it directly penalizes false positive character predictions.

We note that QbE seems to be relatively immune to this. It is likely due to the fuzziness of the query vector.

It is interesting that the best QbE performance is achieved with the original PHOC vector. While it is not totally clear why this is the case, we believe it is because a query based on an image is able to more richly describe the word visually with the larger PHOC vector. In an n-gram, different characters have different widths, meaning the location of each character in one trigram is slightly different than another trigram. While QbS cannot capture this, QbE with a large enough PHOC vector can.

In Section 4.1.3 we explained how we found the window sizes we used. We show in Figure 4.6 the change in mAP as the window size varies for some selected n-grams; an example word image with some windows of varying sizes can be seen in Figure 4.5 to help the

Figure 4.5: This shows a word image with windows of various widths: 200-red, 150-green, 100-blue, 50-yellow, 25-cyan.

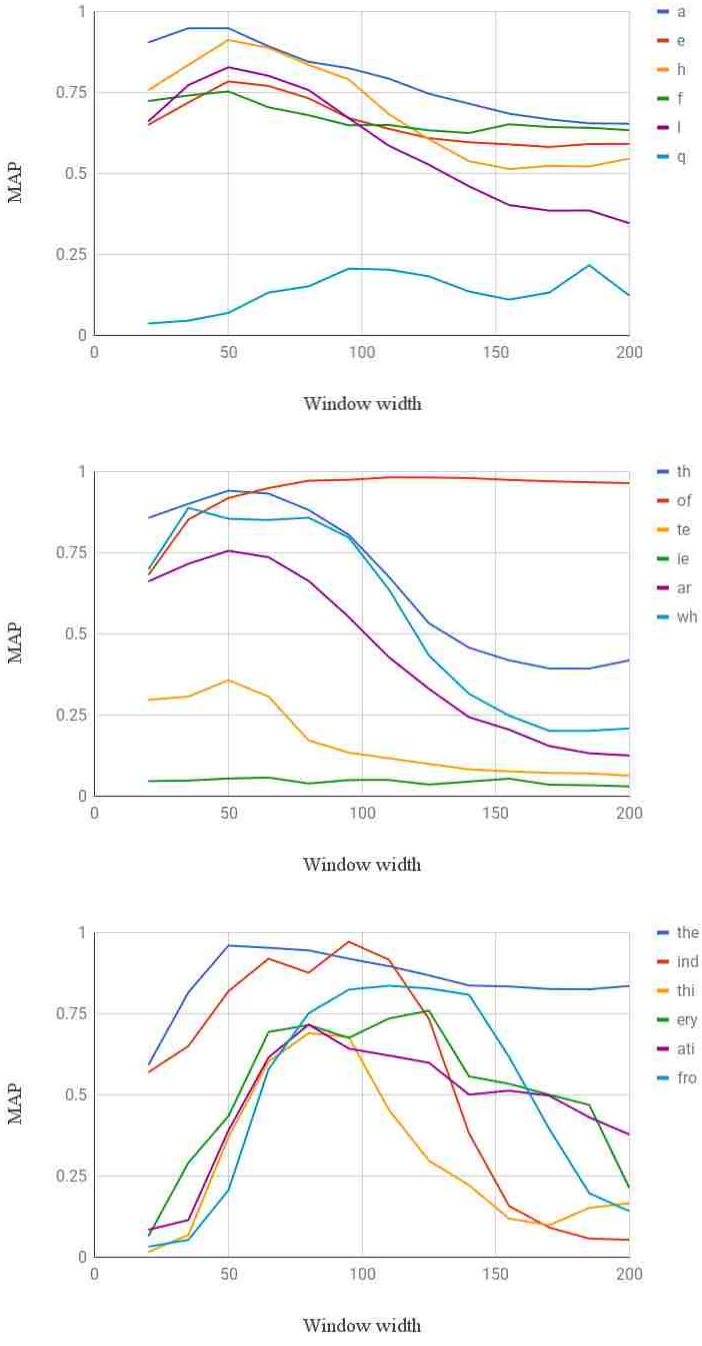


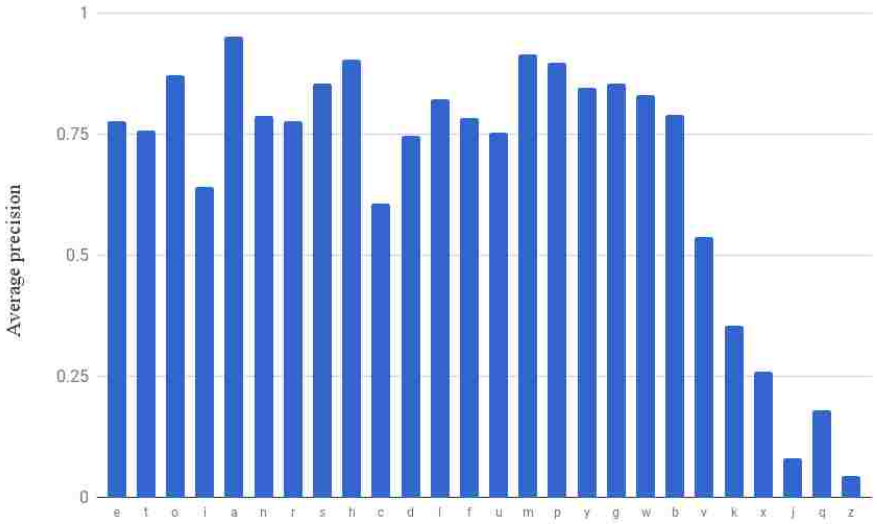
Figure 4.6: These show QbS mAP for 18 n-grams on the Bentham dataset for varying sliding window sizes (in pixels). Figure 4.5 shows examples of some windows sizes.

reader get a relative estimation of what the window sizes are. As can be seen by the peaks in the data, performance can be dependent on having an optimal window size. We cannot assume however that this is simply because the n-grams are becoming fully enclosed in the window; for example, the n-gram "ery" (maroon x) has two approximately equal peaks for widths 100 pixels apart. Simply measuring the width of the n-grams may yield non-optimal window sizes; we must test the actual performance as done in Section 4.1.3.

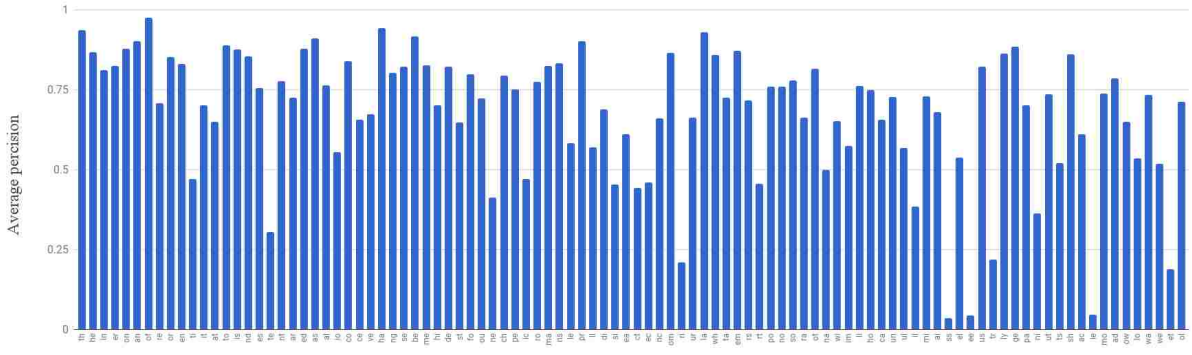
In Figures 4.7, 4.8, 4.9 and 4.10 we show, for the Bentham and Census Names datasets, the average precision of n-grams individually, arranged in descending order of frequency. While there is a general trend of more frequent n-grams being spotted better, there is a wide variance indicating that the distinctiveness of character shape likely plays an important role. We note that many less frequent trigrams are spotted with perfect precision or very poor precision. This is because of the all-or-none effect. There are so few instances (e.g. 4) that with a little bit of luck they can all end up with the top scores or all end up with poor scores.

4.2.3 Respotting Subwords

We had hoped to use approved spottings as new exemplars for an iterative CAT (computer assisted transcription) system. To verify the validity of this approach we created an experiment where we continually refine spotting results by spotting with new exemplars and merging the results (see Section 4.1.5). We use the original PHOC vector with cosine similarity, as that achieved the best QbE. We show the mAP at each iteration out to 50 iterations in Figures 4.11, 4.12, and 4.13, for uni-, bi-, and trigrams respectively on the Bentham dataset. The first exemplar for an n-gram was selected as the middle score of positive instances from a QbS spotting. The following exemplars were the middle score of the previous QbE spotting's positive instances. We chose to do this as there is a variety of qualities of exemplar images. Taking the middle score should roughly give middle quality. The overall quality of mAP is higher in these experiments as we only used n-grams which occur at least 100 times, which tend to be the easier instances. We can see that the iterative refinement leads to better



(a) Unigrams



(b) Bigrams

Figure 4.7: Results for QbS unigram and bigram spotting on the Bentham dataset. N-grams are arranged in descending order of frequency in the test set.

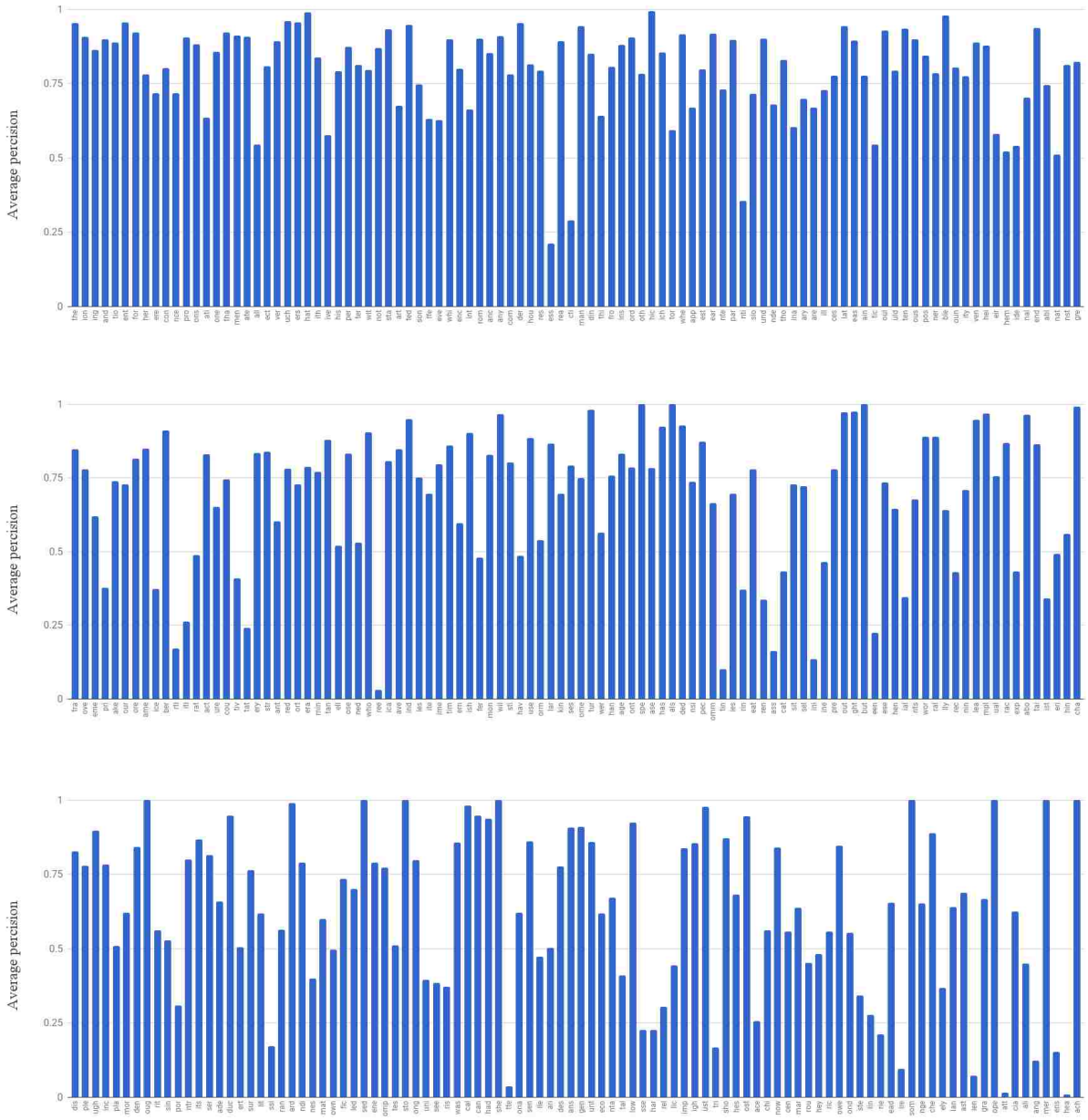
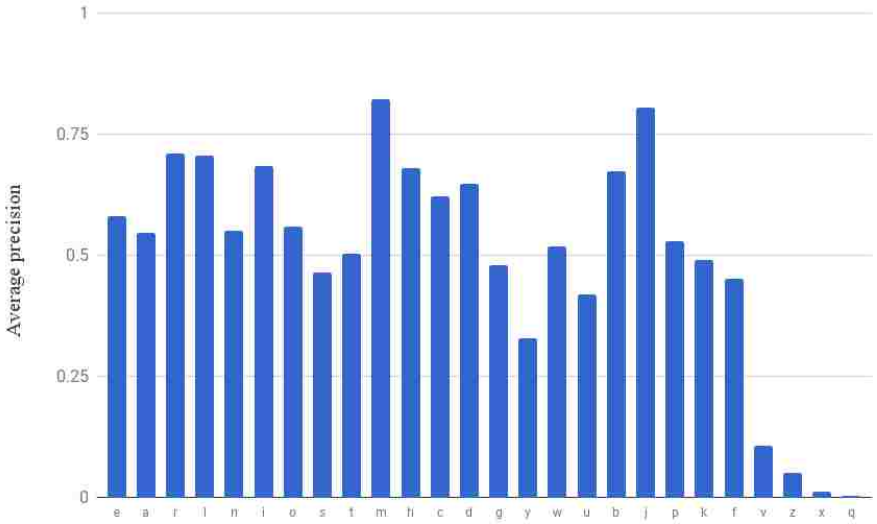
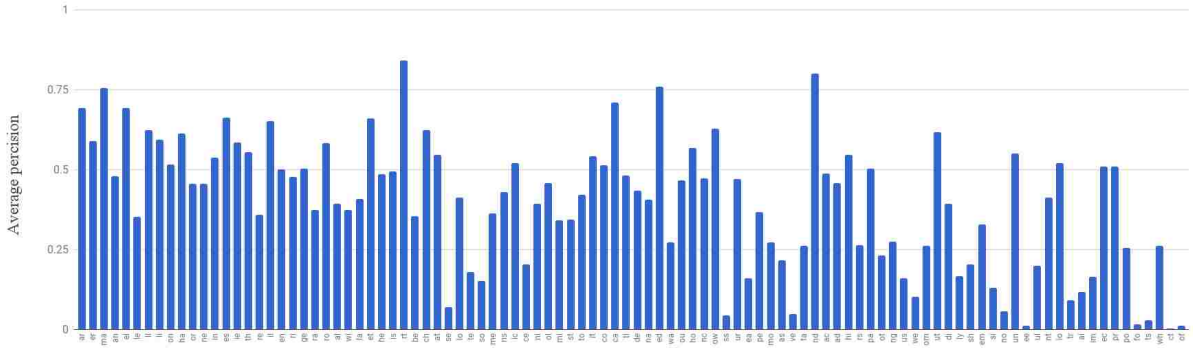


Figure 4.8: Results for QbS trigram spotting on the Bentham dataset. N-grams are arranged in descending order of frequency in the test set.



(a) Unigrams



(b) Bigrams

Figure 4.9: Results for QbS unigram and bigram spotting on the Census Names dataset. N-grams are arranged in descending order of frequency in the test set.

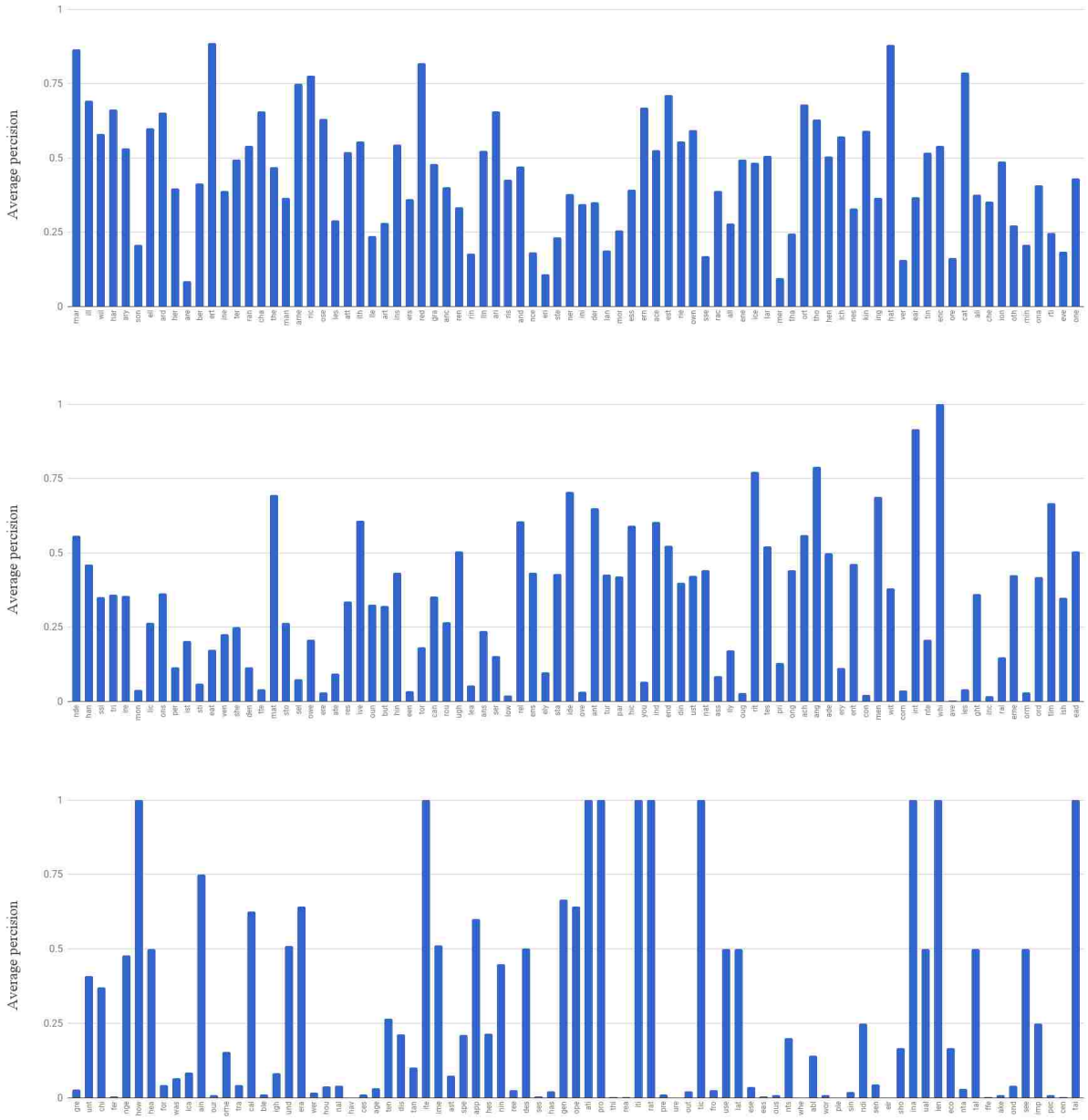


Figure 4.10: Results for QbS trigram spotting on the Census Names dataset. N-grams are arranged in descending order of frequency in the test set.

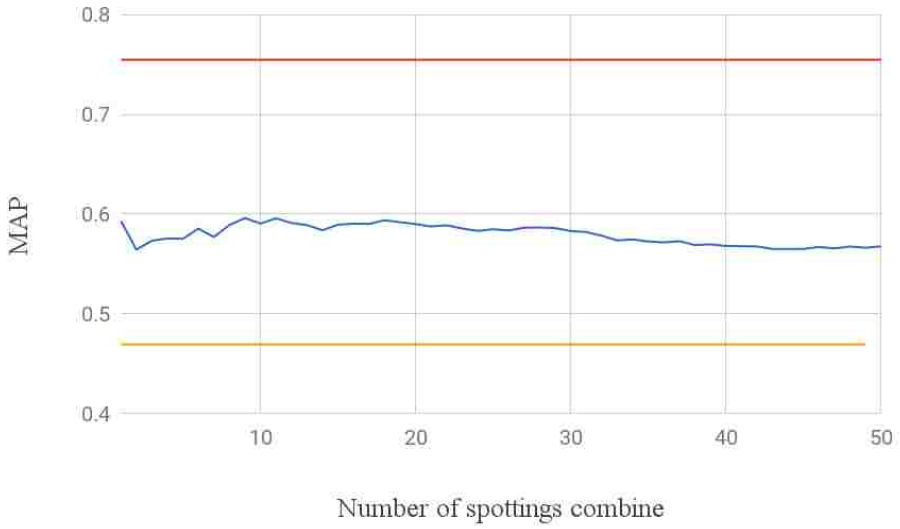


Figure 4.11: Unigram mAP when spotting results of consecutive image queries are merged (blue). The red line represents QbS performance and the yellow the average performance of all 50 QbE queries. While merged QbE spotting performs better than individual QbE spotting, it still is below QbS.

results compared to the average of all the QbE scores for the individual 50 queries, except in the case of trigrams. However, for all n-grams, the best QbE results are worse than QbS; we do not include QbE as part of our CAT system.

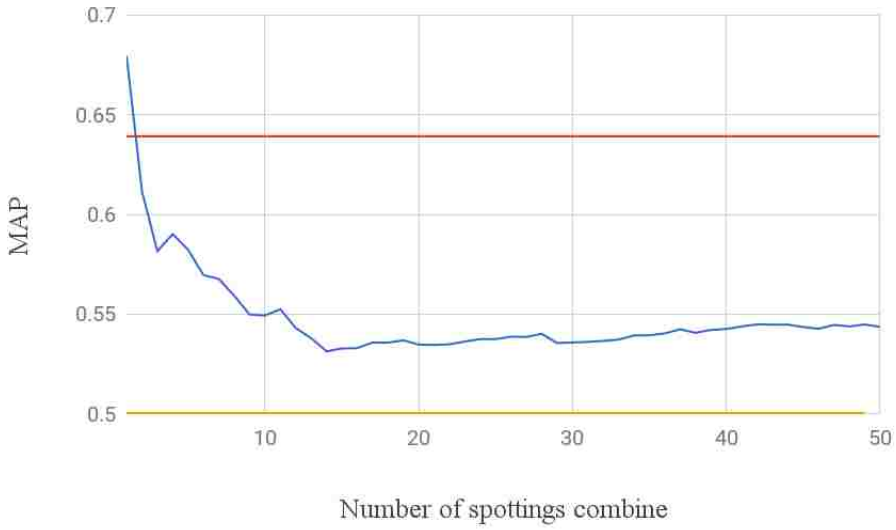


Figure 4.12: Bigram mAP when spotting results of consecutive image queries are merged (blue). The red line represents QbS performance and the yellow the average performance of all 50 QbE queries. While merged QbE spotting performs better than individual QbE spotting, it still is below QbS

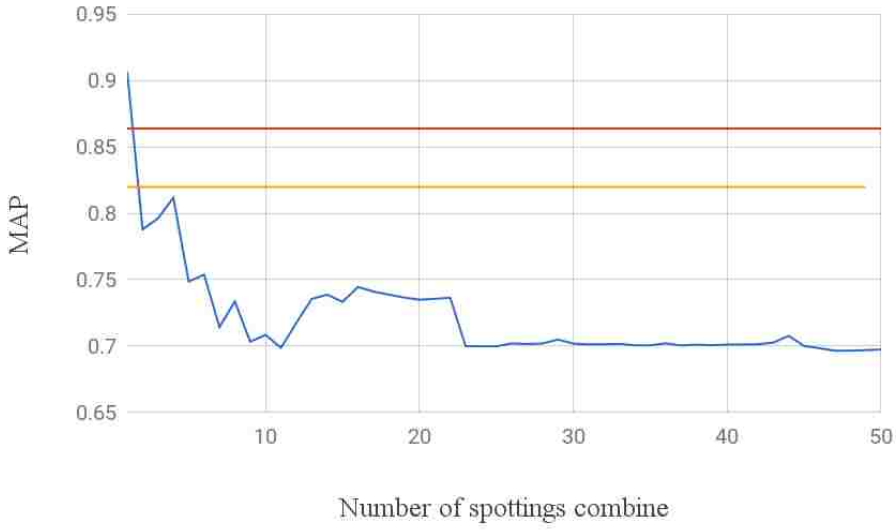


Figure 4.13: Trigram mAP when spotting results of consecutive image queries are merged (blue). The red line represents QbS performance and the yellow the average performance of all 50 QbE queries. Merged QbE spotting performs worse than individual QbE, indicating the merging may be sensitive to particularly bad spottings or queries.

Chapter 5

Applications of Subword Spotting

In this chapter we explain two applications of subword spotting: assisting persons manually transcribing document images, and suffix spotting. We explore using subword spotting to transcribe in the next chapter.

5.1 Manual Transcription Assistant

Frequently when a person is transcribing a handwritten document, they come across a handwritten word they do not recognize. A common solution is to scan the document for similar shapes which are present in the difficult word. If the transcriber can find the same letters, written by the same author, in the context of a word they do recognize, the transcriber can identify the letters, thus aiding in the transcription of a difficult word. However, the same letters may be time consuming to find due to the density of the document and rarity of the characters. QbE subword spotting provides a way to automate this scanning task. The user only needs to select the portion of the difficult word they wish to query with, and the system can return a ranked list of results. We are able to achieve real-time results over a handful of pages by using word segmentations and pre-computing PHOC vectors for a set of reasonable window sizes at 16 pixel stride. We snap users' queries to the closest pre-computed window.

Figure 5.1 demonstrates how the assistant works. First the unrecognized characters are selected (a), in this case a "G", and the bounding box is snapped to the precomputed one. This PHOC vector is compared against all others of the same window size. The ranked list is shown to the user both as a list (c) and by highlighting the results with the color intensity

Figure 5.1: Transcription assistance using subword spotting. (a) The user selects an unknown character ("G", red box) to search the documents. (b) The results of a QbE search are displayed, strength of highlight relative to spotting score. (c) A ranked list of matches is shown (best match at top). The user selects another instance to refine the results (blue arrow). (d) Combined QbE results. Exemplars are highlighted in blue. (e) Ranked combined results, a more common word, "Grace" (red arrow), has moved up to a more visible position.

Figure 5.2: Demonstrating spotting results for four different positions of the threshold slider shown below each image. The red box is the query.

representing the strength of the match (b). Suppose that the top results aren't words the user is familiar with, but the user recognizes the "G" in the first result (blue arrow, "Gash"). By selecting it, the system performs a QbE search, combining the results with that of the original query using the method described in Section 4.1.5. In the combined results (e) the word "Grace" (a more recognizable word) has moved to the third spot.

Figure 5.1 shows the strength of spottings as the strength of the highlight. However, a more interactive means of accomplishing this is a user defined threshold. The user then can slide the threshold until the results they want appear. An example of this is shown in Figure 5.2.

This assistant system is a proof of concept and no formal testing was done with it. Its performance is directly related to the QbE performance reported in Table 4.2.

5.2 Suffix Spotting

Word spotting is a technique that can be used to find certain words in a corpus of handwritten documents. However, there are certain situations where one may want to search for a partial word, such as a prefix or suffix. For example, if one wanted to find names of towns in a corpus of German documents, spotting words with the suffix "-berg" should return many town names.

We identified a list of 41 suffixes and evaluate spotting the suffixes present from this list in the IAM and Census Names dataset. The window size for a given suffix query is found by identifying all n-grams we have window estimates for (see Section 4.1.3) which compose

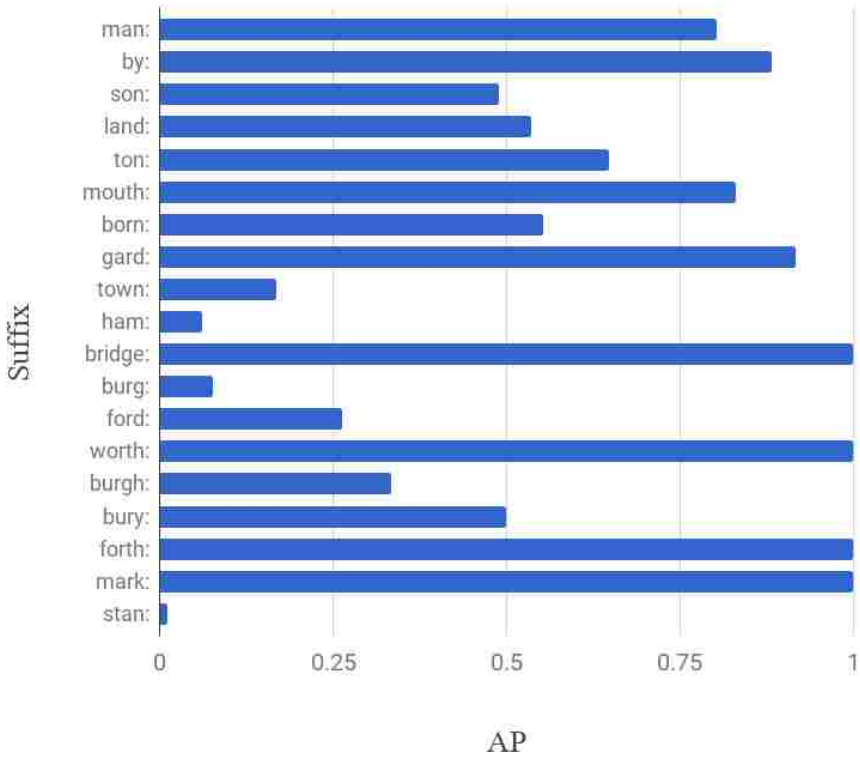


Figure 5.3: Suffix spotting AP of individual suffixes for the IAM dataset. Arranged in descending order of frequency in test set.

the query, and averaging their window widths according to which characters they contribute to. We are able to make the assumption that the spottings only occur at the end of the word, which reduces the number of possible subwindows significantly. We additionally don't need to consider window overlap in our evaluation as the goal is whole word retrieval.

We evaluate QbS suffix spotting on the IAM and Census Names datasets as these have a reasonable number of instances of our suffixes. We achieved a mAP of 0.583 for the IAM dataset and a mAP of 0.484 for the Census Names dataset. We show the AP for the suffixes individually in Figures 5.3 and 5.4. For the IAM dataset the suffixes occur about half the time as the whole words, meaning it may not represent subword spotting as well as the Census Names dataset, which had only one instance of a subword being a whole word (name).

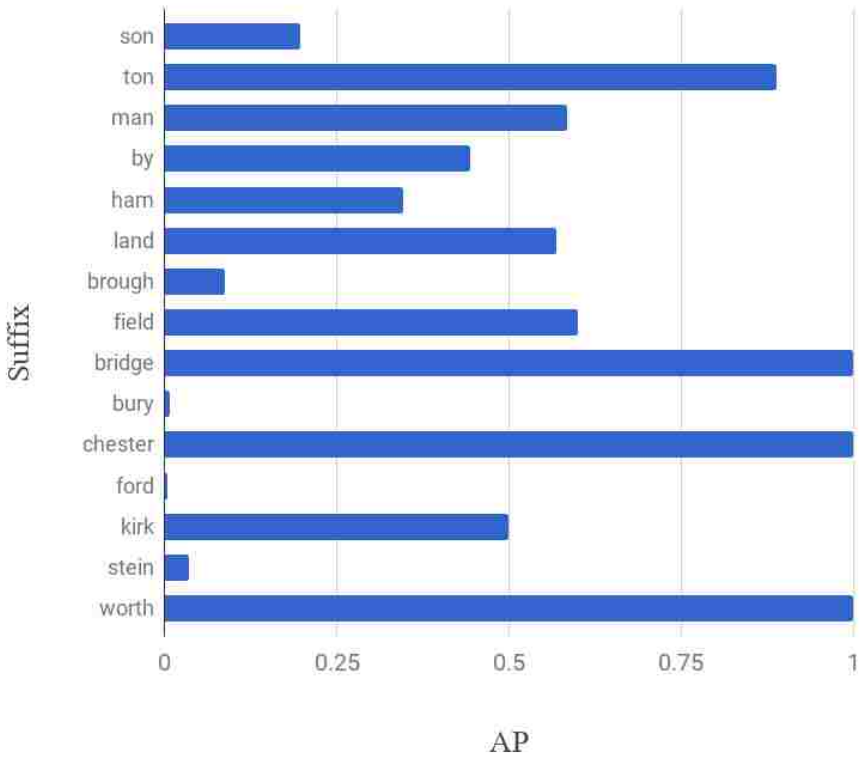


Figure 5.4: Suffix spotting AP of individual suffixes for the Census Names dataset. Arranged in descending order of frequency in test set.

Chapter 6

Application of Subword Spotting to Transcription

Subword spotting can also be used to transcribe words. There are several ways one could do this, but one of the key aspects that makes subword spotting for transcription appealing is that while subword spotting naturally yields partial transcriptions, a partial transcription can be constrained to a particular word (full transcription) by matching it against a lexicon. We aim towards a CAT (computer assisted transcription) model to better leverage the possibility of user feedback on the steps leading to the full transcription.

We describe four transcription methods in this chapter and the results in the following chapter. For all methods we used a lexicon. Our primary lexicon consists of 108,028 words and 6,939 names, as described in Chapter 3.

6.1 Baseline: CAT Through PHOC Vectors

We use a nearest-neighbors approach as a baseline to transcription using our spotting network, similar to what [12] do for word recognition. The PHOC vectors generated by the network on word images are compared to the PHOC vectors of the words in our lexicon using a cosine similarity, and the top N matches from the lexicon are returned. This does not use subword spotting, and thus provides a baseline for transcription using our PHOCNet-based architecture.

To compare it to the other CAT methods, we assume a transcription for a word image is performed by a user selecting the correct transcription from the top seven returned lexicon words. The seven words correspond with the CAT methods which return a list short enough

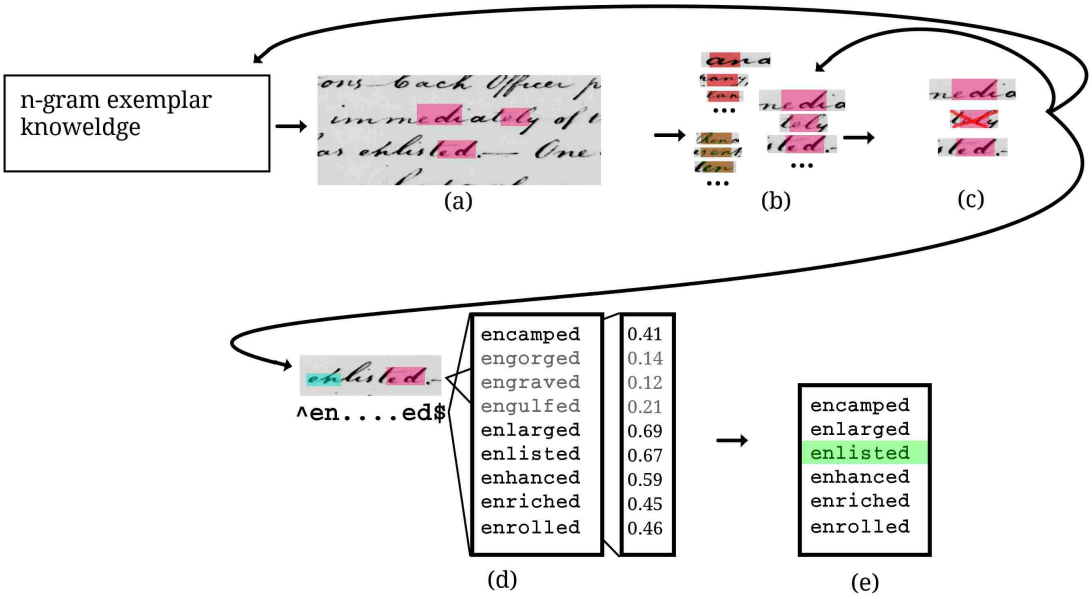


Figure 6.1: An overview of the CAT system which uses approved subword spottings. The arrows represent the flow of data. (a) Subword spotting is performed. (b) Spotting results are sorted and distributed as batches. (c) Users classify the spottings. (d) The spottings are aggregated into a regular expression. This yields a set of possible transcriptions in the lexicon. These are scored by performing word spotting on the word image. (e) The (reduced) list of possible transcriptions is sent to a user to select the correct one.

for a user to quickly review. If the correct transcription is not returned in the top seven, it must be manually transcribed. To be more efficient, we only return a certain portion of the words to be transcribed in this manner (the remainder needing manual transcription). This portion is selected by ordering words according to the mean score of the top seven lexical matches (the words that would be returned).

6.2 CAT Through Approved Subword Spottings

An overview of the CAT system using approved subword spotting can be seen in Figure 6.1. Initially the system spots character n-grams in the corpus of word images (a). These results are pooled and sent to users in batches (b). After the user identifies correct/incorrect spottings (c), the spottings' labels and positions in the word images are used to create regular expressions which are used to query a lexicon for a list of all words matching the constraints

set by the spottings (d). These are scored by word spotting each lexically relevant word in the image, and lower scores are removed from the list (according to an Otsu threshold). This list is then sent to a user, who selects the correct transcription (e). Information from approved spottings is used in further spottings.

When there are no more user tasks the system is finished. Words not transcribed will need to be manually transcribed.

We now go over the individual pieces in greater detail.

6.2.1 Subword Spotting

We follow the same procedure for subword spotting as outlined in Chapter 4. We begin by spotting the n-grams using QbS. We evenly interleave spotting the n-grams, ordering each n by frequency order. That is, we order all trigrams, bigrams and unigrams individually by frequency order, then interleave these three lists such that each list is evenly dispersed among the resulting list.

We attempted to use QbE after spottings are approved as correct by users. We combine QbE results as described in Section 4.1.5 when we had a certain threshold of approved spottings (to ensure stability). However, this did not enhance performance so we did not include this in our final tests.

To reduce the number of possible n-grams needing approval, we only return the top portion of all subwindows for a spotting: 75% for unigrams, 25% for bigrams, and 15% for trigrams. The differing percentages reflect the general frequencies of the n-grams in English.

6.2.2 User Tasks and Interface

The tasks were intended to be able to be completed on smartphones, but the UI was created as a web app to increase flexibility to more devices. The web app requests new batches from the server (system), provides a way for the user to complete the tasks, and sends the completed tasks back to the server.

There are two user tasks: spotting approval and transcription selection. Spotting approval is rejecting or accepting subword spotting results presented as a word image with a highlighted region. Transcription selection is selecting a correct transcription for a given word image, or correcting spottings displayed on the image.

6.2.2.1 Approval

The task that is presented to users most frequently is spotting approval, as most words will require multiple character n-grams, and these must be sorted out from false positives. Thus we considered the most effective way to present these to users to reduce the time spent per task.

Through some trials, we found presenting the potential n-gram spottings as a list, grouped by the n-gram label, and having the user classify each instance to be efficient. The grouping of spottings with the same n-gram label minimizes context switching, and forcing the user to make a decision about each spotting maintains accuracy.

Figure 6.2 shows the interface. The spottings at the bottom are classified as correct or incorrect either with a swipe gesture or a button press. The spottings shown above the current spotting (bottom) allow the user to look ahead and make decisions about the upcoming spottings. The colors of the highlights change when the n-gram label changes to alert the user to the change.

6.2.2.2 Transcription Selection

This task's interface can be seen in Figure 6.3. The user is presented with the image of a word, in context, with the spotted n-grams highlighted. Below are the n-gram labels, each with an 'x' button so the user can indicate an incorrect spotting. Below these is the list of possible transcriptions. They are presented in an order according to their word spotting score, as this frequently places the correct transcription at the top. At the end of this list an

Figure 6.2: Spotting approval UI. Instance being classified is at the bottom of the interface (dark border). The desired label "and" is below it. Instances are classified at the bottom and new instances are added at the top; this allows users to see upcoming instances without their fingers blocking the screen. The first two instances displayed (from the bottom) are incorrect and the third is correct. The next label "ing" can be seen above these with its associated instances above it (two correct instances visible).

Figure 6.3: Transcription selection UI. Both "her" and "for" have been spotted correctly in the image. The user can remove spottings if they are incorrect (red 'x's). The possible transcriptions are ordered according to their word spotting score; in many instances this puts the correct transcription at the top of the list.

additional "None/Error" button allows the user to indicate that the correct transcription is not present and the spotting n-grams are correct.

6.2.3 Word Completion / Regular Expression Generation

We rely on the word image width, estimated character width, and the locations of confirmed spottings (spottings approved by user or threshold) in the word to estimate how many characters have not been spotted in a word. This allows a regular expression to be generated which represents all possible transcriptions of that word.

Spottings are estimated to be a width calculated using the process described in Section 4.1.3, but without the clustering. We average these as an approximation of the character width of an unknown character.

The procedure to create the regular expression is as follows. First we measure the number of pixels between each confirmed spotting and divide this by the estimated character width, providing an estimated number of characters $(gapWidth)/(charWidth) = \epsilon$ (with fractional characters). The range of characters used in the expression is [$\epsilon - 0.8$ to $\epsilon + 0.8$] (rounded to nearest whole character). The first and last characters are similarly computed, using the beginning and end of the image boundaries as the previous and next spottings, but we additionally allow each of those ranges to have one less character than the range computed to account for the fact that frequently first and last letters are longer than others (e.g. capital letters at the beginning of a word, or a long tail on the last character of a word). If spottings overlap by less than 0.15 times the estimated character width, we allow for a character to possibly be between them. If spottings overlap and share boundary characters, for example 'al' and 'le' sharing 'l', they could be either 'alle' or 'ale'. However, because overlap is far more common ('ale' case) we choose to always merge the characters. Allowing both cases causes the resulting regular expression to become bloated and empirically perform worse. Pseudo code for this process is shown in Algorithm 1.

Once we have the regular expression, we find all matches in our lexicon.

Algorithm 1: Regular expression generation from spottings

input : spottings for the given word, consisting of the n-gram and boundaries
output : regular expression identifying possible transcriptions

```
regularExp := "";                                              /* to return */
foreach spotting do
    charsInGap := length of gap to previous spotting or beginning of image / charWidth ;
    /* using float division */
    if charsInGap > 0 then
        minNumOfChars := round(charsInGap −0.8);
        maxNumOfChars := round(charsInGap +0.8);
        if no previous spotting or no next spotting then
            minNumOfChars:= max (minNumOfChars-1,0);
        end
        regularExp := regularExp + "\w{minNumOfChars,maxNumOfChars }" + current
        spotting's n-gram;
    else
        if n-gram shares characters with previous n-gram then
            regularExp:= regularExp + characters of current spotting's n-gram not
            overlapping with previous;
        else
            minNumOfChars := 0;
            if charsInGap > −0.15 then
                maxNumOfChars := 1;
            else
                maxNumOfChars := 0;
            end
            regularExp := regularExp + "\w{minNumOfChars,maxNumOfChars }" +
            current spotting's n-gram;
        end
    end
end
return regularExp;
```

If there are no matches there are three possible scenarios: the word is not in the lexicon, there is an incorrectly approved spotting, or the spottings and unlabeled characters' count or alignment didn't transfer to the regular expression properly. To account for the second scenario, we iteratively try removing individual spottings, regenerating the regular expression, and re-querying the lexicon to see if matches are found. We make the assumption that only one spotting will be incorrect and union the matching lexicon results. We first only remove spottings that have no overlap (overlap meaning two or more spottings have identified a mutual character) and then those that do. If this fails to yield any lexicon matches, we try to account for the third scenario by regenerating the regular expression in a "loose" mode, where each range of unknown characters is expanded by one to allow matches initially constrained by a bad alignment estimation. If there are no matches and we are already using the "loose" regular expression we simply skip the word. It will be manually transcribed eventually.

If there are matches, and less than 49 of them (threshold to mean "not too many"), we score each possible transcription against the image using word spotting. As the possible transcriptions are based only on the regular expression, some will be obviously wrong from a visual point of view (e.g. missing descenders). Following this intuition, we perform an Otsu threshold on the scores to attempt to discard the obviously wrong possibilties. This thresholding is only applied if there are more than 5 possibilities, or if the threshold falls less than half the distance between the highest and lowest score (we're confident we're discarding something poor). If the resulting number of possible transcriptions is less than 7, we package them into a transcription selection task, ordered by spotting score, and enqueue it for a user to complete.

This full transcription batch creation process is described in Algorithm 2.

Algorithm 2: Transcription batch generation

input : spottings and image for the given word
output : transcription batch for word, or no batch

```
regularExp := Algorithm1(spottings);
possibleWords := matches of regularExp from lexicon
if no words in possibleWords then
    foreach spotting that does not overlap with any others do
        temporarily remove the spotting;
        regularExpTmp := Algorithm1(new spotting set);
        possibleWords := possibleWords ⋃ matches of regularExpTmp from lexicon;
    end
end
if no words in possibleWords then
    foreach spotting that does overlap with others do
        temporarily remove the spotting;
        regularExpTmp := Algorithm1(new spotting set);
        possibleWords := possibleWords ⋃ matches of regularExpTmp from lexicon;
    end
end
if no words in possibleWords then
    regularExpTmp := Algorithm1(original spotting set), but minNumOfChars and
    minNumOfChars always expanded by one;
    possibleWords := matches of regularExpTmp from lexicon;
end
if 0 < number of words in possibleWords < 49 then
    possibleScores := [];                                   /* empty list */
    foreach word w in possibleWords do
        possibleScores ← WordSpot(word image, w);          /* insert score into
        possibleScores  */
    end
    thresholdScore := FindOtsuThreshold(possibleScores);
    minScore := minimum score in possibleScores;
    maxScore := maximum score in possibleScores;
    if number of words in possibleWords > 5 OR minScore +( maxScore − minScore
    )/2 < thresholdScore then
        possibleWords := PerformThresh(thresholdScore,possibleScores,possibleWords);
    end
    if number of words in possibleWords < 7 then
        return CreateBatch(possibleWords);
    end
end
do not return batch;
```

6.2.4 Receiving Transcription

The user returns a transcription selection task in one of three ways: with a selected transcription, with a spotting marked for removal, or with an error indication (see Section 6.2.2.2). These are handled in the following ways.

If a transcription was selected, it is stored as the transcription for the word. We attempted to extract new n-gram image exemplars by interpolating where unspotted n-grams were in the word image, but these generally failed to be good exemplars so we chose not to use them.

If a spotting is marked for removal, it is removed and the process described in Section 6.2.3 is followed again, potentially leading to a new task being enqueued.

If the user indicates an error, we first attempt to loosen the regular expression, as described in Section 6.2.3. If no matching words are found, we remove the "worst" spotting. The "worst" spotting is any spotting labeled by a threshold (not user oversight). If no thresholded spottings are present, the "worst" is determined by the worst spotting score.

6.2.5 Batch Distribution

When approving spottings it is a non-trivial task to distribute the spottings to users in an efficient manner. We would like to maximize the impact of humans efforts, ideally quickly finding thresholds for each n-gram that can be used to automatically label the remaining spottings. We tried several approaches to this.

The simplest method is to simply start with the spotting with the lowest score until the user begins to classify enough as false that we can assume we are at a good thresholding location to reject the remaining spottings. This has the downside of not looking for a possible threshold to accept spottings, and every spotting must be approved, which is an inefficiency.

Looking at a graph of true and false spotting instances in Figure 6.4, one can see a distinction between the true and false spottings. If we can identify the boundary between them, it would give us a useful tool to create both an accept and reject threshold. In Figure

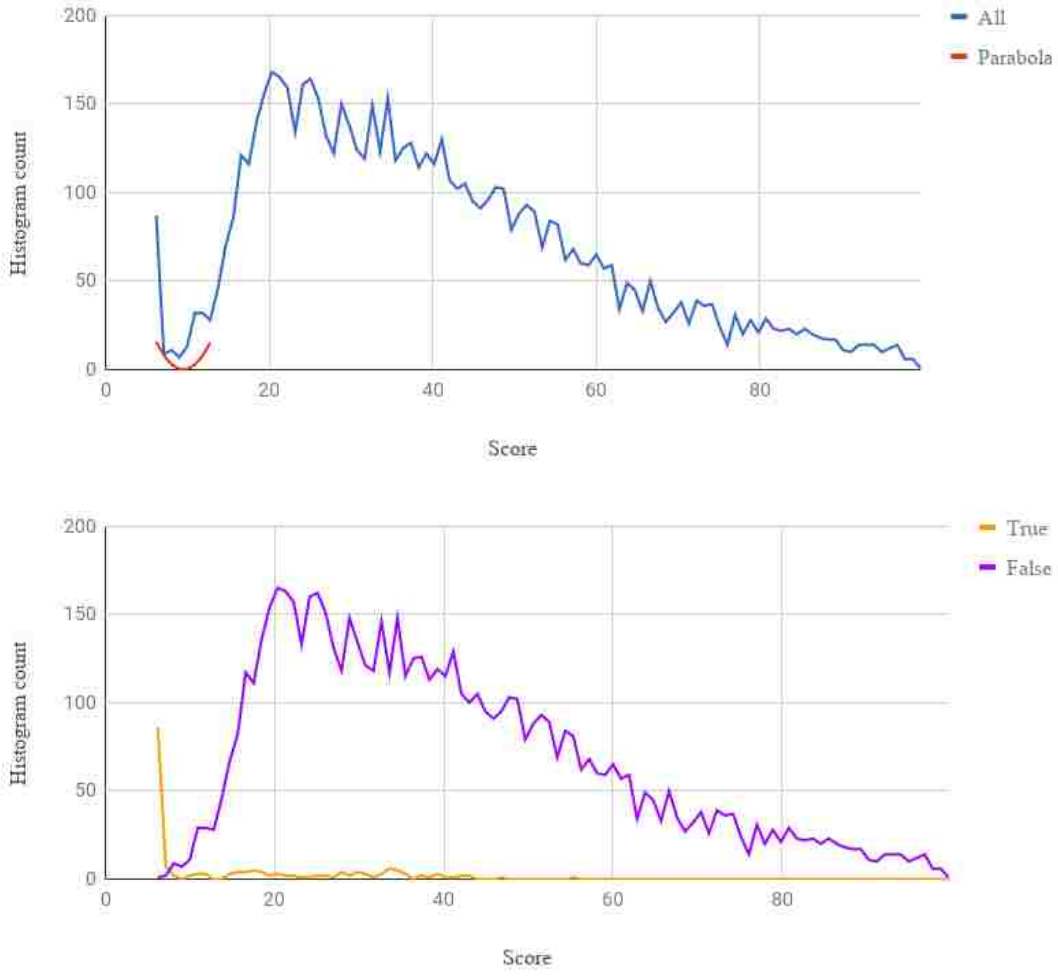


Figure 6.4: Histograms of spotting instances for the trigram "and." The left chart shows the instances aggregated as well as the fitted parabola. The right chart shows the true and false spottings separated. Note the clear separation being modeled by the parabola.

6.5 we see that the distribution of false spottings (yellow line) overlaps significantly with the true spottings (magenta line) making it difficult to distinguish a division boundary when viewed combined (blue line).

We identify the true/false boundary by the following procedure. We sample the best scores, those below 13, and perform a least-squares fitting of a parabola to a histogram of the data (e.g. Figures 6.4 and 6.5). The threshold 13 was heuristically chosen as the peak of the false distibution occurs higher than 13 (generally). This parabola should roughly match the curve of the data and is shown as the red line in Figures 6.4 and 6.5. We use the x-position

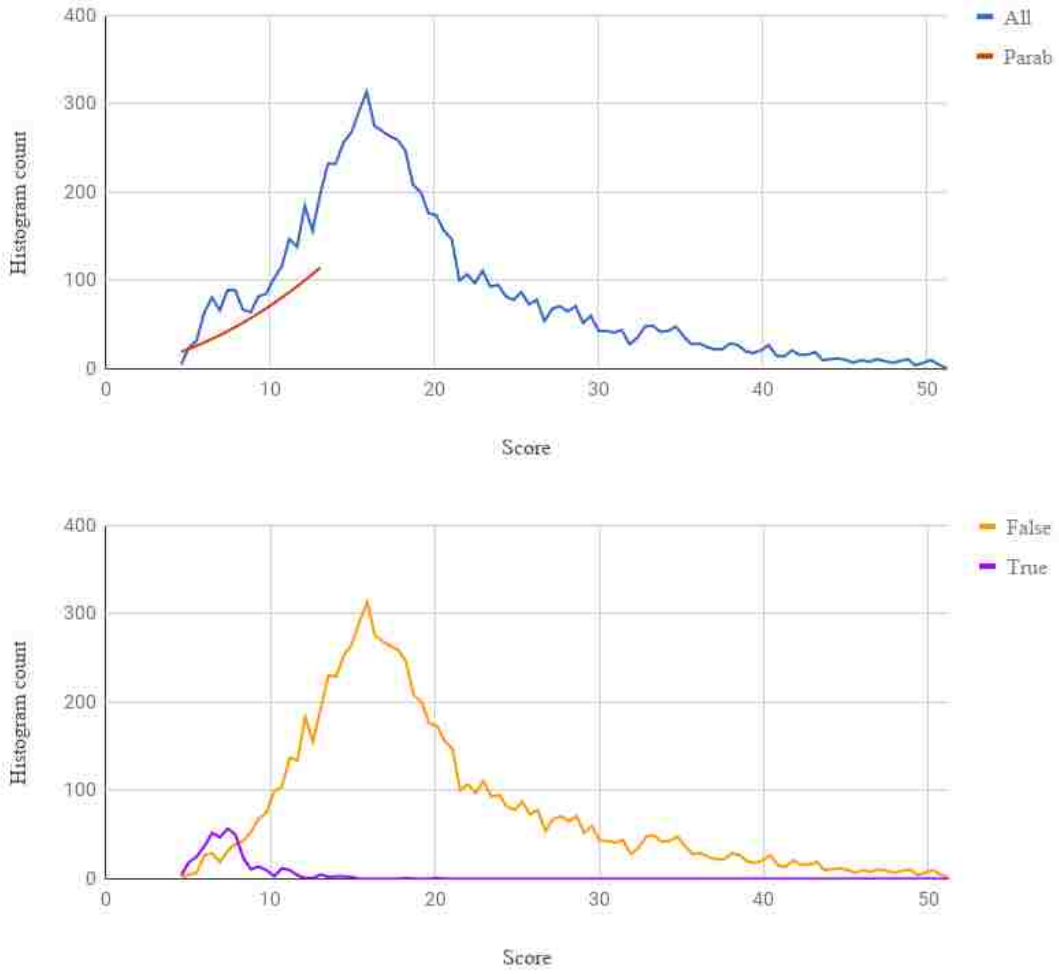


Figure 6.5: Histograms of spotting instances for the bigram "ti." The left chart shows the instances aggregated as well as the fitted parabola. The right chart shows the true and false spottings separated. Note that there is no clean division and the parabola's vertex falls outside the data range.

of the vertex to determine where the true/false split is. In a clean split, as in Figure 6.4, we can see how the parabola matches to two raising distributions, meaning the vertex is a rough mid-point between them. In a case where there is not clean separation, as in Figure 6.5, the parabola half matches the data, causing the vertex to be past the data (or at least very near the minimum score).

While fitting a parabola may seem strange, its simplicity makes it effective. We tried more complicated methods of modeling the data, but these failed too frequently.

When creating a batch, we select instances starting at the estimated true/false boundary and alternately select the next instance (ordered by score) on the true and false sides. Intuitively this means we begin with the instances in which we are least confident about our classification and work towards those in which we are more confident.

6.2.6 Receiving Spotting Approvals

When user labeling is received, we update running averages of approval for instances of the true side and false side of the split. If either of these pass a threshold, we accept or reject all remaining instances on the respective side. We accept remaining true-side instances if the average approval rate exceeds 0.92. We reject the remaining false-side instances if the average approval rate falls below 0.75.

For each approved spotting, we add the spotting to the word and execute the procedure described in Section 6.2.3.

6.3 CAT Through Approved Subword Spottings Using Clustering

The approval of spottings is a bottle-neck for the previous transcription strategy. In [20], they sped up approval by clustering printed characters together and presenting the user with a composite of a cluster. While handwriting is too noisy to composite, we follow a similar method of clustering n-gram spottings together and presenting users with (part of) a cluster to approve or disapprove. Classifying a group of spottings which belong to the same

class should take as long as a single instance; the catch is that we don't know if a group of spottings is all right or all wrong. It's likely our clustering is imperfect so that there is a mix. However, in the case where only a few instances in the cluster are outliers, they should be quickly identified by a user (see Figures 6.6 and 6.7). This follows the same spirit as the UI for [5] which also leveraged humans' ability to quickly identify outliers. In the case where the cluster is totally mixed (50%/50%) one can classify each instance individually, which is what was being done in the above method.

In this transcription strategy we use the same method as above, but change how spotting batches are distributed and don't use QbE in the same way. We will describe the steps of this process next.

6.3.1 Clustering

We first perform a dense QbE comparison for the results of each initial n-gram QbS spotting. This gives us a similarity for all instances. We then perform a hierarchical complete-linkage agglomerative clustering (two clusters are merged if their two most dissimilar spottings are more similar than any other cluster pair). We choose a hierarchical method as this allows us to tune the "purity" of clusters as we receive human feedback. "Purity" is the measure of how much the clusters are of one class ($2 * max(T, F)/(T + F) - 1$ where T and F are the count of true or false spottings in the cluster). The initial level in the hierarchy is the first level where the average cluster size is above 10, which is roughly what we feel an ideal batch size is.

6.3.2 Spotting Approval Batch Distribution

We evaluated two methods of distributing clusters. One simply takes the cluster with the next highest average score. The other method attempts to identify good clusters based on their similarity to approved spottings. To do this we save each approved spotting in a queue. When finding a new cluster to distribute, we take the next instance off of the queue. We then

find the cluster that is closest using the complete-linkage metric, that is the cluster where the instance with the furthest distance from our exemplar is minimized.

In both methods, if a selected cluster is too large (maximum size of 15) we split it and send each piece as a separate batch.

We have a "purity" goal of 0.9 (roughly 1 or 2 outliers per batch). When our moving average purity varies from this (over an ϵ threshold) we move up a level in the hierarchy if the purity is too high, or move down a level in the hierarchy if the purity is too low. If the goal were too high (e.g. 1.0), we would quickly move to a level where the clusters are very small, which would decrease efficiency.

When our moving average accuracy (the portion of instances classified as true by the user) falls below a threshold (0.4) we stop sending batches for that n-gram.

6.3.3 Spotting Approval UI

Examples of the UI are shown in Figures 6.6 and 6.7. The idea is to allow the user to recognize whether the majority of the instances are correct or incorrect and then select the outliers. When a user selects an instance it is darkened. There are two buttons at the bottom which are used to both indicate the cluster type (correct/incorrect) and that they have selected all the outliers (they are finished). This is similar to the UI in [5], however, in ours we also handle the bad cluster scenario.

This is a minimalist interface placing a fair amount of cognitive load onto the user in exchange for efficiency. One can imagine a slightly simpler interface where the cluster type is identified as a button prior to a "done" button.

6.4 CAT Through Unassisted Subword Spotting Using Dynamic Time-Warping Alignment

We want to avoid having to make decisions about spottings, either by thresholding or having user input, which the previous methods rely on. Thresholding forces a decision between recall

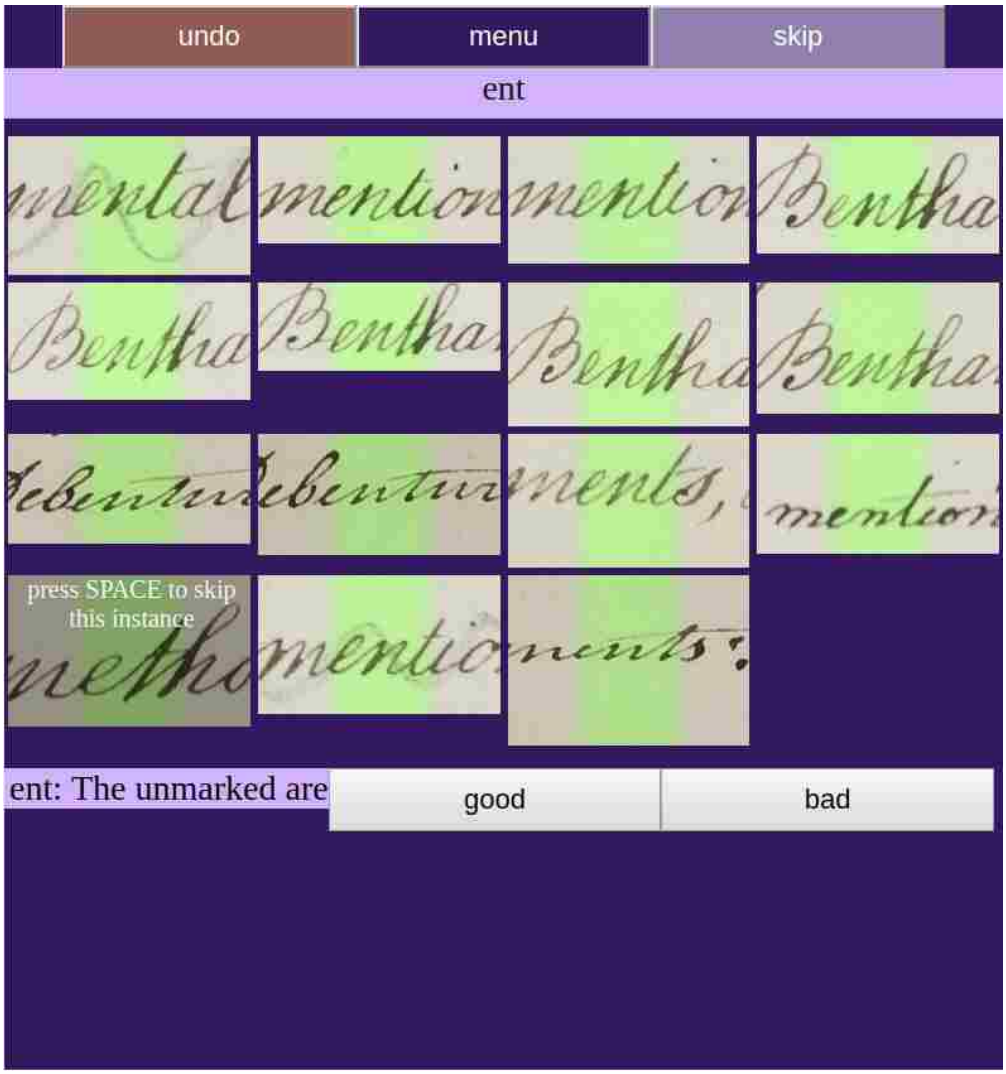


Figure 6.6: The cluster spotting approval UI with a true cluster. One outlier has been identified by the user (darkened instance).

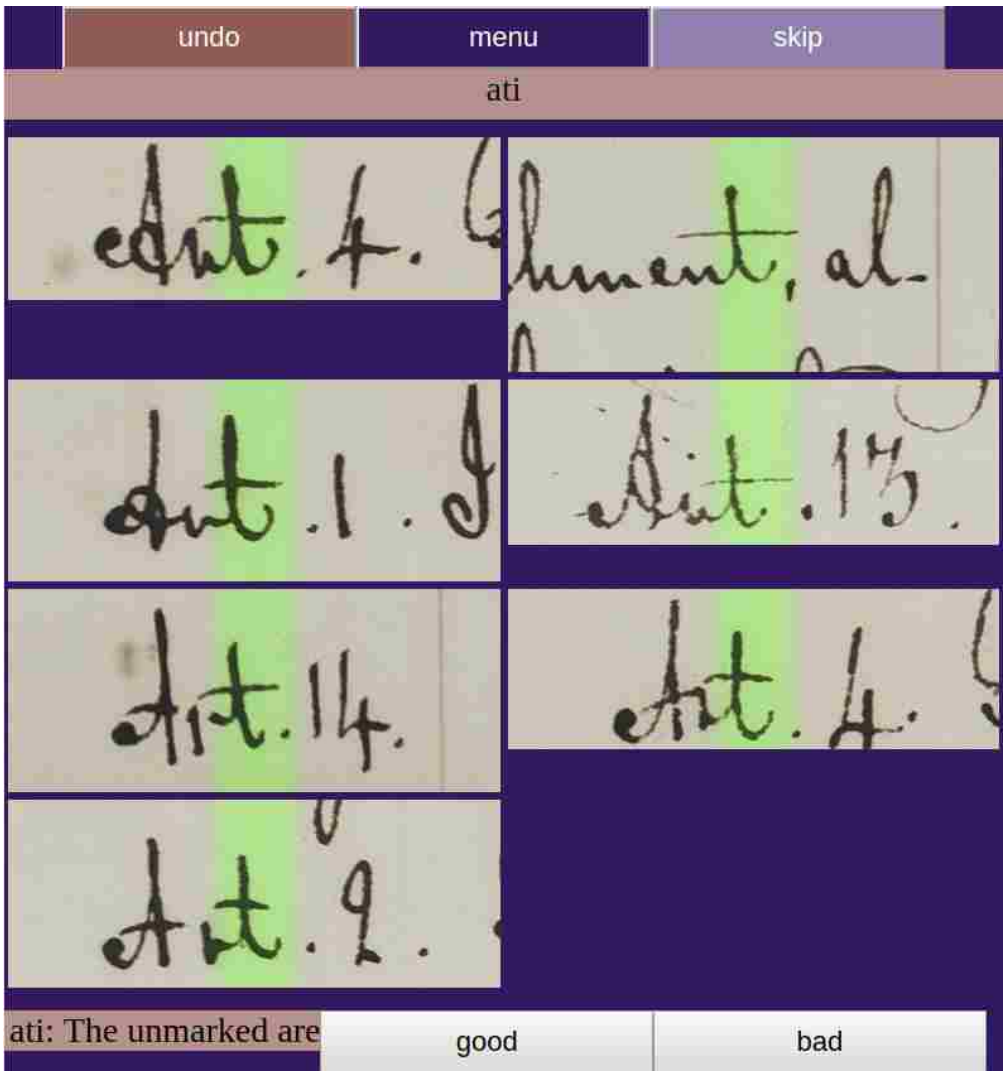


Figure 6.7: The cluster spotting approval UI with a false cluster. No outliers are present, all instances are incorrect.

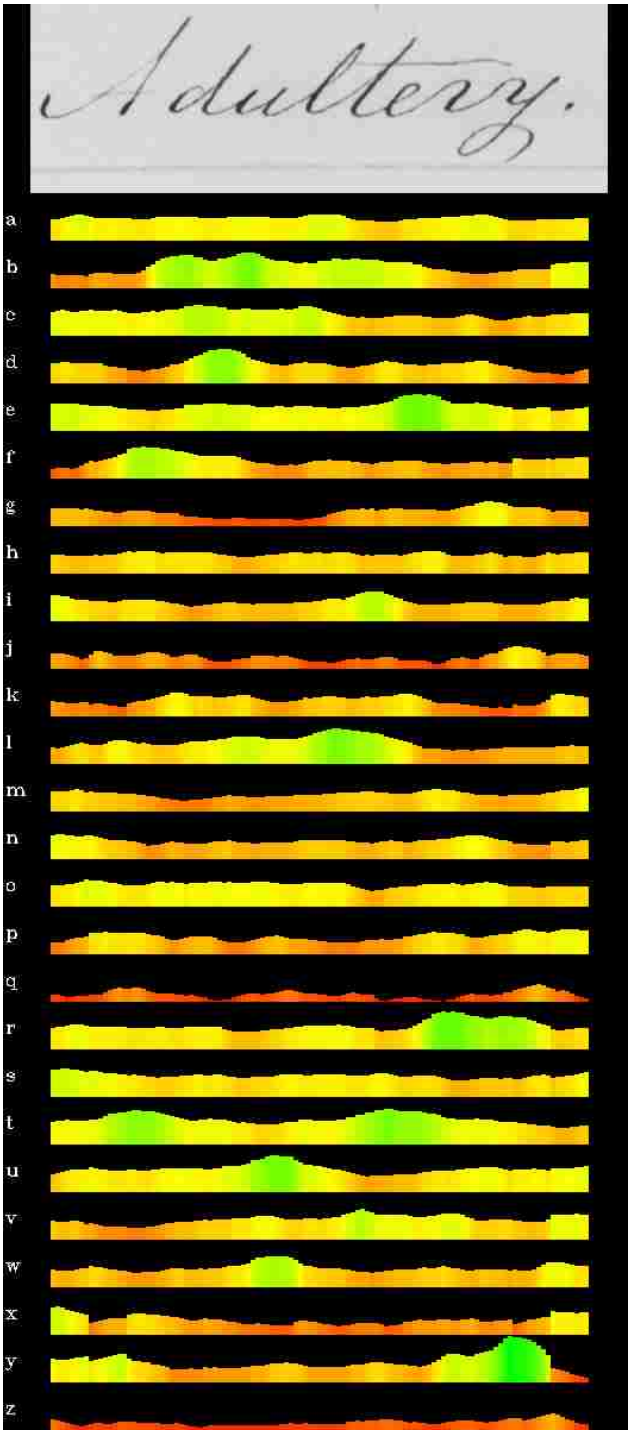


Figure 6.8: An example of a character probability vector for the word "adultery." The relative probability of each character ('a'-'z') at each horizontal position is represented both by the height and color of the graph. The discontinuities occurring at the begining and end of the word (e.g. see 'x') occur due to the merging of unigram, bigram, and trigram scores.

and precision, and user input is costly. A way to avoid these is to simply merge all spottings together into a single character probability vector, resulting in something like Figure 6.8, and then decode this.

The combining happens according to the following procedure, where we process each n (1, 2, and 3) separately into an independent vector before combining them. Each spotted n-gram produces a score at each location of the sliding window. For unigrams this gives us a vector for a relative scoring of a character at each location. For bigrams and trigrams we can offset these locations and create vectors for each character present in the bigram or trigram; these are then summed together. Each n-specific character vector is normalized according to how many n-grams contributed to that character. If a character has had no n-grams contribute to it, we set it to the minimum value observed.

The individual n-specific character vectors are summed. This summed result has a soft-max applied to each horizontal position (over characters) to yield the final character probability vector. There is a little extra work involved given n-grams are spotted at different window sizes; the full process is detailed in Algorithm 3.

We then use dynamic time warping to score the character probability vectors against the lexicon words, the lexicon words being converted a series of one-hot character vectors, one for each character of the word. Similar to the transcription via PHOC method we can then return the top seven matches to the user to select the correct transcription.

Algorithm 3: Character probability vector creation

input : word image
output : character probability vector for word

characterSums is a matrix tracking the combine scores for each n and character, indexed by n, character, position;
characterCounts is a matrix of the same size as characterSums, but tracks how many spottings contributed at each position;
foreach n *in* $\{1, 2, 3\}$ **do**
  foreach n-gram *of length* n **do**
    PHOCs := `DenseSpot(n-gram); /* PHOC vector for each window location along word image */`
    foreach *position* l *in* n-gram **do**
      character := n-gram $[l]$;
      offset := `GetOffset` (l,n,n-gram); `/* Based on estimated character width and window width */`
      characterSums $[n]$[character] := characterSums $[n]$[character] + `OffsetVector(offset,PHOCs);`
      characterCounts $[n]$[character] := characterCounts $[n]$[character]+`OffsetVector(offset,1s); /* 1s is a vector of ones of the size and position as PHOCs. */`
    end
  end
end
CPV is a matrix the same size as characterSums $[n]$;
foreach character **do**
  foreach *horizontal position along the word* p **do**
    numContributing := 0;
    foreach n *in* $\{1, 2, 3\}$ **do**
      if characterCounts $[n]$[character]$[p]! = 0$ **then**
        numContributing := numContributing +1;
        CPV [character]$[p]$:= CPV [character]$[p]$+characterSums $[n]$[character]$[p]$/characterCounts $[n]$[character]$[p]$;
      end
    end
    if numContributing == 0 **then**
      CPV [character]$[p]$:= minValue;
    else
      CPV [character]$[p]$:= CPV [character]$[p]$/numContributing;
    end
  end
end
foreach *horizontal position along the word* p **do**
  CPV $[*][p]$:= `SoftMax(CPV` $[*][p]$`); /* Soft-max occurs across characters */`
end
return CPV

Chapter 7

Evaluation of Transcription Strategies

In this chapter we explain our procedure for testing the transcription strategies described in the previous chapter. Then we examine the results of these tests.

7.1 Evaluation Method

We evaluate all our transcription strategies as CAT (computer assisted transcription) methods. However, rather than having users use the systems, we simulate approximate user behavior using collected statistics. While the statistics may not give results as accurate as a user study, it allows us to easily compare the performance of different transcription strategies using consistent timing data.

The statistics were collected from four members of our team using the system. We collected timing information regarding how quickly they completed batches, whether they were correct (against the corpus ground truth), as well as information regarding the batch's "difficulty." "Difficulty" being some objective parameter(s) which can effect speed of users completing the task. For spottings "difficulty" was the size of the batch and the number of true instances. For transcription selection "difficulty" was the position of the correct transcription (if present). We fit linear models to predict the time needed to complete a batch based on its "difficulty." We also did this for errors. We collected data for both spotting approval methods (instance and cluster).

In more detail, we measured which "difficulty" measure best modeled the data and used that. For transcription selection, there is a fixed time predicted if the correct transcription is

not present, otherwise a linear model based the transcription's position in the list is used to predict time. Two fixed values predict the error for the two transcription situations. For the Bentham dataset, instance spotting approval timing is based on whether the previous batch is the same n-gram. For the Census Names dataset, instance spotting approval timing is based on the number of true instances. For both datasets, error is based on the number of true instances in the batch. For both datasets, cluster spotting approval timing and error is based on the number of instances in the batch.

We also measured how long it took to manually transcribe line images so we could see if the CAT actually sped up the transcription process.

To evaluate, we ran each strategy as a CAT system with a single simulated user. When the simulated user received a batch, it found the correct response and then measured the probability of error and introduced an error if a random value between [0,1] fell below that probability. The simulated user then computed the estimated time based on the appropriate model and waited that long before returning its answer. For many of the transcription strategies, there comes a point where the strategy must be abandoned and the remaining words manually transcribed. We terminate our simulations at this point, but report how much was transcribed.

7.2 Results

We measure the speed of the transcriptions using a words-per-minute metric. This is averaged over the time of the system running, as it varies significantly during the running for the spotting approval method. The results are summarized in Table 7.1. In the following subsections we analyze the results of each method. The results of [37] are included in Table 7.1, however, it is apparent from the differing manual transcription speeds that our experimental set ups are too varied to allow a direct comparison.

We note that most methods have better word-error-rate (WER) than manual transcription. This is because all are aided by the lexicon. One can imagine that the manual

Method	Bentham			Census Names		
	Transcribed	Words/Min	WER	Transcribed	Words/Min	WER
Manual	-	24.27	0.073	-	20.92	0.160
[37] manual	-	6.28	-	-	-	-
[37] CAT	1.0	14.2	-	-	-	-
PHOC vectors 100%	0.872	24.66	0.021	0.691	11.53	0.070
PHOC vectors 75%	0.687	28.42	0.015	0.531	11.27	0.070
PHOC vectors 50%	0.458	29.22	0.016	0.386	13.88	0.043
CAT through approved subword spottings						
Normal UBT	0.588	6.39	0.024	0.354	2.51	0.058
Closest cluster UBT	0.685	2.56	0.019	0.532	1.14	0.075
CAT through unassisted subword spotting using DTW						
CPV+DTW UB	0.563	8.74	0.082	0.337	4.17	0.219

Table 7.1: Highlights in the results from simulations. The letters U, B and T represent whether unigrams, bigrams and/or trigrams were used. The PHOC vector method is the only method we tested that is able to transcribe words at a rate faster than manual transcription. We note that [37] reports manual transcription far below the one we collected. This is probably due to experimental setup. Their CAT system gets almost a 50% speed-up from their reported manual time.

transcription method could be augmented by showing users close lexical matches to the word a user is typing. This should make manual transcriptions WER comparable to the other methods at the cost of some speed as users will spend some time looking at the lexical matches.

7.2.1 CAT Through PHOC Vectors Results

This baseline result is the only method to surpass manual transcription time, and it only did so for the Bentham dataset. It does not use subword spotting. We show results in Table 7.1 for three variations where the portion of words actually sent to the user is changed (i.e. we discard the worst words). As would be expected, decreasing the portion of words sent to the user has a linear relationship with the portion transcribed, however, the speed does not follow this linear relationship. This simply means that the order of good transcription selection batches doesn't correlate directly to score.

We assume that the speed of this method on the Census Names dataset does not surpass manual transcription due to the greater variation in the handwriting. In general,

	Bentham			Census Names		
Method	Transcribed	Words/Min	WER	Transcribed	Words/Min	WER
Normal U	0.121	10.78	0.107	0.130	2.53	0.032
Normal B	0.337	7.28	0.026	0.076	2.53	0.093
Normal UB	0.475	7.78	0.021	0.219	2.53	0.049
Normal BT	0.489	5.67	0.025	0.213	2.54	0.057
Normal UBT	0.588	6.39	0.024	0.354	2.51	0.058
Closest cluster U	0.266	4.93	0.047	0.171	4.17	0.036
Closest cluster BT	0.686	2.55	0.018	0.532	1.16	0.080
Closest cluster UBT	0.685	2.56	0.019	0.532	1.14	0.075
Top cluster U	0.274	5.05	0.050	0.172	4.34	0.033
Top cluster BT	0.693	2.52	0.020	0.529	1.15	0.076
Top cluster UBT	0.689	2.54	0.019	0.530	1.15	0.073

Table 7.2: Results from simulations of CATTSS using user approval on subword spotting results. The letters U, B and T represent whether unigrams, bigrams and/or trigrams were used.

humans are typically better at handling variation. Transcription selection batches without the correct transcription generally take some time to be rejected by users. If the system presents many of these (due to the difficulty of the handwriting), it makes sense that manual transcription would be much faster.

7.2.2 CAT Through Approved Subword Spottings Results

We show more detailed results in Table 7.2 for the following three variations of batch distribution: modeling the "true-false break point" (normal), clusters serving closest-cluster-next and clusters serving highest-scored-cluster-next. We also experimented with other methods of distributing batches, however, these outperformed the other approaches. We don't show an exhaustive list of n-gram combinations, but show those that both cover a variety and the best performance.

All of these methods perform far below manual transcription. In the best variation for the Bentham dataset the simulated user spent 56% of its time approving spottings. The effort required for the user to supervise the subword spotting represents a large slowdown to the system's speed. The Census Names dataset is consistently much slower than the Bentham

dataset as the handwriting in it is more difficult and thus requires even more user supervision in approving subword spottings. Additionally, we run into the same slow-down from bad transcription selection batches described in the previous section.

A page of the testing corpus of the Bentham dataset is show in Figure 7.1 annotated with the spottings which were approved as well as the transcribed words.

7.2.3 CAT Through Unassisted Subword Spotting Using Dynamic Time-Warping Alignment Results

In the previous section we saw the subword spotting approval as a major bottleneck. We removed that bottleneck in this method by aggregating all spottings into character probability vectors and comparing these directly to our lexicon words using DTW. We see large gains doing this, but we still fail by a wide margin, to surpass manual transcription speeds. Details of the results can be seen in Table 7.3. The best results used unigrams and bigrams. These are the smaller n-grams which give less information per instance, but that isn't an issue to this method as it does not require n-gram approval

Given the similarities between this method of using a character probability vector and the baseline method using PHOC vectors, we think it valuable to describe why this method performs worse. Both the PHOC vector approach and the subword spotting approach use the same network. This network was trained on full word images; this means it will generally have better performance on word images, which the PHOC vector approach uses. Additionally, there is some ambiguity introduced by the CPV method's sliding windows, which are not precise in their localization.

The WER for this method is surprisingly low. This is likely to many transcription selection batches being served which do not contain the correct transcription. Users often make the mistake of trusting the system too much and assuming one of the presented words is correct even if none are.

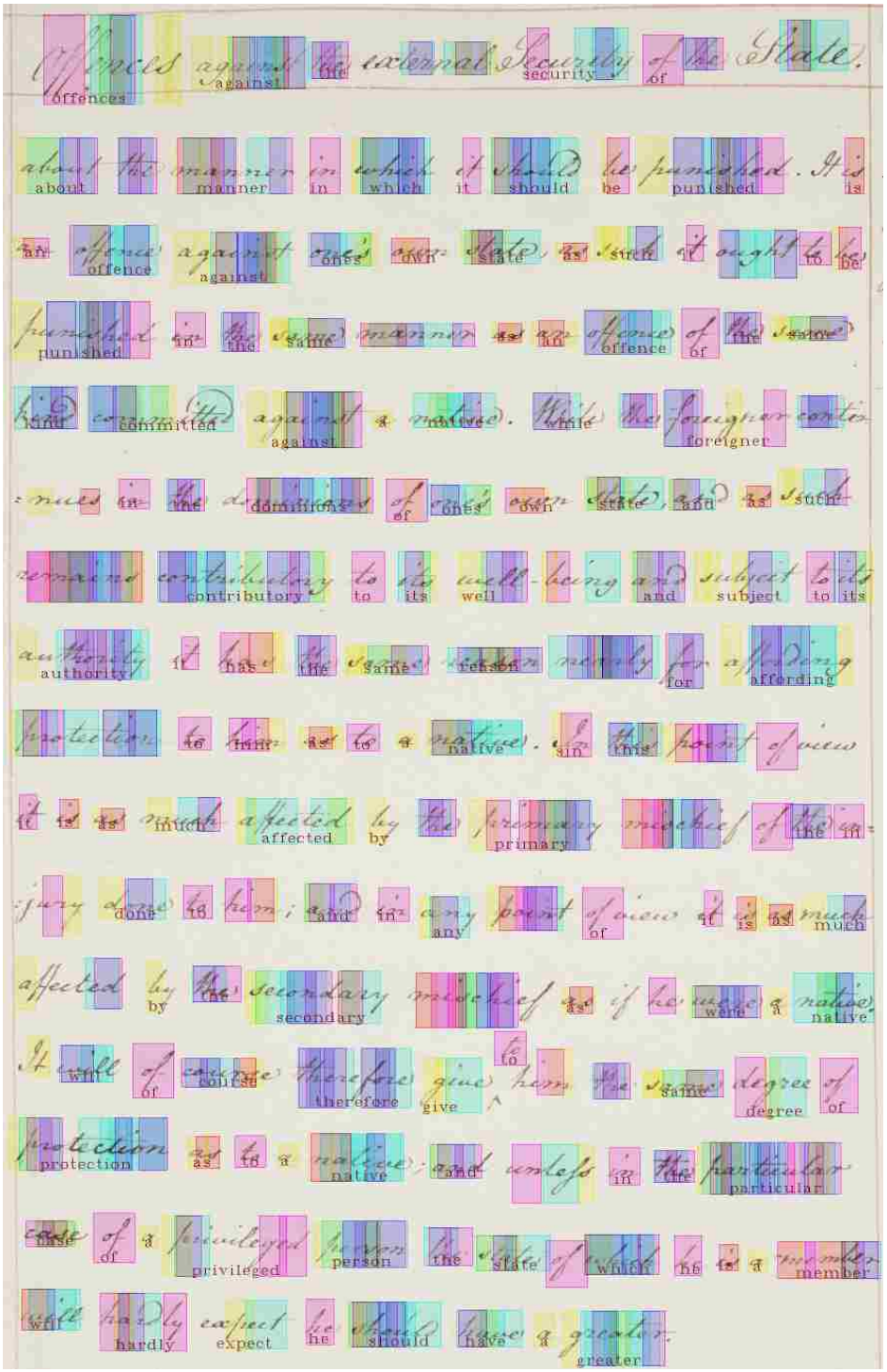


Figure 7.1: A page of the Bentham dataset after running the CATTSS system, with subword spotting approval being distributed using a two-distribution model with only bigrams. Yellow, pink, and cyan boxes represent, respectively, approved unigram, bigram, and trigram spottings. The colors are blended when there is overlap. Text below a word represents the transcription made by the system. We note that some characters missed by the subword spotting tend to follow trends; e.g. "view" is missed twice in this page. This tends to reflect weaknesses of the spotting.Nno trigrams are spotted from "view" and the only bigram is 'ie,' which has poor mAP.

	Bentham			Census Names		
Method	Transcribed	Words/Min	WER	Transcribed	Words/Min	WER
CPV+DTW U	0.570	8.55	0.080	0.325	3.98	0.219
CPV+DTW UB	0.563	8.74	0.082	0.337	4.17	0.219
CPV+DTW UT	0.428	5.89	0.121	0.285915	3.44	0.274
CPV+DTW UBT	0.489	7.11	0.107	0.318	3.89	0.238

Table 7.3: Results from simulations using character probability vectors derived from subword spotting results. The letters U, B and T represent whether unigrams, bigrams and/or trigrams were used.

Chapter 8

Conclusion

In this thesis we have introduced subword spotting and a method of doing it leveraging state-of-the-art word spotting techniques. We have presented results of subword spotting at three granularities (uni-, bi- and trigrams) with two datasets and show it to be a more challenging task than word spotting. We showed three applications of subword spotting: manual transcription assisting, suffix spotting and CAT (computer assisted transcription).

We now summarize what we feel our contributions are with this work and then discuss potential future work.

8.1 Contributions

Our primary contribution is the exploration of subword spotting, a previously unexamined area. We have shown that while word spotting has achieved good results with modern techniques, applying similar techniques to subword spotting does not yield as precise of results. Some insights from our subword spotting experiments that we feel are important are that size estimation of subwords is important if using a sliding window and that the number of characters in a subword and its frequency in the data set does not directly correlate with the precision with which it can be spotted.

A further contribution is our exploration of a few applications of subword spotting. We show how QbE subword spotting can be used to assist people manually transcribing documents by helping locate instances of unknown characters. We feel subword spotting is a good fit for this task as a person may want to search for a group of characters together,

or may not even recognize where character boundaries are. We evaluate the performance of subword spotting when applied to the task of suffix spotting. These searches can achieve results not readily available with word spotting and, as discussed in the next section, can be extended to even more customizable searches. We attempt to apply subword spotting to CAT, but fail to achieve usable results. However, the method has potential given adjustments discussed in the next section.

8.2 Future Work

Several things can potentially improve the subword spotting. We did not perform any preprocessing. State-of-the-art word spotting methods do not deslant word images. However, for subword spotting, deslanting the word images could make localization better as rectangular windows could capture characters without clipping or including adjacent characters. While we attempted to refine the localization of individual spotting results, we did not have success. However, there may be a refinement method we did not try which would succeed.

Everything we did in this work was based on word segmentation. However, it would not be difficult to extend what we have done to a segmentation-free scenario. One would need to be able to distinguish the situation of letters having a space between them, but after this the extension should be straightforward.

While we aggregated spottings into regular expressions, an extension of suffix spotting would be regular expression spotting. That is, the user supplies a regular expression and the system returns all word images which match the expression. With some work, subword spotting should be able to achieve this. One could spot all n-grams in the expression and then use the structure of the expression to aggregate the scores according to their spatial relationships.

Our CAT strategies were not successful, and if they were they would need to be competitive with modern automatic handwriting recognition methods (CNN+RNN) to be applicable. However, there is a slightly different application our method has the potential

to work in. Both our spotting method (PHOCNet) and modern automatic handwriting recognition methods (CNN+RNN) require training data from a corpus before they can begin to work effectively. In a real work-flow this will be a part of the corpus that must be manually transcribed. However, given an effective QbE subword spotting method that does not require corpus specific training, the CATTSS methods described in Chapter 6 should work with a few adjustments. Rather than the system initially spotting a set of n-grams, a user would need to crop a few from the corpus to seed the system. After this, however, it should be able to proceed as outlined in Chapter 6. The key is that the untrained QbE spotting method must be better than the QbS spotting we have demonstrated. Currently such a method does not exist.

Chapter 9

Appendix

	Bentham		Census Names	
n-gram	Spotting width	Predicted width	Spotting width	Predicted width
a	44	39	20	22
b	32	34	40	31
c	44	32	56	40
d	56	46	28	31
e	64	43	20	21
f	32	30	28	28
g	32	30	44	35
h	56	41	44	36
i	96	61	60	38
j	96	62	56	37
k	64	52	40	32
l	44	35	28	24
m	44	49	56	48
n	56	47	20	24
o	32	27	60	40
p	64	51	44	34
q	84	62	60	44
r	32	32	28	23
s	32	32	20	21
t	56	41	44	31
u	64	52	56	40
v	96	64	28	25
w	56	49	56	43
x	96	68	56	38
y	32	31	40	30
z	84	57	60	39
ac	56	57	56	50
ad	56	63	40	47
ai	44	49	60	50
al	64	63	28	38
an	44	58	40	44

ar	44	54	40	41
as	44	53	40	41
at	64	63	60	53
be	44	50	56	48
ca	56	57	44	48
ce	32	43	40	38
ch	44	47	44	49
co	56	53	40	41
ct	32	43	44	42
de	64	61	40	41
di	64	59	28	38
ea	44	51	20	31
ec	44	49	44	42
ed	56	59	28	39
ee	72	61	84	57
el	64	55	28	32
em	84	81	56	56
en	56	58	40	40
er	44	48	40	38
es	32	43	28	33
et	44	45	28	32
fo	56	55	84	63
ge	56	55	40	42
ha	56	60	20	36
he	56	56	40	40
hi	56	54	56	47
ho	32	44	44	45
ic	32	39	84	59
ie	32	41	20	26
il	32	39	60	49
im	44	61	60	57
in	64	60	20	31
io	84	68	28	33
is	32	43	20	28
it	32	41	28	31
la	32	47	20	32
le	56	53	28	34
li	56	49	44	39
ll	56	51	28	33
lo	44	46	20	29
ly	44	51	28	38
ma	96	93	56	60
me	72	79	40	46
mi	84	81	72	65
mo	72	80	40	49

na	84	80	40	46
nc	44	55	60	55
nd	44	61	44	50
ne	72	70	20	32
ng	44	59	56	55
ni	64	64	56	49
no	64	66	20	33
ns	64	63	56	52
nt	32	48	28	37
of	96	77	84	63
ol	56	52	44	41
om	32	58	56	57
on	56	60	28	39
or	32	45	28	33
ot	44	48	20	29
ou	32	47	84	62
ow	84	76	84	67
pa	32	51	56	50
pe	32	49	60	53
po	44	56	28	38
pr	44	54	44	45
ra	64	60	28	37
re	56	54	20	28
ri	56	52	20	27
ro	44	49	56	47
rs	44	50	20	30
rt	44	46	20	28
se	32	45	40	37
sh	56	55	56	50
si	32	41	56	44
so	44	49	20	30
ss	72	66	44	43
st	32	45	28	36
ta	44	51	20	31
te	56	53	60	48
th	44	50	56	51
ti	32	39	56	45
to	72	64	40	37
tr	32	42	20	28
ts	44	49	28	34
ul	32	48	28	38
un	56	63	44	48
ur	32	49	40	41
us	44	56	72	59
ut	32	48	20	31

ve	32	47	20	30
wa	64	70	40	47
we	64	70	20	36
wh	72	76	20	41
wi	64	68	28	39
abl	132	109	40	55
abo	64	77	68	69
ace	72	79	68	65
ach	96	92	84	79
act	72	79	68	66
ade	84	87	68	70
age	108	99	40	54
ain	96	96	56	59
ake	108	101	56	62
ali	64	72	56	57
all	132	104	72	69
als	64	77	68	65
ame	108	116	96	90
anc	84	90	72	73
and	84	98	44	64
ang	84	92	72	75
ans	84	94	84	78
ant	64	80	84	79
any	72	90	72	73
app	108	108	68	71
ard	84	88	40	56
are	84	83	60	60
ari	64	73	56	55
art	96	91	84	74
ary	72	83	56	59
ase	72	79	68	64
ass	108	99	68	67
ast	84	87	68	64
ate	96	90	60	61
ati	64	74	60	60
att	64	74	84	72
ave	84	87	96	82
ber	84	82	72	67
ble	64	73	96	81
but	84	90	68	68
cal	84	83	68	67
can	64	80	72	73
cat	64	73	72	69
cen	64	76	40	54
ces	64	71	60	63

cha	84	84	72	75
che	72	76	40	54
chi	108	94	68	66
cia	108	93	68	64
com	108	106	72	78
con	64	76	40	55
cou	64	75	68	67
cti	64	67	60	59
ded	84	93	72	73
den	64	82	40	57
der	72	82	40	52
des	64	77	68	67
din	108	100	40	56
dis	72	79	68	66
duc	132	120	72	74
ead	64	79	68	70
ear	64	75	44	52
eas	64	75	68	64
eat	72	80	84	70
eco	84	78	60	61
ect	84	81	60	61
een	84	86	96	81
eir	84	77	60	55
ell	84	80	84	72
ely	84	84	60	62
eme	132	116	68	72
enc	84	86	60	66
end	84	94	68	72
ene	72	80	56	59
ens	84	90	68	67
ent	84	86	68	64
era	64	77	60	61
ere	72	75	56	52
eri	84	77	96	75
ern	108	101	84	71
ers	72	78	56	54
ert	64	71	40	47
ery	108	95	60	61
ese	64	71	60	58
ess	84	81	68	64
est	72	75	72	64
eve	96	91	40	48
exp	132	117	60	62
fer	64	72	68	64
ffe	108	98	68	70

fic	96	84	68	66
for	64	72	96	80
fro	84	80	68	65
gen	96	96	68	70
ght	84	86	68	70
gra	64	78	84	78
gre	72	80	96	81
had	84	92	72	77
han	84	93	72	77
har	160	129	56	63
has	64	78	68	71
hat	72	80	60	65
hav	96	96	68	72
hea	84	88	56	62
hei	160	122	40	50
hem	72	92	72	79
hen	108	103	96	82
her	96	87	84	73
hes	160	118	44	57
hey	64	76	56	60
hic	160	116	68	66
hin	160	123	40	55
his	72	76	68	64
hou	84	86	84	75
how	64	83	84	84
ial	64	72	60	61
ica	84	81	68	64
ice	64	67	72	64
ich	64	70	40	53
ide	72	77	60	63
ien	96	88	60	62
ies	64	69	60	57
igh	64	74	72	70
ill	64	68	84	71
ime	96	102	68	70
imp	108	110	72	76
ina	84	88	56	59
inc	84	86	60	63
ind	84	90	44	60
ine	64	74	40	50
ing	64	78	44	58
ini	84	84	96	77
ins	64	76	44	56
int	72	80	40	50
ion	84	85	60	63

ire	64	69	60	55
ish	84	82	68	64
ist	64	69	60	58
ite	64	68	60	55
ith	64	70	56	58
iti	108	88	56	52
its	64	69	60	58
ity	72	76	60	60
ive	72	77	40	47
kin	108	106	96	85
lan	72	86	56	61
lar	64	75	68	65
lat	64	74	40	51
lea	72	80	96	80
led	64	75	40	53
les	72	75	40	49
lic	64	67	96	75
lin	96	88	56	59
lit	64	68	40	46
lle	132	106	40	48
lly	132	108	68	64
low	96	99	44	60
man	96	110	96	91
mar	64	91	84	83
mat	84	102	96	91
men	160	142	72	79
mer	84	97	96	87
min	108	110	84	81
mon	108	113	44	67
mor	84	95	40	58
mpl	108	114	72	79
nal	96	98	72	73
nat	96	96	40	55
nce	84	88	40	54
nde	132	118	84	79
ndi	160	126	68	70
ned	108	108	68	72
ner	72	83	72	67
nes	96	96	56	59
nge	72	86	68	70
nin	132	119	96	85
not	84	89	68	65
now	96	105	72	76
nsi	72	80	68	64
nst	84	86	68	67

nta	84	92	68	69
nte	64	78	84	71
nti	160	120	60	62
ntr	64	77	68	64
nts	84	86	68	67
ome	84	98	72	73
omm	160	146	84	90
omp	132	126	72	79
ona	84	93	60	65
ond	64	82	72	73
one	72	83	84	74
ong	64	82	68	71
ons	64	78	68	67
ont	64	77	40	52
ope	72	80	40	50
ord	96	95	44	55
ore	72	77	60	59
orm	84	99	72	74
ort	64	73	60	58
ose	64	72	60	59
ost	64	74	60	59
oth	64	75	96	82
oug	64	79	40	55
oul	64	77	68	64
oun	132	120	56	63
our	72	80	60	63
ous	64	77	44	56
out	64	75	60	63
ove	64	74	60	61
owe	160	131	56	63
own	132	121	84	84
par	84	89	72	69
pec	96	91	68	65
per	84	85	72	66
pla	108	102	68	67
ple	108	96	40	50
por	84	89	60	63
pos	72	82	56	58
pre	72	81	40	50
pri	64	77	40	49
pro	84	85	60	61
rac	96	88	68	66
ral	96	91	56	57
ran	72	85	96	85
rat	72	79	60	62

rea	64	75	56	58
rec	84	80	60	61
red	96	90	40	52
ree	108	99	60	56
rel	132	101	96	73
ren	96	93	44	53
res	84	82	68	62
ric	160	120	72	64
rie	108	91	40	45
rin	160	119	84	74
ris	132	102	40	47
rit	96	87	84	66
rom	108	107	72	74
rou	64	76	60	63
rti	72	73	60	55
sed	84	89	68	67
see	108	95	72	66
sel	84	79	84	73
sen	64	78	84	75
ser	72	76	96	74
ses	64	75	60	61
she	72	78	68	66
sho	84	82	68	67
sin	72	82	68	64
sio	64	70	60	58
sit	64	69	60	58
som	96	102	40	60
son	84	90	96	82
spe	64	77	56	59
sse	132	107	60	61
ssi	132	111	60	59
sta	72	79	96	78
ste	108	93	96	74
sti	72	75	44	51
sto	84	82	44	51
str	64	72	56	55
sur	64	77	40	52
tai	84	82	60	60
tal	72	80	40	51
tan	64	80	68	69
tat	72	80	60	62
ted	64	75	68	65
ten	72	80	68	64
ter	64	71	44	51
tes	72	77	60	59

tha	72	80	68	69
the	64	72	68	65
thi	72	74	60	62
tho	72	77	72	68
tic	64	67	60	59
tim	84	94	68	71
tin	64	74	40	50
tio	132	98	60	56
tiv	108	93	60	58
tor	64	71	60	58
tra	84	87	40	50
tri	64	71	56	54
tte	132	110	96	77
tur	108	96	60	62
ual	84	91	68	67
uch	64	77	72	73
ugh	64	81	72	77
uld	72	86	68	71
und	96	101	72	77
uni	84	91	68	67
unt	96	95	40	55
ure	96	94	60	61
use	72	80	40	51
ust	64	78	44	56
ven	64	80	72	69
ver	64	74	72	65
was	64	86	72	76
wer	96	95	68	69
whe	84	95	40	59
whi	96	97	72	74
who	72	89	72	77
wil	96	96	72	70
wit	96	92	68	67
wor	108	106	68	70
you	96	93	84	74

Table 9.1: Optimal sliding window widths for spotting and estimated visual widths for each n-gram of interest.

References

[1] D. Aldavert, M. Rusiñol, R. Toledo, and J. Lladós. A study of bag-of-visual-words representations for handwritten keyword spotting. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(3):223–234, 2015.

[2] J. Almazán, D. Fernández, A. Fornés, J. Llados, and E. Valveny. A coarse-to-fine approach for handwritten word spotting in large scale historical documents collection. In *Proceedings of the 13th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2012.

[3] J. Almazán, A. Gordo, A. Fornés, and E. Valveny. Word spotting and recognition with embedded attributes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(12):2552–2566, 2014.

[4] H. Bunke, S. Bengio, and A. Vinciarelli. Offline recognition of unconstrained handwritten texts using hmms and statistical language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 26(6):709–720, 2004.

[5] R. Clawson. Intelligent indexing: A semi-automated, trainable system for field labeling. Master's thesis, Brigham Young University, 2014. URL `http://scholarsarchive.byu.edu/etd/5307/`.

[6] A. Fischer, A. Keller, V. Frinken, and H. Bunke. Lexicon-free handwritten word spotting using character HMMs. *Pattern Recognition Letters*, 33(7):934–942, 2012.

[7] B. Gatos, G. Louloudis, T. Causer, K. Grint, V. Romero, J.A. Sánchez, A.H. Toselli, and E. Vidal. Ground-truth production in the transcriptorium project. In *Proceedings of 11th Interational Workshop on Document Analysis Systems (DAS)*. IEEE, 2014.

[8] A. Graves, S. Fernández, and F. Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*. PMLR, 2006.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37(9):1904–1916, 2015.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[11] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[12] P. Krishnan, K. Dutta, and C.V. Jawahar. Deep feature embedding for accurate recognition and retrieval of handwritten text. In *Proceedings of the 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2016.

[13] Y. Liang, M.C. Fairhurst, and R.M. Guest. A synthesised word approach to word retrieval in handwritten documents. *Pattern Recognition*, 45(12):4225 – 4236, 2012.

[14] R. Manmatha, C. Han, and E.M. Riseman. Word spotting: a new approach to indexing handwriting. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1996.

[15] U. Marti and H. Bunke. The IAM-database: An english sentence database for off-line handwriting recognition. *International Journal on Document Analysis and Recognition (IJDAR)*, 5:39–46, 2002.

[16] U.V. Marti and H. Bunke. Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition system. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 14(1):65–90, 2001.

[17] C. Neudecker and A. Tzadok. User collaboration for improving access to historical texts. *Liber Quarterly*, 20(1):119–128, 2010.

[18] J. Puigcerver. Are multidimensional recurrent layers really necessary for handwritten text recognition? In *Proceedings of the 14th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2017.

[19] T.M. Rath and R. Manmatha. Features for word spotting in historical manuscripts. In *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2003.

[20] G. Retsinas, B. Gatos, A. Antonacopoulos, G. Louloudis, and N. Stamatopoulos. Historical typewritten document recognition using minimal user interaction. In *Proceedings of the 3rd International Workshop on Historical Document Imaging and Processing (HIP)*. ACM, 2015.

[21] G. Retsinas, G. Sfikas, and B. Gatos. Transferable deep features for keyword spotting. In *Proceedings of the 6th International Workshop on Computational Intelligence for Multimedia Understanding (IWCIM)*. IEEE, 2017.

[22] J.A. Rodrıguez. and F. Perronnin. Local gradient histogram features for word spotting in unconstrained handwritten documents. In *Proceedings of the 11th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2008.

[23] V. Romero, A.H. Toselli, L. Rodriguez, and E. Vidal. Using mouse feedback in computer assisted transcription of handwritten text images. In *Proceedings of the 10th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2009.

[24] L. Rothacker, M. Rusiñol, and G.A. Fink. Bag-of-features hmms for segmentation-free word spotting in handwritten documents. In *Proceedings of the 12th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2013.

[25] J. A. Sánchez, V. Romero, A. H. Toselli, and E. Vidal. Icfhr2016 competition on handwritten text recognition on the read dataset. In *Proceedings of the 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2016.

[26] J.A. Sánchez, V. Romero, A.H. Toselli, M. Villegas, and E. Vidal. Icdar2017 competition on handwritten text recognition on the read dataset. In *Proceedings of the 14th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2017.

[27] N. Serrano, A. Giménez, A. Sanchis, and A. Juan. Active learning strategies for handwritten text transcription. In *Proceedings of the 12th International Conference on Multimodal Interfaces and the 7th Workshop on Machine Learning for Multimodal Interaction (ICMI-MLMI)*. ACM, 2010.

[28] N. Serrano, A. Giménez, J. Civera, A. Sanchis, and A. Juan. Interactive handwriting recognition with limited user effort. *International Journal on Document Analysis and Recognition (IJDAR)*, 17(1):47–59, 2014.

[29] R. Shekhar and C.V. Jawahar. Word image retrieval using bag of visual words. In *Proceedings of 10th Interational Workshop on Document Analysis Systems (DAS)*. IEEE, 2012.

[30] S. Sudholt and G.A. Fink. Phocnet: A deep convolutional neural network for word spotting in handwritten documents. In *Proceedings of the 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2016.

[31] S. Sudholt and G.A. Fink. Evaluating word string embeddings and loss functions for cnn-based word spotting. In *Proceedings of the 14th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2017.

[32] A. Toselli, V. Romero, M. Pastor, , and E. Vidal. Multimodal interactive transcription of text images. *Pattern Recognition*, 43(5):1814–1825, 2010.

[33] A.H. Toselli, V. Romero, L. Rodriguez, and E. Vidal. Computer assisted transcription of handwritten text images. In *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2007.

[34] A.H. Toselli, V. Romero, and E. Vidal. Computer assisted transcription of text images and multimodal interaction. In *Machine Learning for Multimodal Interaction*, volume 5237 of *Lecture Notes in Computer Science*, pages 296–308. Springer, 2008.

[35] C. Wigington, S. Stewart, B. Davis, and W. Barrett. Data augmentation for recognition of handwritten words and lines using a cnn-lstm network. In *Proceedings of the 14th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2017.

[36] T. Wilkinson, J. Lindström, and A. Brun. Neural ctrl-f: Segmentation-free query-by-string word spotting in handwritten manuscript collections. *arXiv preprint arXiv:1703.07645*, 2017.

[37] K. Zagoris, I. Pratikakis, and B. Gatos. A framework for efficient transcription of historical documents using keyword spotting. In *Proceedings of the 3rd International Workshop on Historical Document Imaging and Processing (HIP)*. ACM, 2015.