Inkball Models for Character Localization and Out-of-Vocabulary Word Spotting

Nicholas R. Howe
Department of Computer Science
Smith College
Northampton, Massachusetts
Email: nhowe@smith.edu

***Abstract*—Inkball models have previously been used for keyword spotting under the whole word query-by-image paradigm. This paper applies inkball methods to string-based queries for the first time, using synthetic models composed from individual characters. A hybrid system using both query-by-string for unknown words and query-by-example for known words outperforms either approach by itself on the George Washington and Parzival test sets. In addition, inkball character models offer an explanatory tool for understanding handwritten markings. In combination with a transcript they can help to to attribute each ink pixel of a word image to specific letters, resulting in high-quality character segmentations.**

I. Introduction

Around the world, many collections of historical handwritten documents cannot be utilized to their full extent due to limited availability of human annotation and transcription services. This observation motivates research into computational algorithms to make sense of handwritten documents automatically or semi-automatically, thus liberating human expertise for higher-level work. Through pattern recognition such algorithms interpret the markings on a document to infer meaning. One strategy for recognition is *word spotting*, which enables information retrieval by identifying appearances of a query word within a collection. In some instances word spotting can be more tractable than full-blown transcription, yet it allows similar search applications. To succeed at word spotting, an algorithm must recognize an arbitrary word whenever and wherever it appears in the text.

Recognition algorithms divide into *discriminative* algorithms, which seek to directly extract meaning from observations, and *generative* algorithms, which seek to estimate and model underlying factors that generate the observations. Given sufficient training data, discriminative models usually offer higher recognition accuracy [1], yet their focus on direct recognition means that they offer little insight into the cause of their predictions. For example, methods using histograms of oriented gradient (HOG) features [2] or scale-invariant (SIFT) features [3] have been shown to perform well in word-spotting applications. However, the computation and comparison of vectors of HOG features does not readily show why one particular pattern matches a query better than some other. In contrast, generative algorithms model the causal basis of symbol production, and can more easily identify ways in which a particular observation deviates from the expected ideal. Inkball models [4] are a class of generative model for handwriting that have shown good results in word spotting tasks [5]. In these models, character shapes are formed from overlapping disks of ink arranged in specific spatial configurations so as to create the necessary appearance. The likelihood of an observed ink pattern being associated with any underlying model can be quickly computed based upon the distortion that must be imposed upon the model's normal configuration in order to match the observed distribution of ink. Promising instances of a query word correspond to local maxima of the configuration likelihood. Furthermore, matches also give the locations of individual parts of the query word within the target, since the configuration embodies this information.

This paper seeks to extend the explanatory potential of inkball models, and explore the applications enabled by them. While such models have previously been used for spotting full words, the method proposed herein develops models for individual characters and their spatial relationships. This approach naturally leads to a string-based word spotting algorithm. It also offers the potential to support semi-automatic generation of training data for more successful discriminative character recognition, although this application is left for future work.

The next section considers prior results related to this work. The section following describes the methods for character recognition and attribution, and for string-based word spotting. Section IV describes some experimental results, and the final section offers concluding remarks.

II. Prior Work

Howe [5] introduces spring-connected inkball models for word spotting, giving both an energy function and an algorithm for quickly computing minimal-energy configurations given a document image. Each model comprises a set of nodes $Q = \{q_i | 1 \leq i \leq m\}$ arranged in a tree structure with root q_1, and rest offset vectors $\{\vec{t_i} | 2 \leq i \leq m\}$, each $\vec{t_i}$ specifying the 2d position of q_i relative to its parent in the tree, $q_{i\uparrow}$. The expression for the energy includes two terms: E_ξ describing the deformation of the model nodes away from their rest offsets, and E_ω capturing the distance from each node to the nearest medial axis of ink markings in the document, computed from observations I.

$$E(Q, C, I) = E_\xi(Q, C) + E_\omega(C, I) \tag{1}$$

If c_j is the offset of node q_i relative to its parent in a given configuration C, then the deformation term is a sum of squares comparing c_j with t_j.

$$E_\xi(Q,C) = \sum_{j=2}^{m} \frac{\|c_j - t_j\|^2}{2\sigma_j^2} \quad (2)$$

Likewise, the observation term is also a sum of squares, where $\chi(S_I, \vec{v_i})$ denotes the distance between the point v_j and the nearest pixel of the ink medial axis, S_I.

$$E_\omega(C,I) = \sum_{i=1}^{m} \chi(S_I, \vec{v_i}) \quad (3)$$

An efficient dynamic programming algorithm using distance transforms finds the minimal energy and its corresponding configuration for every root position across the observation plane [5]. Matches are detected when the local minimum of the energy falls below a threshold. This process naturally gives rise to a *query by example* (QBE) algorithm: given a query image, build a model by placing disk centers at equally-spaced locations on the medial axis of the ink, then link nearest neighbors into a tree structure with its root at the center of mass. Matches to the model give the word locations in the target.

While much recent work has used the QBE paradigm for word spotting [5], [2], [3], less attention has been paid to *query by string* (QBS), where the search prompt is simple text represented in ASCII or unicode. Besides its convenience, QBS implies the potential to search for *out of vocabulary* (OOV) words, which are words that have never been observed by the algorithm. Some recent work has looked at QBS methods. Aldavert et al. use a bag-of-visual-words technique [6]. Almazan et al. approach the task by mapping both images and strings into a combined representation space such that related entities appear near each other [7], [8]. Roy et al. describe a system capable of searching for OOV words based on character n-grams [9]. While the latter strategy could be used to augment the work presented herein, this paper takes the simpler but more universal approach of using individual characters as building blocks.

Other work has looked at graph-based algorithms for document analysis, particularly Bunke & Reisen [10]. Peng Wang et al. use a graph that characterizes points of the handwriting skeleton using shape context rather than inkballs [11].

III. Methods

Character recognition works in essentially the same way as word recognition: given an exemplar of the character form, the inkball model derived from it can identify similar patterns in previously unlabeled content. However, a single character is both shorter and less distinctive than an entire word. Some letters or letter sequences are quite similar in form: compare 'cl' and 'd', for example. Unsurprisingly, a generative model like the inkball cannot compete in accuracy on this task with the top discriminative character recognition methods.

However, character inkball models perform well for a related but easier task: the localization of a known character string in a word image, or conversely the attribution of ink pixels to individual letters. Essentially a form of guided character segmentation, this task is nontrivial in handwritten manuscripts with connected writing and overlap between characters. The individual character samples generated by this process could be used for many purposes, including as training examples for discriminative character recognition. Since word labels are generally much easier to come by than individual character segmentations, this approach may offer a clever way to augment character training sets semi-automatically.

Localizing all the characters in a training corpus of word images also provides other benefits. One can easily gather statistics on typical spatial configurations between pairs of characters, typical shapes of letters, etc. For example this includes the horizontal separation between characters in languages using a Latin alphabet or equivalent. Where necessary it can also include vertical offsets as well. These measurements in turn form the basis for QBS search: given a character string as query, one can construct an inkball model for the word using an appropriate sequence of individual character models spaced according to the typical separations observed in the training corpus.

A. Character Localization

Like word spotting, character localization uses a formulation based upon energy minimization, where the energy in question can be conceptualized as something like a negative log posterior probability. The energy formula balances a number of competing considerations, each described by its own subterm. Notationally, let $\ell_1, \ell_2, ...\ell_n$ represent the characters of the (known) word string, $\mathcal{Q} = (Q_{\ell_1}, Q_{\ell_2}, ...Q_{\ell_n})$ the corresponding inkball models, $\mathcal{C} = (C_1, C_2, ...C_n)$ their distinct configurations, and B a binarized version of word image I. The energy is a combination of continuity terms in x, y, and scale, plus a coverage term representing the likelihood that the configuration set can explain the observations B. Each subterm in the equation below is described further in the following text.

$$\mathcal{E}(\mathcal{Q}, \mathcal{C}, B) = \lambda_x E_X + \lambda_y E_Y + \lambda_s E_S + \lambda_p E_P \quad (4)$$

The character displacement energies E_X and E_Y measure how well the relative displacement between the root nodes of adjacent character models $\Delta_x(C_i, C_{i+1})$ and $\Delta_y(C_i, C_{i+1})$ match the expected displacements $\Delta_x^E(\ell_i, \ell_{i+1})$ and $\Delta_y^E(\ell_i, \ell_{i+1})$. For left-to-right text, the vertical displacement energy takes a simple Gaussian form, while the horizontal displacement is modeled by a Poisson distribution (represented by $\mathcal{P}$ below), which disallows negative values.

$$E_X = \sum_{i=2}^{n} \mathcal{P}(\Delta_x(C_i, C_{i+1}), \Delta_x^E(\ell_i, \ell_{i+1})) \quad (5)$$

$$E_Y = \sum_{i=2}^{n} \frac{\|\Delta_y(C_i, C_{i+1}) - \Delta_y^E(\ell_i, \ell_{i+1})\|^2}{2\sigma_y^2} \quad (6)$$

The expected displacements should be scaled to match the image resolution. They may come from measurements on a training set if available, or from simple heuristics otherwise. For example, $\Delta_y^E(\ell_i, \ell_{i+1}) = 0$ for English and related

languages, while $\Delta_x^E(\ell_i, \ell_{i+1}) = w/(n+1)$ where W is the width of the word image.

The scale energy E_S accounts for cases where letters in a word (or expression) may be of different sizes. This difference can be captured by replacing Q_i with $\Phi(Q_i, s_i)$, where all the default offsets in Q_i are scaled by the multiplicative factor s_i. In this formulation s_i becomes an additional piece of configuration data, and $\Phi(Q_i, 1)$ is an identity for the original Q_i. The scale match energy penalizes mismatches in scale according to their severity. Note that the form used below presumes that all characters should be of the same size, an assumption that might not apply in all applications.

$$E_S = \sum_{i=1}^{n-1} \frac{\|s_i - s_{i+1}\|^2}{2\sigma_s^2} \tag{7}$$

The pixel coverage energy E_P measures how well the proposed configurations explain B^+, the set of the foreground pixels of B. Each poorly explained pixel contributes a fixed amount toward the energy; to lower its energy, a pixel must be attributed to exactly one character model Q_{ℓ_i} as rendered in its configuration C_i (see below). In this case, its energy contribution is discounted by $1 - e^{-E_i/\alpha}$ where E_i is the character model fit energy from Equation 1 and α is a soft threshold. Denote the rendering of Q_{ℓ_i} as R_i, its foreground pixels as R_i^+, and the union of the foreground pixels of all the other character renderings as $\bar{R}_i^+ = \bigcup_{j \in 1..n \setminus i} R_i^+$.

$$E_P = \frac{1}{\rho h^2} \left[\|B^+\| - \sum_{i=1}^{n} \|(R_i^+ \cap B^+) \setminus \bar{R}_i^+\| e^{-E_i/\alpha} \right] \tag{8}$$

The normalization factor of $1/\rho h^2$ provides rough independence from image resolution. Here h is the height of the cropped word image B and ρ is an estimate of ink density, set at 0.15.

Rendering proceeds as follows: Place a disk of ink at the location of each node specified in the configuration. The radius of each disk is taken from the initial sample that generated the model, as recorded during its creation. The more densely sampled the model, the more the disks will overlap and the higher the quality of the rendering.

In practice, problematic overlap usually occurs only between adjacent letters. Therefore it is more efficient to compute a modified E_p that accounts just for neighbor overlap. To simplify the notation below, let $R_0^+ = R_{n+1}^+ = \emptyset$.

$$E'_P = \frac{1}{\rho h^2} \left[\|B^+\| - \sum_{i=1}^{n} \|(R_i^+ \cap B^+) \setminus R_{i-1}^+ \setminus R_{i+1}^+\| e^{-E_i/\alpha} \right] \tag{9}$$

Armed with the energy function defined above, a search procedure finds the set of configurations $\mathcal{C}$ that minimize it. The first step computes the minimal configuration energies for all Q_{ℓ_i} at every root position, using the dynamic programming technique described in prior work [5]. The local minima of these energies across root position and scale (typically fewer than one dozen per character) are taken as configurations worthy of further consideration. A second dynamic programming step (essentially the Viterbi algorithm) selects from these candidates an appropriate sequence of configurations that matches the known word label and minimizes Equation 4. Section IV-A describes experimental results for this technique.

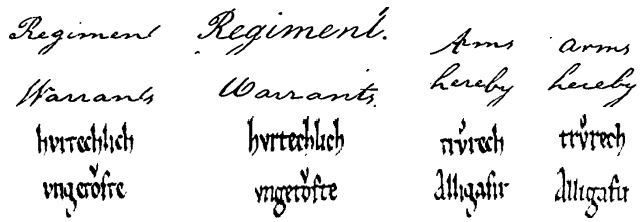

Fig. 1. Sample synthetic query images (left) vs. real test images (right)

B. String-Based Search

Character localizations inferred using the technique described above help to enable string-based search. Given hard data on the typical displacement between adjacent letters plus one or more examples of each letter form, a text query converts easily to a word image or inkball model.

A lack of observational data for every character bigram presents one obstacle to this plan. Some character sequences occur frequently, but others are rare and thus may not be observed within a specific training set. Nevertheless the QBS application requires an estimate of the separation for all character transitions. In some cases queries may even include characters not represented in the training set at all; this is a more difficult problem but is fortunately less common.

Appropriate separation estimates for unobserved bigrams can be filled in via smoothing, by analogy to bigram frequency estimation in language models. Begin with computed character locations on training data. Bin the displacements between adjacent characters according to their character bigram. All measurements for *aa* go into one bin, all for *ab* into a second, and so on. Some bins will likely still be empty at this point, so in every bin also include the mean displacement over all measurements. The typical displacement for a particular bigram is then taken to be the median over any measurements in the corresponding bin. The experiments in the next section follow this procedure, also omitting from the computation the words with per-character energy above the 90th percentile. The results in Figures 3-4 suggest that this precaution may be more conservative than necessary.

Displacement information plus a set of individual character appearance models give a synthetic rendering for any query string. The experiments in the next section use a set of character samples identified by hand. To render, arrange the samples according to their order of appearance and the values in the displacement table. Although a word image can be generated in this manner, in practice inkball models are produced directly from the models of the sample characters plus displacements.

Figure 1 shows some sample synthetic query images. Imperfections may occur in the connections between adjacent letters, but on the whole they look natural enough. Alternate forms for a single character sometimes occur, and can be handled by generating multiple query images. However, this may not be necessary for good performance: all experiments in this paper simply use the most common form.

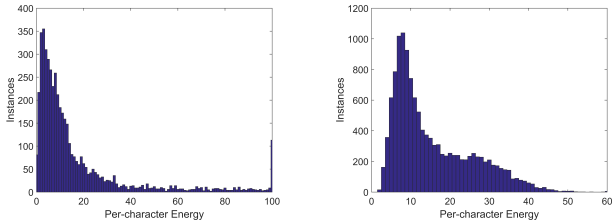

Fig. 2. Histogram of character fit energy for all training words in GW20 (left) and Parzival (right).

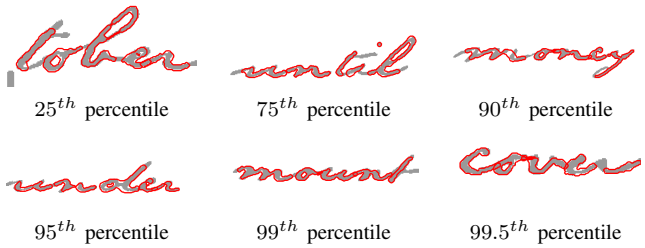

Fig. 3. Sample character fittings from GW20, sampled according to total character fit energy.

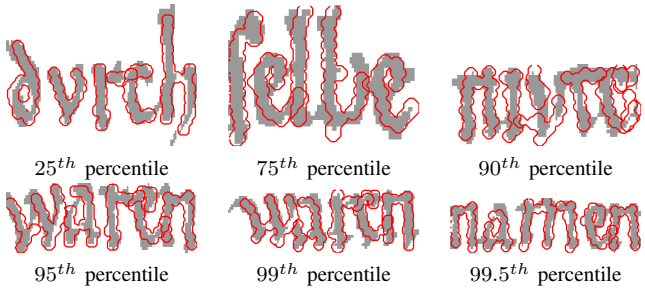

Fig. 4. Sample character fittings from Parzival, sampled according to total character fit energy.

IV. Experiments

The experiments employ two data sets commonly used for testing word spotting algorithms: 20 pages from the letters of George Washington (GW20) [12], and 47 pages from a text in medieval German (Parzival) [13]. Both have predefined word segmentations, and the Parzival images as distributed have already been preprocessed for binarization and deslanting. The GW20 set uses a four-fold separation of words, while the Parzival set has predefined training, test, and validation sets. Since the inkball model techniques have no need for validation, the latter is combined with the training set here.

A. Character Localization

The first set of experiments evaluates the character localization algorithm. No ground truth data are available for this task in either GW20 or Parzival. However, visual inspection shows that the results are quite good. Figure 2 shows histograms of the per-character energy for all words in the training set on each corpus. Figures 3-4 show examples drawn from the distribution at various points. The word image is shown in gray, and the fitted characters are a red outline. Even at the 90^{th} percentile and above most results avoid serious errors.

Once character models are fitted to a word image, they can be used to attribute the pixels in that image toward the particular character they belong to. This can be useful in applications where multiple realistic examples of a character would be useful, and annotation by hand is too slow or too expensive. Figure 5 shows an example.

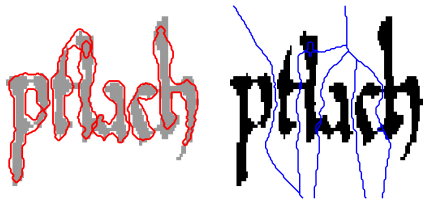

Fig. 5. Example of pixel attribution. Left shows character fit, right shows attribution boundaries.

The attribution process is not quite so simple as matching with the rendered character models, since characters may not overlap perfectly, and occasionally multiple models may overlap at a pixel. The adopted procedure thus follows a set of heuristics that produce usable results in most cases.

- Connected ink components that overlap in whole or in part with only one rendered letter model R_i are attributed entirely to that letter.
- Components that overlap with no rendered letter models are attributed to noise.
- Components that overlap with multiple rendered letter models are attributed pixel by pixel to whichever model is nearest in Euclidean distance. Thus where two or more B_i overlap at a pixel (distance 0), that pixel will be attributed to each of the corresponding characters.

B. Query By String

The string-based query experiments adopt the algorithm described in Section III-B. Prior work on QBE word spotting could use only the subset of words common to both the training and test sets [5], since by definition there are no examples available for words outside the training vocabulary. Figure 6 shows the performance of the QBS algorithm from this paper in comparison on the same limited set.[1] As might be expected, the mean average precision is lower, but still useful for many purposes: for GW20 the numbers are 54.9% vs. 75.6%, with most of the discrepancy coming at higher recalls, and for Parzival the mean average precision (mAP) is 59.6% vs. 79.6%, with nearly a uniform discrepancy across the curve. The difference between the upper and lower curve in each case represents the penalty paid for QBS convenience.

However, the true strength of the QBS algorithm lies in its ability to search for OOV terms, which the QBE algorithms cannot handle at all. Figure 7 shows the performance on OOV words only. The mean average precision on this subset is 48.1% for GW20 and 60.3% for Parzival.

Given that QBE does better on known words, while QBS handles OOV words, a hybrid offers the best performance.

[1] The recall-precision curves and mean average precision shown in this paper differ slightly from the cited work [5], but are based upon the same results. The difference stems from a nonstandard handling of interpolated precision in the prior work, which is corrected in this paper.

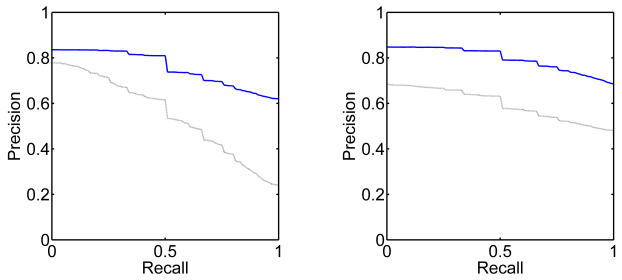

Fig. 6. Precision vs. recall for known words in GW20 (left) and Parzival (right). Solid line uses QBE, dotted line uses QBS.

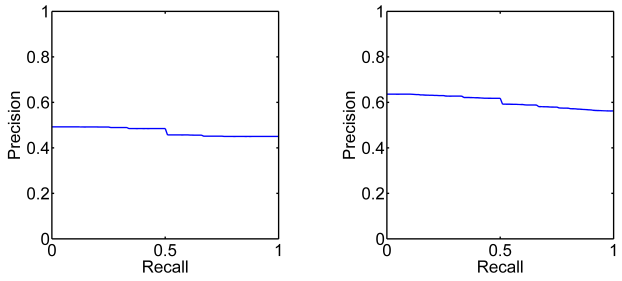

Fig. 7. Precision vs. recall for OOV words in GW20 (left) and Parzival (right). The curves are flat because most OOV words appear in the test set only once.

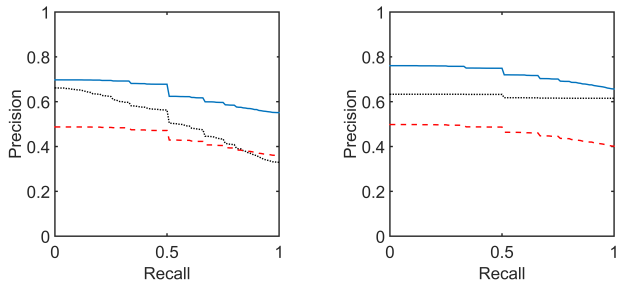

Fig. 8. Precision vs. recall for all words in GW20 (left) and Parzival (right). The solid curve is for the hybrid QBE/QBS approach, the dotted line shows QBS only, and the dashed line shows QBE with background precision on the OOV words.

Figure 8 shows results for such a system in comparison with full QBS and with QBE alone when OOV words are taken into account. The hybrid system reaches a mean average precision over all words of 69.9% for Parzival and 63.9% for GW20, as compared respectively with 60.0% and 51.8% using QBS for all queries, or 46.7% and 44.0% for unassisted QBE. By comparison, Aldavert et al. report mAP of 56.54% on GW20 with OOV words [6], while Almazan et al. report a corresponding mAP of 91.11% [8].

V. Conclusion

Inkball character models provide a tool for understanding details of handwriting in an intuitive fashion. When character models fit to a word image, they not only associate the pixels of that image with an individual character, but with a specific part of that character. This may have implications for stroke recovery and style analysis, a promising subject for future work. In any case, the fit models permit the extraction of further data from a training corpus, such as the character separation data demonstrated here.

The query by string method described herein serves both to demonstrate one of the applications of inkball character models, and as an interesting goal in its own right. String-based search provides better convenience for users, and is more flexible because it can handle out-of-vocabulary terms. The system demonstrated here shows very good performance on the data sets used for testing. Many refinements are possible, for example using multiple interchangeable models of each character, but these are left for future work.

References

[1] A. Ng and M. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes," *Advances in Neural Information Processing Systems*, vol. 14, p. 841, 2002.

[2] J. Almazán, A. Gordo., A. Fornés, and E. Valveny, "Segmentation-free word spotting with exemplar SVMs," *Pattern Recognition*, vol. 47, no. 12, pp. 3967–3978, 2014.

[3] M. Rusiñol, D. Aldavert, R. Toledo, and J. Lladós, "Efficient segmentation-free keyword spotting in historical document collections," *Pattern Recognition*, vol. 48, no. 2, pp. 545–555, 2015.

[4] M. Revow, C. Williams, and G. Hinton, "Using generative models for handwritten digit recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 18, no. 6, pp. 592–606, 1996.

[5] N. Howe, "Part-structured inkball models for one-shot handwritten word spotting," in *International Conference on Document Analysis and Recognition*, 2013.

[6] D. Aldavert, M. Rusiñol, R. Toledo, and J. Lladós, "Integrating visual and textual cues for query-by-string word spotting," in *International Conference on Document Analysis and Recognition*, 2013, pp. 511–515.

[7] J. Almazán, A. Gordo., A. Fornés, and E. Valveny, "Handwritten word spotting with corrected attributes," in *International Conference on Computer Vision*, 2013.

[8] J. Almazán, A. Gordo, A. Fornés, and E. Valveny, "Word spotting and recognition with embedded attributes," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 36, no. 12, pp. 2552–2566, 2014.

[9] U. Roy, N. Sankaran, K. P. Sankar, and C. V. Jawahar, "Character n-gram spotting on handwritten documents using weakly-supervised segmentation," in *International Conference on Document Analysis and Recognition*, 2013.

[10] H. Bunke and K. Riesen, "Recent advances in graph-based pattern recognition with applications in document analysis," *Pattern Recognition*, vol. 44, no. 5, pp. 1057–1067, 2011.

[11] P. Wang, V. Eglin, C. Garcia, C. Largeron, J. Llados, and A. Fornes, "A coarse-to-fine word spotting approach for historical handwritten documents based on graph embedding and graph edit distance," in *International Conference on Pattern Recognition*, 2014, pp. 3074–3079.

[12] V. Lavrenko, T. Rath, and R. Manmatha, "Holistic word recognition for handwritten historical documents," in *Proc. of the IEEE Workshop on Document and Image Analysis for Libraries DIAL'04*, 2004, pp. 278–287.

[13] A. Fischer, M. Wüthrich, M. Liwicki, V. Frinken, H. Bunke, G. Viehhauser, and M. Stolz, "Automatic transcription of handwritten medieval documents," in *Proc. 15th Int. Conf. on Virtual Systems and Multimedia*, 2009, pp. 137–142.