Inkball Models as Features for Handwriting Recognition

Nicholas R. Howe
Smith College
Northampton, Massachusetts, USA
Email: nhowe@smith.edu

Andreas Fischer
University of Fribourg
1700 Fribourg, Switzerland
Email: andreas.fischer@unifr.ch

Baptiste Wicht
University of Fribourg
1700 Fribourg, Switzerland
Email: baptiste.wicht@unifr.ch

***Abstract*—Inkball models provide a tool for matching and comparison of spatially structured markings such as handwritten characters and words. Hidden Markov models offer a framework for decoding a stream of text in terms of the most likely sequence of causal states. Prior work with HMM has relied on observation of features that are correlated with underlying characters, without modeling them directly. This paper proposes to use the results of inkball-based character matching as a feature set input directly to the HMM. Experiments indicate that this technique outperforms other tested methods at handwritten word recognition on a common benchmark when applied without normalization or text deslanting.**

***Keywords*-Image processing; Image recognition; Optical character recognition software;**

I. Introduction

Handwritten manuscripts cannot fully enter the digital world without reliable methods for machine transcription. Modern modes of scholarship rely upon search and other transcript-enabled functionality, yet transcription by human effort consumes too many resources for wide application. This observation motivates research to improve the accuracy and reliability of algorithms for automatic recognition of handwritten text.

Handwriting recognition refers to the process of inferring a writer's intended communication based upon observation of the markings made on a document. The Hidden Markov Model (HMM) has proven a key tool in this task because it explicitly models the production of observational data based upon a sequence of symbolic hidden states. Although an HMM can be applied to any set of observations, intuition suggests that features highly correlated with the proposed set of hidden states will prove more successful than features with low correlation.

This paper explores the use of inkball models to generate a spatially varying feature set that is then used to train an HMM for character recognition. Hidden states in the HMM correspond to characters in the target language, and each character has a corresponding inkball model generated from a protoype of that character. Fit scores for each of the character models form the feature set that is input to the HMM at each sequential position.

A. Related Work

Inkball models were previously introduced in word spotting applications [1], and were later used to model individual character prototypes [2], [3]. This paper adapts the latter approach by using fit scores of the character prototypes as input to an HMM. The suggested method is related to the framework of graph similarity features [4] where graph-based representations of character prototypes are used to compute fit scores for HMM-based recognition.

HMM are a standard statistical framework for sequence recognition tasks. Originally introduced for speech [5], they have also been widely used for handwriting recognition [6] based on different handwriting features. Well-known examples include Marti and Bunke's geometric features [7] and the SIFT-like gradient features proposed by Rodriguez and Perronin [8] and Terasawa and Tanaka [9], respectively. In this paper, they will serve as references for experimental comparison with the proposed inkball features.

II. Method

Prior work on inkball models has presented them in a formal mathematical style [1]. This paper adopts a more operational description, although the method and implementation remains the same. Each feature fed to the HMM corresponds to the match fitness of an inkball model representing one character prototype, evaluated at the current x location in the image. The sections below first describe how character models are selected, then how the match scores are subsequently computed for each model.

A. Prototype Selection

The first step in developing a recognition engine is to select a set of prototype graphemes and build inkball models out of them. In principle these graphemes may range from parts of characters to constructs over multiple characters. However, this paper focuses on single-character models as the most obvious starting point.

Several character selection strategies are tested in Section III-E. Although they differ in methodology and difficulty, the results show that the model selection has fairly little impact on the final results. In particular, more careful selection strategies perform only slightly better than *ad hoc* selection by hand.

2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 96-101

The two selection algorithms employed are based upon k-medoids and information gain, respectively. Both depend upon a distance measurement $\delta(c_1, c_2)$ taken between pairs of candidates c_1 and c_2. In each case we use the two-way inkball fit described in the following section, and consider every character in the document (segmented via an automated algorithm [3]) as a potential candidate.

The k-medoids algorithm adapts k-means to the case where averaging over several examples is not well-defined. It proceeds in the same fashion as k-means, except that at each step the cluster centers are not the actual centroid but the cluster member that is closest to it. After convergence, each cluster center becomes a prototype character. Executing the algorithm requires computation of the mutual distance between all examples belonging to each character class. The experiments test up to $k = 5$.

A second method for selecting prototypes relies upon information gain. Information gain may be computed as the difference in entropy of a set before and after a partition into subsets, and is commonly used to select features in decision trees. For a given candidate prototype, the distance to all other characters must be computed, both those of the same class and of the other classes as well. These distances can be thresholded to divide the population into two subsets, and the maximum information gain over all possible thresholds is computed and stored for each candidate prototype. Intuitively, members of the same character class should be closer together, so thresholding the distances should tend to improve the purity of the subsets. For each character class, the candidate with the greatest information gain becomes the prototype. Additional prototypes can be selected via a greedy process. Given the partitioning induced by the already selected prototype(s), a new candidate that produces the greatest additional information gain across all the partitions is selected next, and so on.

B. Inkball Model Development

Once a set of character prototypes has been selected, they are converted to inkball models. An inkball model consists of overlapping disks of ink arranged in a 2D structure specified in terms of the expected spatial offset between neighboring disks. To generate the model corresponding to a particular character sample, first thin it to single pixel width. Place disks at all junctions and endpoints, then along the skeleton according to two rules:

1) If possible, place a new disk on the skeleton at a distance ρ from its nearest neighbor already in place.
2) Otherwise place a new disk on the skeleton at the maximum possible distance from those previously placed, and at least $\rho/2$ from its nearest neighbor.

Placement stops when it is no longer possible to add new disks according to either rule.

Links are created greedily between nearest neighbors until the disks are all joined in a tree structure. The expected offset for each neighbor pair is taken directly from their relative positions in the prototype character. If q_i is the ith node, and $q_{i\uparrow}$ is its parent, then the offset $\vec{t}_i$ is the 2D vector specifying the position of q_i relative to $q_{i\uparrow}$. Assuming without loss of generality that q_1 is the root, $\vec{t}_1$ is undefined. The tree and associated offsets form the inkball model.

We restrict the model structure to a tree in order to keep the matching process tractable, as described in the following section. This means in practice that character prototypes containing loops must be represented with a break placed arbitrarily somewhere around the loop. When starting with online data, the model can be made to follow the pen trace, which will contain a natural break. This option is not available with offline data.

C. Inkball Model Fitting

Both model selection and HMM training call for computing the best match between an inkball model and an arbitrary sample of handwritten text. Qualitatively speaking, a model fits well to a text image if its nodes lie close to the skeleton of the observed text when the two are superimposed. Recognizing that real handwriting always varies in practice, the model is allowed to deform slightly to conform better to the image. The match score defined below thus incorporates both the proximity to observations and the degree of deformation; a good score is achieved when the match is close and the deformations small. In most cases, the goal is to find the configuration with the minimum possible energy, optionally while also constraining the position of the root node.

To be more precise, define a *configuration* $C = \{v_1, ...v_n\}$ of an inkball model Q as a placement of its n nodes with respect to an image, with $\vec{v}_i$ the location of node q_i. These placements in turn give the *configuration offsets* $\vec{s}_i = \vec{v}_i - \vec{v}_{i\uparrow}$. Now let Ω represent the Euclidean distance transform of the thinned text image; in other words, $\Omega(\vec{v}_i)$ is the distance from $\vec{v}_i$ to the nearest skeleton point. The match score of a single configuration attempts to approximate the negative log probability of the model given the observation, and is the weighted sum of two parts: a deformation term E_ξ and a data fidelity term E_Ω.

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

$$E_\xi(Q, C) = \sum_{i=2}^{n} \|\vec{s}_i - \vec{t}_i\|^2 \tag{2}$$

$$E_\Omega(C, \Omega) = \sum_{i=1}^{n} \Omega(\vec{v}_i)^2 \tag{3}$$

For simplicity the expressions above drop some normalization coefficients given in prior work [1], since in practice they are held constant and therefore act merely as a multiplier to the overall energy function. Given the above definitions, the fit of a model to a text image is the minimum

value of E over all possible configurations. This value, along with the configuration that produces it, can be computed relatively efficiently via the dynamic programming algorithm given in the next section.

D. Fit Optimization

Functions of 2D position such as Ω can be represented discretely over a grid, typically sampled at each pixel location. The same sort of grid can also represent other 2D functions, such as one that gives the minimal energy over the set of configurations sharing a root located at $\vec{v}$:

$$\mathcal{E}(\vec{v}) \equiv \min_{C|\vec{v}_1=\vec{v}} E(Q, C, \Omega) \tag{4}$$

Generalizing this concept further, define a family of functions based on inkball model Q as follows: $\mathcal{E}^{(i)}(\vec{v})$ is the function giving the minimal energy for the subtree of Q that has root q_i, when q_i is placed at $\vec{v}$.

With these definitions in place, the stage is set to describe the computation of $\mathcal{E}(\vec{v})$ via dynamic programming. For the moment, consider an inkball model with no branches, rooted at q_1 and linked via consecutive q_i to the leaf at q_n. Think of the discrete values of $\vec{v}_i$ represented by the grid locations as possible states of q_i. The computation on this simple model is analogous to a Markov chain, with the energies $\mathcal{E}^{(i)}(\vec{v})$ dependent only on $\mathcal{E}^{(i+1)}(\vec{v})$ and the transition costs from $\vec{v}_{i+1}$ to $\vec{v}_i$. $\mathcal{E}^{(i)}(\vec{v})$ is the minimum over all possible child positions $\vec{v}_{i+1}$ of the child's energy, plus the transition cost from $\vec{v}_{i+1}$ to $\vec{v}$, added to the intrinsic costs associated with $\vec{v}_i$ itself.

$$\mathcal{E}^{(i)}(\vec{v}) = \min_{\vec{u}} \mathcal{E}^{(i+1)}(\vec{u}) + \|(\vec{u} - \vec{v}) - \vec{t}_{i+1}\|^2 + \Omega(\vec{v})^2 \tag{5}$$

Explicitly computing all entities in this minimization would be impractical, but fortunately an operation called the *generalized distance transform* (GDT) computes the answer efficiently, in a number of operations that rises linearly with the number of pixels [10]. Representing the GDT as Γ gives a simplified expression.

$$\mathcal{E}^{(i)}(\vec{v}) = \Gamma\left(\mathcal{E}^{(i+1)}(\vec{v} - \vec{t}_{i+1})\right) + \Omega(\vec{v})^2 \tag{6}$$

For the general case where a node may have multiple children, each contributes to the parent's energy.

$$\mathcal{E}^{(i)}(\vec{v}) = \Omega(\vec{v})^2 + \sum_{j:q_{j\uparrow}=q_i} \Gamma\left(\mathcal{E}^{(j)}(\vec{v} - \vec{t}_j)\right) \tag{7}$$

A remarkably simple algorithm arises from this equation, again using discrete grids to represent 2D functions such as $\mathcal{E}^{(i)}(\vec{v})$. In the description below, each capitalized variable represents such a grid, and the *Shift* operation translates the values in the grid by the specified vector, using bicubic interpolation to handle fractional translations. Γ^* is a byproduct of the GDT computation and records for each point in the grid which particular $\vec{u}$ produces the minimum value in Equation 5.

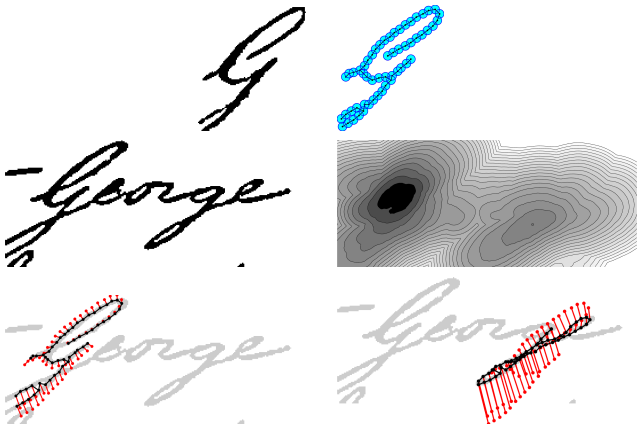

Figure 1. Several steps in the model fitting process. First row: a character G prototype and the corresponding model. Second row: a sample word and the corresponding fit score $\mathcal{E}(\vec{v})$ for the model G. Third row: recovered configurations for two local optima, with warping from the default configuration shown in red.

for all nodes in postorder traversal **do**
  Initialize: $E \leftarrow \Omega^2$
  for all children j of the current node **do**
    Translate: $T \leftarrow Shift(\mathcal{E}^{(j)}, -\vec{t}_j)$
    Transform: $G \leftarrow \Gamma(T)$
    Optional: $L^{(j)} \leftarrow \Gamma^*(T)$
    Add: $E \leftarrow E + G$
  end for
  $\mathcal{E}^{(i)} \leftarrow E$
end for

The final result of this algorithm $\mathcal{E}^{(1)}$ holds the minimum possible configuration energy for the model as a 2D function of the root node position. To get the best fit possible across an entire image, simply take the minimum over the grid. For a two-way symmetric comparison between two images, take the maximum of each one-way match.

If desired, the corresponding configuration may be recovered from the $L^{(i)}$ recorded during minimization using the algorithm below.

$\vec{v}_1 \leftarrow \arg\min_{\vec{u}} \mathcal{E}^{(1)}(\vec{u})$
for all nodes in preorder traversal **do**
  for all children j of the current node **do**
    $\vec{v}_j \leftarrow L^{(j)}(\vec{v}_i) + \vec{t}_j$
  end for
end for

Figure 1 illustrates several steps in the process of building an inkball model from a prototype and matching it to an image.

E. HMM Training and Recognition

The experiments use the spatially varying model fit scores for a set of grapheme prototypes as input to an HMM. The best fit $\mathcal{E}_Q^{(i)}(\vec{v})$ for the character prototype Q naturally varies with both the x and y position of the root. The HMM

expects a 1D function of the x position, which is achieved by minimizing over columns.

$$f_Q(x) = \min_y \mathcal{E}_Q^{(i)}(x, y) \tag{8}$$

A number of different grapheme sets are tested, but in each case the procedures are similar. Each prototype is fit to segmented words or lines of text using Algorithm 1 above. The results from Equation 8 for the entire set of prototypes become the observations available to the HMM.

We use continuous character HMM with a linear topology and a mixture of Gaussians to model the feature distribution. Diagonal covariance matrices are considered to reduce the model complexity. System parameters include the number of mixtures per state and the number of states per character. The character models are trained with labeled word images using the Baum-Welch algorithm and word recognition is performed with Viterbi decoding [5].

III. Experiments

All experiments use the GW20 corpus, a well-benchmarked set of 20 pages from the correspondance of George Washington [11]. Setting up for experiments on additional datasets requires a nontrivial effort because all possible character samples must be segmented and compared. This paper therefore focuses on the development and relative comparison of candidate methodologies using the inkball and HMM combination, for which purpose GW20 suffices handily. Once the foundational investigations are complete, future work can evaluate the method thoroughly in comparison with reference methods.

Due to the relatively small size of the database, we perform a four-fold cross-validation. In each fold, 10 training pages are used for training the character HMM, 5 validation pages are used for optimizing system parameters with respect to the recognition accuracy, and 5 test pages are used for evaluating the final performance. Finally, the test set results are averaged over the four cross-validations.

For training, the number of states per character HMM is adopted from previous work [12]. It corresponds to a fraction of the estimated mean character width. The number of mixtures is incremented step-by-step from 1 to 19, retraining the models four times on the training set in each step. Afterwards, the number of mixtures M is optimized over $M \in \{1, 4, \ldots, 19\}$ on the validation set with respect to the recognition accuracy. Finally, the system is evaluated on the independent test set. Considering a closed vocabulary, each word image is assigned to one of 1,200 word classes taking into account equal word priors.

The HTK toolkit[1] is used for Baum-Welch training and Viterbi decoding.

A. Inkball Features

[1] http://htk.eng.cam.ac.uk

Table I
Inkball features (variations). Word error in percentage for four cross-validations (CV).

Features	CV-1	CV-2	CV-3	CV-4	Average
BL	26.49	**20.69**	**29.81**	21.04	**24.51**
BM	29.90	23.48	32.59	26.03	28.00
S3	27.15	20.86	30.88	21.86	25.19
DC	**25.99**	24.32	30.55	20.87	25.43
WC	26.32	23.91	30.64	**20.05**	25.23

In a first experiment, several variants of the inkball features are computed based on a manually selected set of prototypes, that is one inkball prototype for each of the 60 characters. Besides the baseline (**BL**), the following variants are tested:

- Experiment **BM** uses a part-structured boundary model [2] instead of an inkball model.
- Experiment **S3** includes three versions of each prototype character, scaled to 80%, 100%, and 125% size respectively.
- Experiment **DC** adds an additional feature for each character prototype equal to the derivative of the feature value with respect to the x coordinate. This is intended to indicate whether a particular location is near the center of a character or close to the edge.
- Experiment **WC** adds an additional feature for each character prototype equal to the minimum score achieved by the feature value within a character-width window around the current x value. Character width is computed as the standard deviation of the x coordinates of the character prototype.

The word error results are shown in Table III. On average, none of the variants significantly outperforms the baseline features but the results are included for completeness of the record.

B. Prototype Selection

In principle any character sample may be used as a prototype, but it is possible that carefully selected prototypes will give better result. Section II-A described two selection methods, one based upon the k-medoids (**KM**) algorithm and one upon information gain (**IG**). With each selection method, it is also possible that using multiple prototypes per character could improve results, particularly for characters with more diversity of form. Feature sets **KM1** through **KM5** and **IG1** through **IG3** test this possibility, with the numeral in each case indicating the number of prototypes used per character (where available; some appear too rarely).

The outcome of this second experiment is shown in Table II. The best results are achieved with **IG2** when selecting up to two prototypes per character based on the information gain criterion, which slightly outperforms the baseline features. Note that the prototypes have been selected on the first 10 pages of the manuscript. Although they are

Table II
INKBALL FEATURES (PROTOTYPE SELECTION). WORD ERROR IN PERCENTAGE FOR FOUR CROSS-VALIDATIONS (CV).

Features	CV-1	CV-2	CV-3	CV-4	Average
BL	26.49	20.69	29.81	21.04	24.51
KM1	26.11	22.24	29.88	21.83	25.01
KM2	26.52	21.75	29.88	20.92	24.77
KM3	26.61	20.02	30.12	20.43	24.30
KM4	26.90	20.94	29.07	20.96	24.47
KM5	27.43	20.47	30.19	20.98	24.77
IG1	25.45	20.59	**27.82**	**20.10**	23.49
IG2	**25.04**	**19.69**	27.98	20.59	**23.33**
IG3	26.03	20.43	28.40	21.09	23.99

Table III
INKBALL FEATURES WITH TRUNCATED DEFORMATION ENERGY. WORD ERROR IN PERCENTAGE FOR FOUR CROSS-VALIDATIONS (CV).

Condition	CV-1	CV-2	CV-3	CV-4	Average
BL ($\tau = \infty$)	26.49	20.69	29.81	21.04	24.51
T4 ($\tau = 4$)	23.15	18.45	**24.86**	17.96	21.11
T8 ($\tau = 8$)	**22.41**	**17.79**	25.35	**17.22**	**20.69**
T16 ($\tau = 16$)	24.30	19.19	27.82	19.03	22.58
T32 ($\tau = 32$)	26.52	21.33	29.14	21.58	24.64
T64 ($\tau = 64$)	27.18	23.15	29.79	22.73	25.71

selected from the test data for cross-validations CV-2 and CV-3, 60 prototypes represents less than 0.3% that might be affected, of more than 21,000 characters in the manuscript. A followup experiment using prototypes selected instead from the last ten pages shows no significant changes in the outcome, confirming the insignificance of the prototypes' source.

C. Modified Deformation Energy

A third experiment looks at a simple modification to the deformation term of the match score, E_ξ. Specifically, Equation 2 is replaced by a truncated Gaussian. The match thus penalizes deformations normally until the energy reaches a threshold τ, after which the penalty stays fixed at τ for all higher deformation levels. This reflects the intuitive notion that all sufficiently bad matches should be treated equally. the

$$E_\xi(Q, C) = \sum_{i=2}^{n} \min\left(\|\vec{s}_i - \vec{t}_i\|^2, \tau\right) \quad (9)$$

Table III shows the result for varying levels of τ. The baseline is effectively $\tau = \infty$. The other experiments **T4** through **T64** test varying values for τ, with the numeric suffix indicating the value.

D. Reference Feature Sets

Three different feature sets known to work well in handwriting recognition form a control group. All the reference features are extracted using a C++ keyword spotting tool[2].

[2]https://github.com/wichtounet/word_spotting

M01 is a well-established feature set that was originally proposed for modern handwriting recognition [7]. Using a horizontal sliding window, nine geometrical features are extracted at each column of the image, from left to right. There are three global features per window, the number of black pixels, its center of gravity and its second order moment. The six local features are the position and gradient of the upper and lower contours, the number of transitions from black to white and the number of black pixels between the upper and lower contour.

R08 is a set of features inspired from SIFT, proposed for handwriting word spotting [8]. A square sliding window moves from left to right over the image extracting 128 features at each column of the image. The features are extracted by fitting a 4×4 grid on the window. In each of the cell, the gradients are quantized into 8 bins of regularly spaced orientations. Finally, the features of each frame are normalized so that their component sum to 1.

T09 uses a slit-style Histogram Of Gradients (HOG) feature for handwriting keyword spotting [9]. A narrow window slides from left to right, extracting 384 features per column. A 4×4 grid is fitted on each window and gradients are accumulated into 16 orientation bins, in each cell, using the signed gradient. A 4×2 block, sliding vertically, is then used to accumulate and normalize the final features, generating vertical redundancy.

E. Comparison with the State of the Art

Table IV shows the results of the control experiment, which compares the proposed inkball features with the three reference methods. For the inkball features, we show the both best feature selection method (**IG2**) and the best setup using a truncated Euclidean (**T8**).

The comparison to related work should be approached cautiously. None of the comparison methods perform as well as the proposed inkball features under the described experimental conditions. Relative improvements of 43% on average are reported for **T8** when compared with the best reference features.

However, it is important to point out that we used a set of word images made available by [11] that might not be ideal for the reference features. Especially the geometrical features from Marti & Bunke [7] rely on normalization techniques such as skew removal, slant removal, and scaling. None of these operations have been applied in our experiments. Furthermore, the word images, which have been extracted from the original pages based on bounding boxes, often contain parts of other characters and underlines. These artifacts add noise to the geometrical features, especially to the contour-related ones. This might explain the low performance of the Marti & Bunke features, which have achieved stronger results on the George Washington dataset in other studies, for example in [12] where a word error

Table IV
COMPARISON WITH REFERENCE FEATURES. WORD ERROR IN PERCENTAGE FOR FOUR CROSS-VALIDATIONS (CV). IMPROVEMENT IS INDICATED WITH RESPECT TO BEST REFERENCE.

Features	CV-1	CV-2	CV-3	CV-4	Average
M01 [7]	53.71	54.37	60.41	54.61	55.78
R08 [8]	33.91	35.74	42.22	34.08	36.49
T09 [9]	35.75	37.23	41.32	38.55	38.21
Proposed: IG1	25.04	19.69	27.98	20.59	23.33
Improvement	26%	45%	32%	40%	36%
Proposed: T8	**22.41**	**17.79**	25.35	**17.22**	**20.69**
Improvement	34%	50%	39%	49%	43%

rate of 24.1% is reported when using normalized text line images and bigram language models.

Although it is not therefore established as the best method under any circumstances, the results do support the conclusion that the proposed inkball features outperform the reference methods on unnormalized images. This is an advantage since normalization often is a challenging task for historical manuscripts and avoiding it may be preferable in some cases.

IV. CONCLUSION

This paper has proposed a novel use for inkball models, casting them as input features to a traditional HMM. The method achieves good results with just one prototype sample per character class chosen in *ad hoc* manner. Slightly better performance can be achieved using prototypes selected to maximize information gain, but the work required to do this may outweigh the modest benefit.

More research is necessary to answer fundamental questions about this approach. The results should be extended and tested on additional manuscripts and writing styles, and also compared to the reference algorithms on the deskewed clean images those methods prefer. A diverse multiwriter collection will presumably require more prototypes per character, yet several of the results in Table III hint that adding features to the HMM can hurt overall performance. This suggests that a feature selection mechanism may also offer promising avenues for improvement. Many options remain to be explored within this new framework.

REFERENCES

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

[2] N. Howe, A. Yang, and M. Penn, "A character style library for syriac manuscripts," in *Proceedings of the 2015 Workshop on Historical Document Imaging and Processing*. ACM, 2015.

[3] N. Howe, "Inkball models for character localization and out-of-vocabulary word spotting," in *International Conference on Document Analysis and Recognition*, 2015.

[4] A. Fischer, K. Riesen, and H. Bunke, "Graph similarity features for HMM-based handwriting recognition in historical documents," in *Proc. Int. Conf. on Frontiers in Handwriting Recognition*, 2010, pp. 253–258.

[5] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–285, 1989.

[6] T. Ploetz and G. A. Fink, "Markov models for offline handwriting recognition: A survey," *Int. Journal on Document Analysis and Recognition*, vol. 12, no. 4, pp. 269–298, 2009.

[7] U.-V. Marti and H. Bunke, "Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system," *Int. Journal of Pattern Recognition and Artificial Intelligence*, vol. 15, pp. 65–90, 2001.

[8] J. A. Rodrıguez and F. Perronnin, "Local gradient histogram features for word spotting in unconstrained handwritten documents," in *Proceedings of the Int. Conf. on Frontiers in Handwriting Recognition*, 2008, pp. 7–12.

[9] K. Terasawa and Y. Tanaka, "Slit style HOG feature for document image word spotting," in *Proceedings of the IEEE Int. Conf. on Document Analysis and Recognition*. IEEE, 2009, pp. 116–120.

[10] P. Felzenszwalb and D. Huttenlocher, "Distance transforms of sampled functions," *Theory of Computing*, vol. 8, no. 19, 2012.

[11] 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.

[12] A. Fischer, H. Bunke, N. Naji, J. Savoy, M. Baechler, and R. Ingold, "The HisDoc project. automatic analysis, recognition, and retrieval of handwritten historical documents for digital libraries." *Beihefte zu Editio*, vol. 38, pp. 91–106, 2014.