Synthetically Generated Semantic Codebook for Bag-of-Visual-Words Based Word Spotting

David Aldavert and Marçal Rusiñol
Computer Vision Center (CVC), Dept. Ciències de la Computació
Edifici O, Universitat Autònoma de Barcelona
08193 Bellaterra (Barcelona), Spain
{*aldavert,marcal*}*@cvc.uab.cat*

***Abstract*—Word-spotting methods based on the Bag-of-Visual-Words framework have demonstrated a good retrieval performance even when used in a completely unsupervised manner. Although unsupervised approaches are suitable for large document collections due to the cost of acquiring labeled data, these methods also present some drawbacks. For instance, having to train a suitable "codebook" for a certain dataset has a high computational cost. Therefore, in this paper we present a database agnostic codebook which is trained from synthetic data. The aim of the proposed approach is to generate a codebook where the only information required is the type of script used in the document. The use of synthetic data also allows to easily incorporate semantic information in the codebook generation. So, the proposed method is able to determine which set of codewords have a semantic representation of the descriptor feature space. Experimental results show that the resulting codebook attains a state-of-the-art performance while having a more compact representation.**

***Keywords*-Word Spotting; Bag of Visual Words; Synthetic Codebook; Semantic Information;**

I. Introduction

Handwritten keyword spotting is the document image retrieval task devoted to obtain a ranked list of words that are relevant to a user's cast query. In its most simple formulation, document images are already pre-processed and segmented into individual words. The user casts a query in form of an example of the keyword he wants to retrieve, to then obtain a ranked list in which desirably the words having the same transcription are ranked better than the rest of the words. This paradigm is known as segmentation-based query-by-example keyword spotting, which is the scenario in which we are centered in this paper.

Since the seminal papers of Manmatha et al. [1], [2] that introduced the problematic of handwritten keyword spotting more than twenty years ago, many advances have been proposed. Performances reached on public datasets have been steadily increasing with the proposal of better feature representations and retrieval strategies. In addition to the overall retrieval accuracy, many other advances have been made as well. Segmentation-free methods have been proposed [3], [4], [5], [6], [7], query-by-string techniques have emerged [8], [9], [10], [11], [12], and different methods have incorporated techniques from the information retrieval field such as relevance feedback [13], re-ranking [4] or query expansion [4].

Although systems which incorporate a learning step to improve the retrieval accuracy obtain a better performance than systems purely based on visual information [5], [10], [14], [9], unsupervised methods are more desirable in certain scenarios. For example, in large document collection with hundreds of pages and without any annotation, an unsupervised method can be used directly without manually annotate a subset of pages. Also, an unsupervised method can be used for instance to group similar looking word snippets into clusters [15]. This word clusters then can be used to simply accelerate the retrieval system but also to propagate the annotations provided by the user or to search consensus to the annotations given by a text recognition system.

Unsupervised word spotting methods based on the Bag-of-Visual-Words paradigm can attain a high retrieval performance when the methods used at each step are selected carefully [16]. Besides its retrieval accuracy these methods have the advantage that words are represented by a fixed-length vector, so standard dimensionality reduction techniques have been used to efficiently store and index large collections of documents [3], [17]. However, these methods require the use of a codebook to encode locally extracted descriptors into codewords. The performance of the system is dependent on the quality of the codebook and the number of codewords which yields a better trade-off between dimensionality (i.e. memory usage) and performance has to be found. On small datasets, creating a codebook does not have a high cost, but, in large collections, with hundreds of thousands of words written by multiple writers the computational cost of generating the codebook might be prohibitive. A straightforward solution is to randomly sample a subset of word snippets to generate the codebook. However, this approach has the drawback that certain characters and writing styles may be underrepresented by the codebook. Therefore, we propose a codebook trained from synthetic data which incorporates semantic information in the generation process to determine the optimal size and cardinality of the codewords. The use of synthetic data has several advantages (c.f. [18], [14], [19]): it ensures that all characters are properly represented and it allows to simulate the script variability present in documents written by multiple writers. Since there are many true-type fonts which replicate the human handwritten style it is easy to incorporate many different versions of the same character. Additionally, it also allows

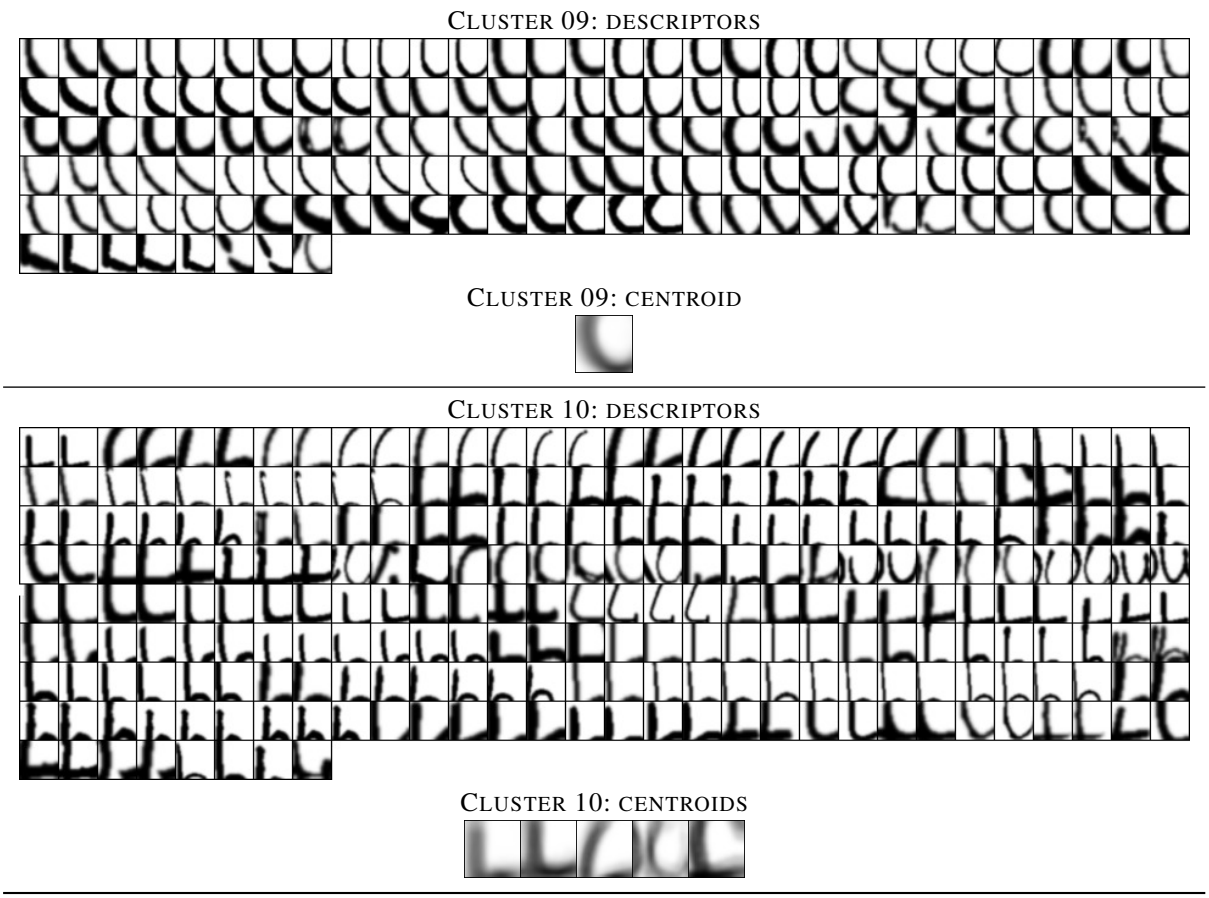

Figure 1: Two examples of the clusters generated by the codebook. CLUSTER 09 contains descriptors from symbols [0, 8, c, g, b, j, h, p, f, x, u], while CLUSTER 10 is formed by descriptors from [h, b, 4, 6, t, k, u, o, f, d, j, l, p, y, w, a, q, i, n]. CLUSTER 10 is represented by 5 centroids because it has at least a nested cluster.

to incorporate semantic labels to the information used to create the codebook. This extra information is used to obtain an actual measure of the clustering accuracy of the codebook. Thus, the codebook is able to automatically determine the amount of codewords needed to properly represent the feature space. The main contributions of the paper are threefold. We present a method to generate a codebook from synthetic data. A new procedure used to encode descriptors into visual words is proposed. Finally, we provide the basis of a method to encode descriptors efficiently.

The rest of the paper is structured as follows: in Section II we present the method used to to create the codebook from synthetic data. Then, in Section III, we show how descriptors are encoded into visual words. Finally, in Section IV, we present the spotting performance attained by the proposed codebook and, in Section V, we discuss the main contributions of the paper.

II. SYNTHETIC CODEBOOK GENERATION

The codebook is trained with HOG descriptors [20] extracted from characters generated by true-type fonts that replicate the human handwriting style. Training samples are then pairs $\mathbf{x} = (\mathbf{d}, c)$, where $\mathbf{d}$ is the HOG descriptor and $c \in \mathcal{C}$ is the semantic label of the character. Therefore, the training set is formed by the training samples extracted from all the considered characters. Additionally, we also incorporate training samples that cover multiple characters (i.e. bigrams). We use the statistical data reported by Jones and Mewhort to select the bigrams most common in the English language [21]. Therefore, the training set is generated from 62 different characters and 1874 character bigrams, and has 36 different semantic labels as we do not differentiate between upper and lower case characters.

We generate the codebook by fist grouping the training samples using agglomerative clustering and then using the Shannon entropy to partition these tree into multiple clusters.

A. Agglomerative Clustering

Agglomerative clustering is a bottom-up hierarchical clustering algorithm that recursively groups the two closest clusters until all samples are grouped together. This procedure generates a binary tree that later has to be partitioned into clusters by using some criteria (e.g. fixed number of clusters, cluster compactness [22], *a contrario* approach [23]). The distance between clusters can be computed in many ways but the most common are the distance between the closest two elements (i.e. single-linkage), the distance between the two further away el-

ements (i.e. complete linkage) and the average distance between all elements of the cluster. The estimation of these distances though limit the practical usage of the method as the complexity of the standard algorithms have a $O(N^2)$ complexity both in terms of memory and runtime. For average distances, Leibe et al. proposed the average-link clustering with nearest neighbor chains [22] which reduces the memory complexity to $O(N)$. However, their algorithm can only be used when the dot-product or the Euclidean distance are used as similarity measure between clusters. Therefore, we decide to use the dot-product as similarity measure as HOG descriptors are L_2-normalized so in this case the dot-product is equivalent to the Euclidean distance. Furthermore, the dot-product also allow us to use other distance measures via explicit feature maps [24]. Hence, we are also able to compare HOG descriptors using the Histogram intersection and the χ^2 similarity measures.

Finally, we need to reduce the number of samples used to create the codebook. Although the memory complexity has been reduced to $O(N)$ the temporal complexity remains $O(N^2)$. In order to improve the algorithm runtime, we reduce the number of samples extracted at each character. Instead of using random sampling, we apply the agglomerative clustering at each character independently and then we generate clusters by selecting the sub-trees that have at least R samples. These clusters are then fed to the general agglomerative tree to generate the final codebook.

B. Shannon Entropy

Once we have generated the binary tree, we need a method to partition the nodes in order to obtain the clusters. We want that the partitions are created automatically from data so the user does not need to tweak another parameter. Since the samples have the character label besides the descriptor, we can use this information to partition the tree into semantically meaningful clusters. Therefore, we calculate at each node the Shannon entropy, as in [25]:

$$S_c(L,T) = \frac{2I_{c,t}(L)}{H_c(L) + H_t(L)}$$

where H_c is the class entropy of the samples at the node, H_t is the entropy of the samples division at the two children and $I_{c,t}(L)$ is the mutual information of the split:

$$\begin{aligned} H_c(L) &= -\sum_{c\in\mathcal{C}} \frac{n_c}{n} \log_2 \frac{n_c}{n}, \\ H_t(L) &= -\frac{n_l}{n} \log_2 \frac{n_l}{n} - \frac{n_r}{n} \log_2 \frac{n_r}{n} \\ I_{c,t}(L) &= H_c(L) - \frac{n_l}{n} H_c(L_l) - \frac{n_r}{n} H_c(L_r) \end{aligned}$$

Here, L, L_l and L_r respectively denote the set of samples at the current node, the left child and the right child and, n, n_l, n_r and n_c denote the cardinality of these sets with n_c being the number of samples within category c.

Using this measure, the higher the Shannon entropy the better are the categories distributed between the two descending nodes. Then, we compute this measure at each node and we partition the trees at the nodes where the Shannon entropy attains a local maxima, i.e. the nodes where it is higher than its direct ascendants and descendants. In order to avoid generating small clusters, nodes which do not have at least 50 samples are not considered. By following this procedure, we are able to generate clusters automatically and these clusters have some semantic significance. In Fig. 1, we see an example of the descriptors grouped in two different clusters. In this example, we can see that clusters contain elements from multiple characters as the features sampled from the image are too small and do not contain enough information to perfectly discriminate between the different characters. Although a perfect semantic separation is more desirable, it is not possible to achieve without a more complex descriptor or kernels (e.g. χ^2-Radial Basis Function kernel). Finally, applying the measure locally results in clusters being nested. This means that a cluster can be a sub-tree from another larger cluster so, we may need multiple centroids to represent them properly. For example, the second cluster in Fig. 1 has a nested cluster and thus it is represented by multiple centroids.

III. Descriptor Encoding

Once we have created the codebook we need to define how descriptors are going to be represented as visual words.

A. Codeword Encoding

We are going to represent descriptors using first derivative encoding [26], [27], i.e. descriptors are represented as the residual between the encoded descriptor and a selected codeword. So we need to represent that codewords can be represented by a centroid, but our agglomerative codebook can generate nested codewords. Therefore, codewords are represented by as many centroids as necessary to ensure that no overlapping exist. In Fig. 2, we can see a simplified

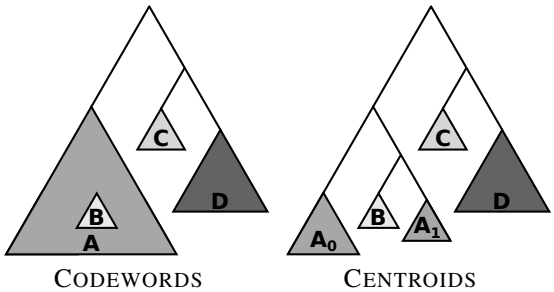

Figure 2: Schema of the agglomerative tree codewords and centroids.

representation of an agglomerative tree where codeword B is nested inside codeword A. Here, codeword A is represented by two centroids, A_0 and A_1, instead of the centroid at the highest level of the sub-tree so there is no overlapping with centroid B. In Fig.1, we can see a

real example of the centroids representing two different clusters.

Then, the contribution of each centroid $\mathbf{C}_i$ of the codebook to encode a given descriptor $\mathbf{d}$ is given by

$$\mathbf{w}_i = W(\mathbf{d}, \mathbf{C}_i) = \exp\left(-\frac{(1 - sim(\mathbf{d}, \mathbf{C}_i))^2}{2\sigma_i^2}\right)$$

where $sim(\mathbf{d}, \mathbf{C}_i)$ is the similarity measure between the descriptor and centroid and, σ_i is the standard deviation of the similarity of the elements within the centroid. In order to increase the sparseness of the encoding, we set to zero the different weights $\mathbf{w}_i$ that satisfy that $\mathbf{w}_i/\mathbf{w}_{\max} < t$ where $\mathbf{w}_{\max}$ is the maximum weight and $t \in [0, 1]$ is a threshold. When $t = 0$ we use all centroids to encode a descriptor while when $t = 1$ we only use the most similar centroid. Finally, weights are normalized to ensure that the sum of all weights is 1.

These weights multiplied by the residuals between the descriptor and the centroids are the resulting encoding. Since a codeword may be represented by several centroids, the contributions of all its centroids are accumulated together. For instance, in the example at Fig. 2 the codebook only has 4 codewords thus the histogram of visual words has $4d$ dimensions where d is the dimensionality of the descriptor. Then, the weighted residuals from A_0 and A_1 are both accumulated in the first d dimensions of the histogram.

B. Approximate Codebook

The encoding method previously described is computationally intensive as it requires computing multiple exponential weights to encode a single descriptor. In order to reduce the computational cost of the descriptor encoding step, we propose the use of an additional codebook which is used to approximate the descriptors. We use a Hierarchical k-Means (HKM) similar to the Vocabulary Tree [28] to approximate the descriptors. The codebook has degree 10, we limit it at a maximum depth of 8 levels and one million leafs. It is build using a priority queue that prioritize nodes which are more populated. In this codebook, we use the Euclidean distance to compare the descriptors regardless of the distance measure used by the agglomerative codebook.

The main idea is to then use the leafs of this codebook as an approximation of the encoded descriptors. Then, we pre-compute the encoding of the leaf descriptors since we know them *a priori*. Thus, instead of computing the weight of each centroid of the agglomerative tree for a given descriptor, we only need to traverse the HKM tree and use the weights stored at the leaf. Although we are adding quantization errors when following this procedure, we are also greatly reducing the encoding computational cost which may be a desirable trade-off when dealing with large collections.

IV. Results

We generate the codebook using ten different true type fonts which generate between 4000 and 7000 descriptors per character. We group these descriptors in clusters of at least ten descriptors (i.e. $R = 10$) reducing the contribution of each character to 450-800 descriptors. Therefore, the algorithm only needs to aggregate around 50000 descriptors in each evaluated configuration. In all experiments, the Bag-of-Visual-Words (BoVW) signature is generated by densely sampling HOG descriptors each 4 pixels from squared regions of 16, 24, 32 and 40 pixels, a spatial pyramid with 5 horizontal partitions and power factorization at 0.5. We have evaluated the codebooks obtained using different descriptor dimensionality, filter ratio and similarity measures on the George Washington dataset [29], [15]. The dataset consist of 20 pages with 4860 segmented words. The performance of the retrieval system is evaluated computing the mean Average Precision (mAP) score for any word snippet that appears at least twice in the dataset and returning the overall performance of the system as the mean of mAP scores. In Fig. 3, we plot the results obtained by five different queries. All results have been obtained on a Linux box with an Intel® Xeon® E5-1620 CPU running at 3.50GHz and 16 Gb of RAM.

Dim.		$r \geq 0\%$		$r \geq 80\%$		$r \geq 90\%$		$r \geq 95\%$	
	Simil.	EUC	HIS	EUC	HIS	EUC	HIS	EUC	HIS
32	CHI	61.6	61.6	68.7	65.1	68.5	65.0	68.2	64.4
	EUC	40.1	41.9	52.3	55.8	52.7	54.4	54.6	56.6
	HIS	50.1	52.3	67.3	65.9	68.2	66.5	68.2	66.0
128	CHI	47.7	50.6	69.7	67.4	70.9	68.0	71.0	68.1
	EUC	40.5	42.2	54.1	58.2	57.7	62.1	59.8	63.3
	HIS	43.0	45.3	68.1	66.3	70.5	67.7	70.7	67.5

Table I: mAP scores at the Washington dataset.

In Table I, we show the mAP score obtained when generating the codebook with different configurations. The dimensionality column (*Dim.*) specifies the dimensionality of the HOG descriptors. In this experiment, we have tested them using 2×2 and 4×4 spatial bins resulting in descriptors of 32 and 128 dimensions. The similarity column (*Simil.*) indicates which similarity measure has been used to compare the descriptors when creating the agglomerative tree. The abbreviations *CHI*, *EUC* and *HIS* correspond to χ^2, Euclidean and Histogram intersection respectively. The mAP scores are divided into the ratio used to filter weights $w_i/w_{\max} \geq t$ where $t \in \{0\%, 80\%, 90\%, 95\%\}$. When $r \geq 0\%$ we accept all weights while we filter all weights which are smaller than 0.95 of the maximum weight in when $r \geq 95\%$. The similarity between the histograms of visual words is calculated using both Euclidean distance and Histogram Intersection similarity measures. The obtained results show that creating a more sparse encoding by filtering out small weights improves the performance of the algorithm. It also shows that using χ^2 or Histogram intersection to compare descriptors consistently gives a better codebook while the Euclidean distance is better when comparing the histograms of visual words. Comparing both descriptors, we see that the higher dimensional descriptor provides a better accuracy. How-

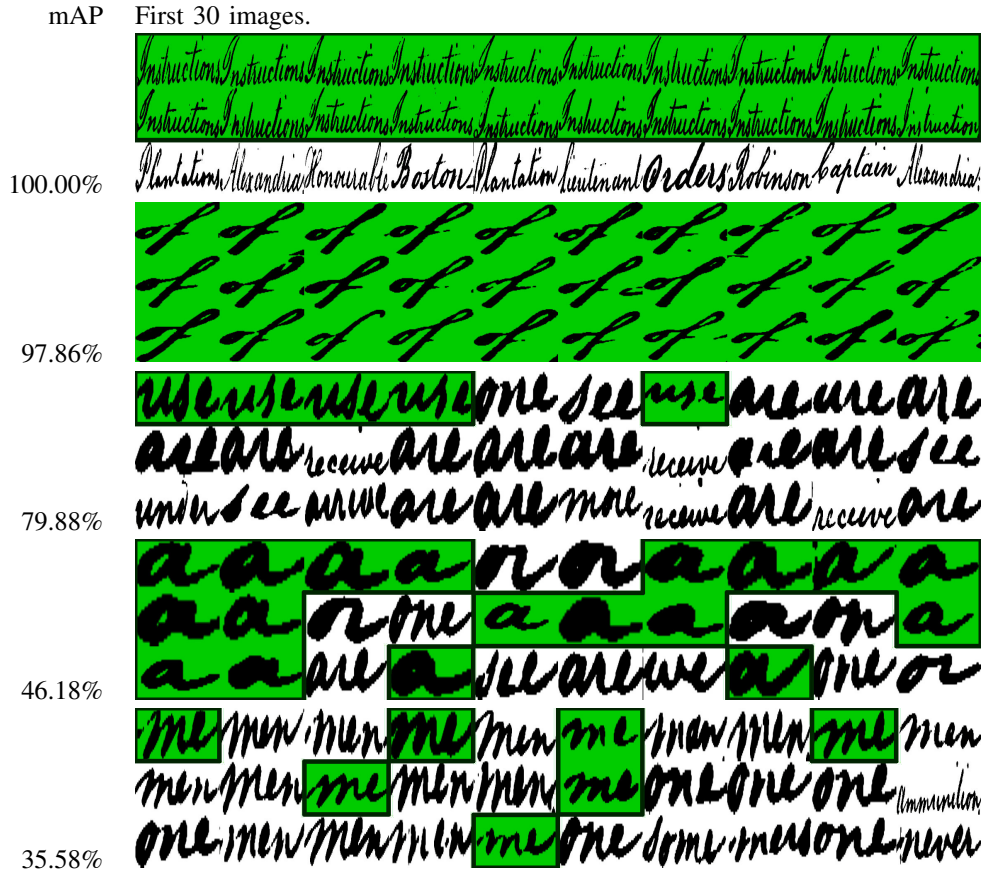

Figure 3: Some qualitative results obtained in the George Washington dataset.

Dim.	32			128		
Sim.	CHI	EUC	HIS	CHI	EUC	HIS
	823	741	788	852	787	849

Table II: Sizes of the evaluated codebooks.

ever, the performance increase is modest so depending on the application a smaller descriptor may be more suitable. Comparing these results with other word spotting methods, we can observe that proposed algorithm outperforms most unsupervised spotting methods [16]. We can also see that the proposed codebook shows a similar performance to a carefully crafted BoVW. For example, we reach a 71.0% mAP score while standard BoVW reaches $72,35\%$ mAP. However, our BoVW signature is more compact as we use less spatial bins (5 vs. 24 spatial divisions) and the codebook is much smaller. The codebooks generated by our method have between 750 and 850 codewords (see tableII) while a standard k-means codebook uses 4096 codewords (i.e. the k-means codebook is between 4,8 and 5,4 larger).

Finally, the codebooks using HOG-32 descriptors need on average 7 minutes to be created while HOG-128 require around 40 minutes on average. The encoding runtime for HOG-128 descriptors takes around 490 ms on average to encode a word snippets. This runtime can be reduced by more than an order of magnitude when the descriptors are approximated by a HKM codebook. In this case, encoding takes around 9.2 ms an average per word snippet. However, approximating the descriptors have the drawback that the mAP score consistently drops a 3-5% in all configurations.

V. Conclusions

In this paper, we have proposed a method to automatically generate a codebook from synthetic data. The main idea is to create a codebook which is database agnostic, i.e. a codebook which has a good performance independently from the data which is used to create it. This is important when processing large collections of documents as creating a codebook can be extremely time consuming. Thus, our algorithm is able automatically determine the amount and size of the clusters by incorporating semantic information into the codebook generation process. Besides, we have proposed the use of an additional codebook to approximate the descriptors and greatly reduce the descriptor encoding computational cost. The experimental results show that the codebook attains a similar performance to other unsupervised bag-of-visual-words spotting algorithms.

Acknowledgments

This work was supported by the Spanish projects TIN2014-52072-P and TIN2017-89779-P, by the CERCA

Programme / Generalitat de Catalunya and by the AGAUR and FEDER project 2016 LLAV 00057.

REFERENCES

[1] R. Manmatha, C. Han, E. Riseman, and W. Croft, "Indexing handwriting using word matching," in *Proc. ACM Intern. Conf. Digit. Libr.*, 1996, pp. 151–159.

[2] R. Manmatha, C. Han, and E. Riseman, "Word spotting: A new approach to indexing handwriting," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 1996, pp. 631–637.

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

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

[5] L. Rothacker, M. Rusiñol, and G. Fink, "Bag-of-features HMMs for segmentation-free word spotting in handwritten documents," in *Proc. Int. Conf. Doc. Anal. Recognit.*, 2013, pp. 1305–1309.

[6] A. Kovalchuk, L. Wolf, and N. Dershowitz, "A simple and fast word spotting method," in *Proc. Int. Conf. Front. Handwrit. Recognit.*, 2014, pp. 3–8.

[7] L. Rothacker, S. Sudholt, E. Rusakov, M. Kasperidus, and G. Fink, "Word hypotheses for segmentation-free word spotting in historic document images," in *Proc. Int. Conf. Doc. Anal. Recognit.*, 2017.

[8] D. Aldavert, M. Rusiñol, R. Toledo, and J. Lladós, "Integrating visual and textual cues for query-by-string word spotting," in *Proc. Int. Conf. Doc. Anal. Recognit.*, 2013, pp. 511–515.

[9] S. Sudholt and G. Fink, "PHOCNet: A deep convolutional neural network for word spotting in handwritten documents," in *Proc. Int. Conf. Front. Handwrit. Recognit.*, 2016, pp. 277–282.

[10] J. Almazan, A. Gordo, A. Fornes, and E. Valveny, "Word spotting and recognition with embedded attributes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 12, pp. 2552–2566, December 2014.

[11] L. Rothacker and G. Fink, "Segmentation-free query-by-string word spotting with bag-of-features HMMs," in *Proc. Int. Conf. Doc. Anal. Recognit.*, 2015, pp. 661–665.

[12] S. Sudholt and G. Fink, "Evaluating word string embeddings and loss functions for CNN-based word spotting," in *Proc. Int. Conf. Doc. Anal. Recognit.*, 2017.

[13] M. Rusiñol and J. Lladós, "Boosting the handwritten word spotting experience by including the user in the loop," *Pattern Recognit.*, vol. 47, no. 3, pp. 1063–1072, March 2014.

[14] P. Krishnan and C. Jawahar, "Matching handwritten document images," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 766–782.

[15] T. Rath and R. Manmatha, "Word spotting for historical documents," *Int. J. Doc. Anal. Recognit.*, vol. 9, no. 2, pp. 139–152, April 2007.

[16] D. Aldavert, M. Rusiñol, R. Toledo, and J. Lladós, "A study of bag-of-visual-words representations for handwritten keyword spottings," *Int. J. Doc. Anal. Recognit.*, vol. 18, no. 3, pp. 223–234, September 2015.

[17] S. Sudholt and G. Fink, "A modified isomap approach to manifold learning in word spotting," in *Proc. Ger. Conf. Pattern Recognit.*, 2015, pp. 529–539.

[18] N. Gurjar, S. Sudholt, and G. Fink, "Learning deep representations for word spotting under weak supervision," in *Int. Workshop Doc. Anal. Syst.*, 2018, to appear.

[19] J. Rodriguez-Serrano and F. Perronnin, "Synthesizing queries for handwritten word image retrieval," *Pattern Recognit.*, vol. 45, no. 9, pp. 3270–3276, September 2012.

[20] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan, "Fast human detection using a cascade of histograms of oriented gradients," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2, 2006, pp. 1491–1498.

[21] M. Jones and D. Mewhort, "Case-sensitive letter and bigram frequency counts from large-scale english corpora," *Behav. Res. Methods Instrum. Comput.*, vol. 36, no. 3, pp. 388–396, August 2004.

[22] B. Leibe, A. Leonardis, and B. Schiele, "Robust object detection with interleaved categorization and segmentation," *Int. J. Comput. Vis.*, vol. 77, no. 1–3, pp. 259–289, May 2008.

[23] L. Gómez and D. Karatzas, "Textproposals: A text-specific selective search algorithm for word spotting in the wild," *Pattern Recognit.*, vol. 70, no. Supplement C, pp. 60–74, October 2017.

[24] A. Vedaldi and A. Zisserman, "Efficient additive kernels via explicit feature maps," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 3, pp. 480–492, March 2012.

[25] F. Moosmann, E. Nowak, and F. Jurie, "Randomized clustering forests for image classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 3, pp. 1632–1646, March 2008.

[26] F. Perronnin and C. Dance, "Fisher kernels on visual vocabularies for image categorization," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2007, pp. 1–8.

[27] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 9, pp. 1704–1716, September 2012.

[28] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2, 2006, pp. 2161–2168.

[29] V. Lavrenko, T. Rath, and R. Manmatha, "Holistic word recognition for handwritten historical documents," in *Proc. IEEE Int. Workshop Doc. Image Anal. Libr.*, 2004, pp. 278–287.