Efficient Exemplar Word Spotting

Jon Almazán
www.cvc.uab.es/~almazan
Albert Gordo
agordo@cvc.uab.es
Alicia Fornés
afornes@cvc.uab.es
Ernest Valveny
ernest@cvc.uab.es

Computer Vision Center
Departament de Ciències de la Computació
Universitat Autònoma de Barcelona
Barcelona, Spain

Abstract

In this paper we propose an unsupervised segmentation-free method for word spotting in document images. Documents are represented with a grid of HOG descriptors, and a sliding window approach is used to locate the document regions that are most similar to the query. We use the Exemplar SVM framework to produce a better representation of the query in an unsupervised way. Finally, the document descriptors are precomputed and compressed with Product Quantization. This offers two advantages: first, a large number of documents can be kept in RAM memory at the same time. Second, the sliding window becomes significantly faster since distances between quantized HOG descriptors can be precomputed. Our results significantly outperform other segmentation-free methods in the literature, both in accuracy and in speed and memory usage.

1 Introduction

This paper addresses the problem of word spotting: given a dataset of document images and a query word – usually cropped from one of the documents – , the goal is to identify and retrieve regions of those documents where the query word may be present. We focus on the unsupervised word spotting problem, where no labeled data is available for training purposes. This is a very common scenario since correctly labeling data is a very costly task.

Traditionally, word spotting systems have followed a well defined flow. First, an initial layout analysis is performed to segment word candidates. Then, the extracted candidates are represented as sequences of features [14, 17, 22]. Finally, by using a similarity measure – commonly a Dynamic Time Warping (DTW) or a Hidden Markov Model (HMM)-based similarity –, the query word is compared and candidates are ranked according to this similarity. Examples of this framework are the works of Rath and Manmatha [16] and Rodríguez-Serrano and Perronnin [18]. One of the main drawbacks of these systems is that they need to perform a costly and error prone segmentation step to select candidate windows. Any error introduced in this step will negatively affect the following stages, and so it is desirable to avoid segmenting the image whenever possible. Unfortunately, since the comparison of window regions, represented by sequences, is based on costly approaches such as a DTW

http://dx.doi.org/10.5244/C.26.67

or a HMM, it is not feasible to perform this comparison exhaustively with a sliding window approach over the whole document image.

Late works on word spotting have proposed methods that do not require a precise word segmentation, or, in some cases, no segmentation at all. The recent works of [7, 8] propose methods that relax the segmentation problem by requiring only a segmentation at the text line level. One of their drawbacks, however, is that they require a large amount of annotated data to train the Hidden Markov Models [7] or the Neural Networks [8] that they use. Additionally, the line segmentation still has to be very precise to properly encode the lines.

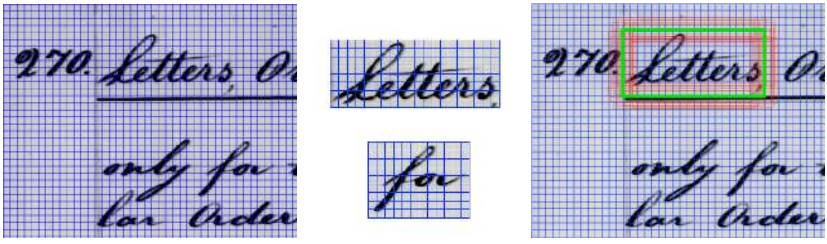
In [9], Gatos and Pratikakis perform a fast and very coarse segmentation of the page to detect salient text regions. Queries are represented with a descriptor based on the density of the image patches. Different transformations are applied to the queries to capture some variability. Then, a sliding window search is performed only over the salient regions of the documents using an expensive template-based matching.

The recent [19] addresses the segmentation problem by representing regions with a fixed-length descriptor based on the well-known bag of visual words framework [3]. In this case, comparison of regions is much faster since a dot-product or Euclidean distance can be used, making a sliding window over the whole image feasible. To further improve the system, unlabeled training data is used to learn a LSI space, where the distance between words is more meaningful than in the original space.

We argue that the current unsupervised word spotting methods can be improved in several ways, and we focus on the segmentation-free family of approaches. **First, they can be improved in the choice of low level features.** The features of [14, 17, 22] produce sequence representations, which are usually slower to compare than fixed-length representations. The work of [9] uses a descriptor based on the patch density, which is insufficient to capture all the fine-grained details. The bag of visual words approach of [19] showed very promising results. One drawback, however, is that the window size cannot be trivially adapted to the length of the query. As noted by the authors, the performance of the method is very dependent on the length of the query with respect to the fixed size of the window. We address these issues by using HOG descriptors [4], which have been shown to obtain excellent results when dealing with large variations in the difficult tasks of object detection and image retrieval. The document images are represented with a grid of HOG descriptors, all of them of the same cell size (*cf*. Fig 1(a)). In this case, we do not have a fixed window size; instead, the window size is adjusted to the query size, covering several HOG cells (*cf*. Fig 1(b)).

Second, spotting methods can be improved in the learning of a more semantic space. In [19], LSI is used to learn a latent space where words and documents are more related. However, learning a semantic space with LSI may be too conditioned to the words used in the unsupervised training stage, and adapting to new, unseen words may be complicated. Instead, we propose to perform this unsupervised learning once the query has been observed, and adapt the learning to the particular query. For this task, we propose to use a similar approach to the Exemplar SVM framework of [13, 21]. Although we focus on a completely unsupervised case, this framework could take advantage of annotated data, *e.g*., if several instances of the query word were available.

Finally, these methods can be improved in the cost of storing the descriptors of all the possible windows of all the dataset items. Assuming HOG descriptors of 31 dimensions represented with single-precision floats of 4 bytes each (*i.e*., 124 bytes per HOG), and 50,000 cells per image, storing as few as 1,000 precomputed dataset images would require 5.8GB of RAM. Since documents will not fit in RAM memory when dealing with large collections, we are left with two unsatisfying options: either recomputing the descriptors of



(a) HOG grid. (b) (c)

Figure 1: a) Grid of HOG cells. Only one small part of the image is shown. b) Two random queries of the George Washington dataset. The windows adjust around the HOG cells. c) A query (in green) and some positive samples (in red) used to learn the Exemplar SVM. To avoid clutter, only some of the positive windows are shown.

every document with every new query, or loading them sequentially from a hard disk or a solid-state drive (SSD). Any of these approaches would produce a huge performance drop in the speed at query time[1]. To address this problem, we propose to encode the HOG descriptors using Product Quantization (PQ) [11]. Encoding the descriptors with PQ would allow us to preserve a much larger amount of images in RAM at the same time. As a side effect, computing the scores of the sliding window also becomes significantly faster.

Therefore, the contribution of this paper is threefold. First, we show that a sliding window approach on top of a HOG grid can be used to solve word spotting tasks in a very natural way. Second, we show that the Exemplar SVM framework [13, 21] can be used to learn, in an unsupervised way, a better representation of the word queries. Although this training is performed at query time, it only has to be done once per query, and its cost is negligible if the number of elements in the dataset is large. Last, we propose to compress the HOG descriptors by means of PQ. With compression, document descriptors can be precomputed and stored in memory, instead of being calculated at query time or retrieved from a hard disk.

The rest of the paper is organized as follows. Section 2 proposes to use a grid of HOG descriptors and sliding window search for word spotting tasks. This will be our baseline system through the paper. Section 3 extends it to make use of the Exemplar SVM framework. Section 4 introduces the use of PQ to compress the HOG descriptors of the document. Finally, Section 5 deals with the experimental validation and Section 6 concludes the paper.

2 Baseline System

Word spotting – and, particularly, handwritten word spotting – is a challenging problem for descriptors because they have to deal with many kinds of deformations, variability in the styles, etc. For this reason, we propose to use HOG descriptors [4], which have been shown to obtain excellent results in the difficult tasks of object detection and image retrieval (see, *e.g.*, [6] or [21]).

In our baseline system, the document images are divided in equal-sized cells (see Fig 1a) and represented with HOG histograms. We follow [6] and use HOG histograms of 31 dimensions to represent each cell. Queries are represented analogously using cells of the same size in pixels. Note that the number of cells in a query, and therefore the final dimensionality

[1] A similar point can be argued about the methods of [9] or [19].

of the descriptor, depends on the size of the query image. The score of a document region can be calculated as the convolution of the query with respect to that region, using the dot-product as a similarity measure between the HOG descriptors of each cell. This can also be understood as calculating the dot-product between the concatenated HOGs of the query ($\mathbf{q}$) and the concatenated HOGs of the document region ($\mathbf{y}$), *i.e.*, $\text{sim\_dp}(\mathbf{q},\mathbf{y}) = \mathbf{q}^T\mathbf{y}$, although we perform a convolution instead of concatenating the descriptors explicitly. Following this approach, we can compute the similarity of all the regions in the document image with respect to the query using a sliding window and rank the results. Non-Maximum Suppression (NMS) is performed to remove overlapping windows.

We further modify the baseline in two ways. First, instead of using the HOG descriptors directly, we reduce their dimensionality down to 16 dimensions with PCA. We observed no loss in accuracy because of this, probably because of the "simplicity" of text patches. Second, instead of calculating the dot-product, we are interested in the cosine similarity, *i.e.*, calculating the dot-product between the L2 normalized descriptors. The cosine similarity is a typical choice in document retrieval, and we observed experimentally that L2 normalizing the vectors can indeed make a significant difference in the results. Note that the L2 normalization is performed at the region level, not at the cell level. Fortunately, we do not need to explicitly reconstruct the regions to normalize the descriptors, since $\text{sim\_cos}(\mathbf{q},\mathbf{y}) = (\frac{\mathbf{q}}{||\mathbf{q}||})^T(\frac{\mathbf{y}}{||\mathbf{y}||}) = \frac{1}{||\mathbf{q}||}\frac{1}{||\mathbf{y}||}\mathbf{q}^T\mathbf{y} = \frac{1}{||\mathbf{q}||}\frac{1}{||\mathbf{y}||}\text{sim\_dp}(\mathbf{q},\mathbf{y})$. Therefore, we can calculate the $\text{sim\_dp}(\mathbf{q},\mathbf{y})$ score with a convolution without explicitly concatenating the descriptors and later normalize it with the norms of the query and the document region. The norm of the query can in fact be ignored since it will be constant for all the document regions and therefore does not alter the ranking. As for the region patch, we can accumulate the squared values while performing the convolution to construct, *online*, the norm of the region patch without explicitly reconstructing it.

3 Exemplar Word Spotting (EWS)

We note that the cosine similarity may not be the optimal way to compare document regions, and that learning a better metric may yield significant improvements. In [19], this is achieved by learning, *offline*, a latent space through LSI, where the cosine similarity is an appropriate measure. Unfortunately, it is not clear how we could adapt this latent learning to our grid of HOGs framework.

Here we take a different approach and propose to learn, *at query time*, a new representation of the query, optimized to maximize the score of regions relevant to the query when using the dot-product. This new representation can be understood as *weighting* the dimensions of the region descriptors that are relevant to the query. We achieve this goal by means of the Exemplar SVM framework [13, 21]. Let us assume that we have access to a set $\mathcal{P}$ of regions that are relevant to the query. These are described as the concatenation of their PCA-compressed HOG descriptors, and are L2 normalized. Analogously, let us assume that we have access to a set $\mathcal{N}$ of regions that are not relevant to the query. In this case, we can solve the following optimization problem

$$\underset{\mathbf{w}}{\text{argmin}}\ \frac{1}{2}||\mathbf{w}||^2 + C\sum_{\mathbf{y}_p\in\mathcal{P}} L(\mathbf{w}^T\mathbf{y}_p) + C\sum_{\mathbf{y}_n\in\mathcal{N}} L(-\mathbf{w}^T\mathbf{y}_n), \tag{1}$$

where $L(\mathbf{x}) = \max(0, 1-\mathbf{x})$ is the hinge loss and C is the cost parameter. Solving this

optimization produces a weight vector $\mathbf{w}$, which can be seen as a new representation of the query. This new representation has been directly optimized to give a high positive score to relevant regions, and a high negative score to non-relevant regions when using the dot-product with L2 normalized regions. As in the baseline system, we can rearrange the terms so that $\text{sim\_dp}(\mathbf{w}, \frac{\mathbf{y}}{||\mathbf{y}||}) = \frac{1}{||\mathbf{y}||}\text{sim\_dp}(\mathbf{w},\mathbf{y})$, where $\text{sim\_dp}(\mathbf{w},\mathbf{y})$ can be calculated without reconstructing the region vectors and $||\mathbf{y}||$ can be calculated online while performing the convolution.

Unfortunately, in most cases we will not have access to labeled data, and so $\mathcal{P}$ and $\mathcal{N}$ will be unavailable. To overcome this problem, $\mathcal{P}$ is constructed by deforming the query, similarly to what is done in [21]. In our case, we slightly shift the window around the query word to produce many almost identical, shifted positive samples (see Fig 1(c)). As a side effect, at test time, sliding windows that are not completely centered over a word will still produce a high score, making the approach more resilient to the sliding window granularity. To produce the negative set $\mathcal{N}$, we sample random regions over all the documents. Note that, since we do not have access to segmented words, we can not guarantee that a given negative region will contain a complete word or a word at all. This is different from unsupervised methods that perform word segmentation such as that of [18]: even if they do not use labeled data, they have access to the bounding boxes of training words. Finally, as in [21], positive samples could also appear in this randomly chosen negatives set.

4 Feature Compression

One important drawback of sliding-window based methods such as this or [9, 19] is the cost of recomputing the descriptors of every image with every new query. As we will see during the experimental evaluation (*cf*. Table 1, Section 5), computing the HOG descriptors of an image can take between 50% and 90% of the total test time per document, and this has to be recomputed for every new query. Precomputing and storing the HOG descriptors is usually not a feasible option because of the large amount of memory that would be necessary to maintain them: as noted in the introduction, storing only one of our documents would require approximately 6MB of memory. Even if we compress the HOG descriptors with PCA down to 16 dimensions, we can barely fit 330 documents in 1GB of Memory. Since documents have to be kept in RAM to be rapidly accessed, storing large collections of documents would be extremely expensive or directly unfeasible.

In this section we propose to encode the HOG descriptors by means of Product Quantization (PQ) [11]. This technique has shown excellent results on approximate nearest neighbor tasks [10, 12], maintaining a high accuracy while drastically reducing the size of the signatures. In our case, we will reduce the size of the HOG descriptors down to one single byte, achieving a compression ratio of 64:1 with minimal loss. After PQ compression, we can store up to 25,000 images per GB or RAM. As a side effect, because of the dimensionality reduction, the sliding window comparisons will also be faster. Without PQ, on one of our best performing configurations, we can analyze approximately 7 documents per second. After PQ, we can analyze approximately 67 document images per second, an almost 10 fold improvement in speed.

In the following section we will first give an overview of PQ, and then we will describe how PQ fits in our system, both in the exemplar training and the sliding window stages.

4.1 Product Quantization

In vector quantization the goal is to find a finite number of reproduction values $\mathcal{C} = \{\mathbf{c}_1, \dots \mathbf{c}_k\}$ (usually through k-means), and then represent the vector with the the closest reproduction value, *i.e.*, given a vector $\mathbf{x} \in \mathbb{R}^D$ and a quantizer q, then

$$q(\mathbf{x}) = \underset{\mathbf{c}_i \in \mathcal{C}}{\operatorname{argmin}}\, d(\mathbf{x}, \mathbf{c}_i), \tag{2}$$

where d is usually the Euclidean distance. Assuming k centroids, then quantized vectors can be encoded using as few as $\log_2 k$ bits by storing the indices of the centroids and not the centroids themselves, which are kept in a different table. Unfortunately, using vector quantization is not possible if the dimensionality of the vectors is not trivially low: as noted in [11], to encode a descriptor of 128 dimensions using only 0.5 bits per dimension, we would need to compute 2^{64} centroids, which is not feasible.

PQ addresses this issue by quantizing groups of dimensions independently. Vectors are split into m groups of D/m dimensions, and quantizers are learned independently producing $m \times k$ centroids. If we denote with $\mathbf{x}^j$ the j-th group of vector $\mathbf{x}$, and with $\mathbf{c}_{ji} \in \mathcal{C}_j$ the i-th centroid learned from group j, then

$$q_j(\mathbf{x}) = \underset{\mathbf{c}_{ji} \in \mathcal{C}_j}{\operatorname{argmin}}\; d(\mathbf{x}^j, \mathbf{c}_{ji}), \tag{3}$$

and $q(\mathbf{x}) = \{q_1(\mathbf{x}), q_2(\mathbf{x}), \dots, q_m(\mathbf{x})\}$. In this case, to produce a b bits code, each quantizer needs to compute only $2^{b/m}$ centroids, which is reasonable if m is chosen appropriately.

One advantage of PQ is that to compute the distance between a query $\mathbf{x}$ and a (quantized) document y from a dataset – represented by m indices to the centroids table –, it is not necessary to quantize the query or to explicitly decode the quantized document. We can, at query time, precalculate a look-up table $\ell(j,i) = d(\mathbf{x}^j, \mathbf{c}_{ji})$. Note that this table does not depend on the number of elements of the dataset, only on the number of groups m and centroids k, and so the cost of computing it is negligible if the number of documents is large. Once this look-up table has been constructed, we can calculate the distance between query $\mathbf{x}$ and a quantized document y as $\sum_{j=1}^{m} \ell(j, y_j)$, where y_j is the j-th index of y, without explicitly reconstructing the document.

In our case, we use PQ to encode our PCA-HOG descriptors of 16 dimensions. For performance reasons, we decided to apply PQ using one single group of 16 dimensions, and use 256 centroids, *i.e.*, 8 bits to represent it, 0.5 bits per dimension. However, if accuracy is an issue, we can split the vectors in 2 groups of 8 dimensions. In this case the accuracy loss is negligible and we can still obtain a 31-fold compression rate with respect to the 16-dimensional descriptors.

To learn the Exemplar SVM, we need to create the sets $\mathcal{P}$ and $\mathcal{N}$. As seen in Figure 1(c), positive samples do not exactly align with the precomputed cells, and therefore we need to recompute the descriptors for those regions. Note that this is a small amount of descriptors compared to the whole image. The negative samples, however, can be chosen so that they align with the precomputed grid of HOGs, decoded, and then fed to the SVM training method. Similarly to [20], we decode the features and learn our classifier on the decoded data. When calculating the sliding window, we no longer have to compute the dot-product between the query and the HOG descriptor to obtain the score of a cell, and it is enough to perform one single access to the look-up table to obtain that score. Intuitively, this could lead to up to a 16 fold speed-up with respect to the uncompressed HOG descriptors. In practice, we have obtained up to 10 fold speed-ups.

5 Experiments

We evaluate our approach on two public datasets: The George Washington (GW) dataset [15, 16] and the Lord Byron (LB) [19][2] dataset. These datasets are comprised of 20 pages each and contain approximately 5,000 words. George Washington contains handwritten text while Lord Byron only contains typewritten text. We use the same evaluation protocol as [19], and so our results are directly comparable. Each word is considered as a query and used to rank all the regions of every document in the dataset. A region is classified as positive if it overlaps more than 50% with the annotated bounding box in the groundtruth and negative otherwise. For every document, we keep only the 1,000 regions with the highest score and perform NMS to avoid overlaps of more than 20%. As in [19], the query image, if retrieved, is considered a true positive. Finally, we combine the retrieved regions of all the documents and rerank them according to their score. We report the mean Average Precision, which is a standard measure in retrieval systems and can be understood as the area below the precision-recall curve.

To compute the HOG grid, we use cells of sizes 8, 12, and 16 pixels, and analyze the effect of the cell size both in terms of the accuracy and the time to perform the retrieval. The cell size also affects the number of cells per image, and therefore smaller cells will require more memory if we want to precompute them. On our datasets, each document image contains approximately 25,000 cells when using a cell size of 16, 45,000 when using a cell size of 12, and 100,000 when using a cell size of 8. We also vary the step size of the sliding window between one and two cells, since this can noticeably affect both the accuracy and the temporal cost of the system. When performing the unsupervised learning of PCA and PQ, we randomly sample 10,000 HOGs from all the documents. When learning the Exemplar SVM, we used the LIBLINEAR package [5] and fixed the C trade-off to 0.01. This provided reasonable results on both datasets. Note that, if validation sets were available, this parameter could be fine-tuned to obtain better results on both datasets. We produce 121 positive samples for each query (11 shifts in horizontal $\times$ 11 shifts in vertical) and 7,744 negative samples (64 times the number of positive samples). Increasing the number of negative samples led to a very slight improvement in the accuracy, but we did not consider it worth the extra cost during training. Note that [13, 21] propose to use several iterations of hard negative mining. However, we did not experience any gain in accuracy by doing so.

We used MATLAB's built-in profiler to measure the running times of the different sections of our pipeline on an Intel Xeon running at 2.67GHz using one single core. We noticed that the profiler added approximately a 10% overhead that we have not subtracted. Therefore, the actual times may be slightly faster than reported. Note that, although we used MATLAB, the core sections (computing the HOG descriptors, training the SVM, calculating the scores with a sliding window, and the NMS) were implemented in C. When reporting the sliding window times, these already include the time to perform the ranking and the NMS.

5.1 Results and Discussion

Figure 2 shows the mean Average Precision of our vanilla method and its improvements – EWS and EWS+PQ – on the GW and LB datasets as a function of the HOG cell size and the sliding window step size. Table 1 reports running times of the core sections on GW. We did not observe different trends between the times of GW and LB. We can highlight the

[2] We obtained the exact images and groundtruth after direct communication with the authors of [19].

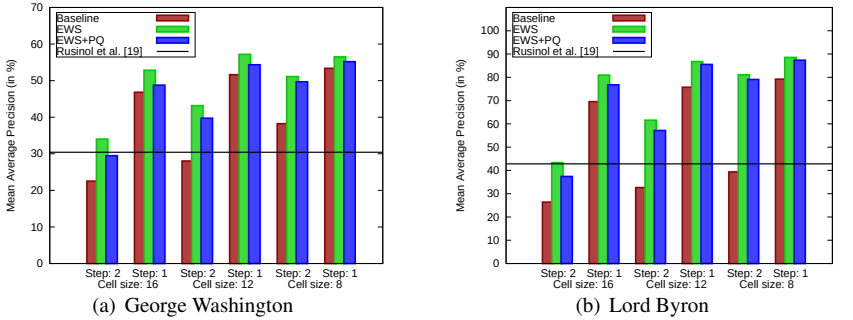
following points:

Influence of the cell size and the step size: On both datasets, reducing the cell size from 16 to 12 pixels significantly improves the accuracy, particularly when using a step size of 2. Reducing the cell size down to 8 pixels can yield another small improvement. This is due to the fact that we have more cells in each query word, and hence more information, but also because the granularity of the sliding window becomes finer when the cell size is smaller. Note that decreasing the cell size negatively affects the times of all the stages as well as the memory required to store the images.

Influence of the Exemplar SVM: In all cases learning a good representation of the query improves the results. The improvement is particularly acute when using a step size of 2, since we have learned how to retrieve slightly shifted windows. Although there is an overhead at query time because of the training of the SVM, this has to be learned only once per query, and the cost can be acceptable if the number of documents in the dataset is large.

Influence of PQ: After PQ, the accuracy of the methods suffers a small drop. However, the sliding window times become between 4 and 10 times faster, depending on the configuration, and a much larger number of documents can fit in memory at the same time.

When aiming to the highest accuracy, it makes sense to use the smallest possible cell size and no PQ. However, when considering the time and memory constrains, using slightly larger cell sizes (*e.g.* 12) and PQ seems a reasonable trade-off. With this configuration, we obtain a 54.5% mAP on GW and a 85.5% mAP on LB, needing approximately 15ms per document. These numbers compare very favorably to the 30.5% and 42.8% reported in [19], with times of about 340ms per document on an unoptimized MATLAB+python implementation.



(a) George Washington (b) Lord Byron

Figure 2: Mean Average Precision of our method for different setups.

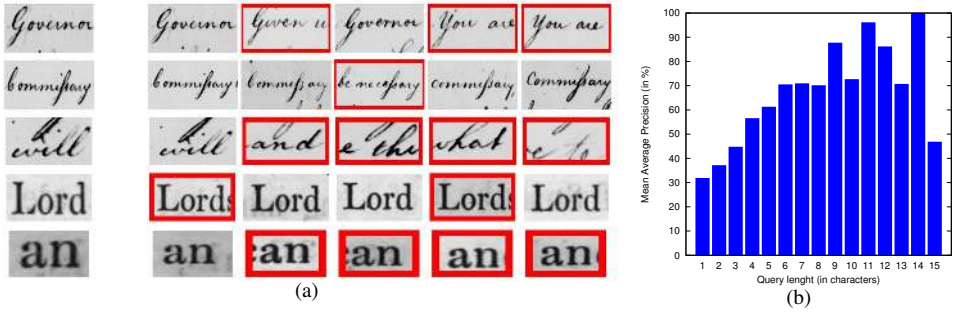
Bin size (pixels):	16		12		8	
Step size (cells):	2	1	2	1	2	1
Train Exemplar SVM (ms / query)	560		1,090		2,430	
Extract HOGs (ms / doc)	467		503		528	
Sliding Window (ms / doc)	15	40	39	130	177	655
Sliding Window with PQ (ms / doc)	2	8	6	15	20	73

Table 1: Measured times of the core components of our approach for different configurations on the GW dataset.

For the sake of completeness, we also compare our results with those of [15] and [18]. The authors of [15] report results between 52% and 65% on the GW dataset depending on the particular fold they evaluate with, while [18] reports a 54%. Note however that these

results are not completely comparable since i) they use different partitions, and ii) they work on already segmented words, *i.e.*, they are not segmentation-free. Moreover, [18] does not include the query as a positive sample, which we do to keep compatibility with [19]. It is reasonable to expect the results of [18] to improve if they took the query into account. This suggests that DTW or HMM based methods can produce excellent results when paired with good features as in [18]. Although they are too slow to be used over the whole document page, they could be used to rerank the best regions returned by a sliding-window method such as ours.

Finally, Figure 3(a) illustrates some typical failure cases, such as confusions with similar-shaped words, giving too much weight to artifacts in the query word, or retrieving substrings from longer words. This second problem in particular is one drawback of the segmentation-free, sliding-window methods since, as opposed to DTW, do not penalize matching a short word with a substring of a long word. As observed in Figure 3(b), this can very significantly reduce the mAP of short queries.



(a) (b)

Figure 3: a) Failure cases. First two rows: words have a very similar shape. Third row: an artifact in the query leads to results with the same artifact. Fourth and fifth rows: we detect the query word as a substring of a longer word. This is common when querying short words. b) Mean Average Precision on the GW dataset as a function of the query length.

6 Conclusions and Future Work

In this paper we have shown how a combination of HOG descriptors and sliding windows can be used to perform unsupervised word spotting, both on handwritten and machine-printed text. This method can be extended using Exemplar SVMs to represent the queries, improving the results at a minimum extra cost at query time. Finally, we have shown how the HOG descriptors can be aggressively compressed with Product Quantization with only a small loss in accuracy. We have obtained excellent results when comparing to other segmentation-free methods in the literature. We have published the MATLAB code implementation for training and testing the Exemplar Word Spotting on the web page [1], in the hope that it would provide a basis for new papers on word spotting.

We would like to note that the variability of the writing style on the datasets that have been used in [19] and here is very small. This variability is usually much higher in a multi-writer scenario. For example, two writers may write the same word with very different widths, and only one window size will not be able to correctly capture both of them. We believe that, as presented, our method would have difficulties in this scenario. To overcome

this problem we propose as future work to severely deform the query varying the stretch and the slant, and learn several Exemplar SVMs as in [13]. This would increase the query time, but not the memory required to store the datasets. As a consequence of this, the training time of the SVMs may become prohibitive even with fast batch solvers such as LIBLINEAR. Therefore, we also want to study the effect of Stochastic Gradient Descent-based solvers such as [2], which would considerably speed up the training process.

Acknowledgment

This work has been partially supported by the Spanish projects TIN2011-24631, TIN2009-14633-C03-03 and CSD2007-00018, by the EU project ERC-2010-AdG-20100407-269796 and by two research grants of the UAB (471-01-8/09).

References

[1] J. Almazán, A. Gordo, A. Fornés, and E. Valveny. Exemplar Word Spotting library. URL http://almazan.github.com/ews/.

[2] L. Bottou. Stochastic Gradient Descent project. URL http://leon.bottou.org/projects/sgd.

[3] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Workshop on Statistical Learning in Computer Vision, ECCV*, 2004.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[5] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 2008.

[6] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramaman. Object detection with discriminatively trained part based models. *IEEE TPAMI*, 2010.

[7] A. Fischer, A. Keller, V. Frinken, and H. Bunke. HMM-based word spotting in handwritten documents using subword models. In *ICPR*, 2010.

[8] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke. A novel word spotting method based on recurrent neural networks. *IEEE TPAMI*, 2012.

[9] B. Gatos and I. Pratikakis. Segmentation-free word spotting in historical printed documents. In *ICDAR*, 2009.

[10] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010.

[11] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 2011.

[12] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: re-rank with source coding. In *ICASSP*, 2011.

[13] T. Malisiewicz, A. Gupta, and A. Efros. Ensemble of Exemplar-SVMs for object detection and beyond. In *ICCV*, 2011.

[14] U.-V. Marti and H. Bunke. Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition systems. *IJPRAI*, 2001.

[15] T. Rath and R. Manmatha. Word image matching using dynamic time warping. In *CVPR*, 2003.

[16] T. Rath and R. Manmatha. Word spotting for historical documents. *IJDAR*, 2007.

[17] J. Rodríguez-Serrano and F. Perronnin. Local gradient histogram features for word spotting in unconstrained handwritten documents. In *ICFHR*, 2008.

[18] J. Rodríguez-Serrano and F. Perronnin. A model-based sequence similarity with application to handwritten word-spotting. *IEEE TPAMI*, 2012.

[19] M. Rusiñol, D. Aldavert, R. Toledo, and J. Lladós. Browsing heterogeneous document collections by a segmentation-free word spotting method. In *ICDAR*, 2011.

[20] J. Sánchez and F. Perronnin. High-dimensional signautre compression for large-scale image classification. In *CVPR*, 2011.

[21] A. Shrivastava, T. Malisiewicz, A. Gupta, and A. Efros. Data-driven visual similarity for cross-domain image matching. In *Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH Asia)*, 2011.

[22] A. Vinciarelli, S. Bengio, and H. Bunke. Offline recognition of unconstrained handwritten texts using HMMs and statistical language models. *IEEE TPAMI*, 2004.