Zoning Aggregated Hypercolumns for Keyword Spotting

Giorgos Sfikas, George Retsinas, Basilis Gatos
Computational Intelligence Laboratory,
Institute of Informatics and Telecommunications,
National Center for Scientific Research Demokritos,
GR-15310 Agia Paraskevi, Athens, Greece
{*sfikas, georgeretsi, bgat*}*@iit.demokritos.gr*

***Abstract*—In this paper we present a novel descriptor and method for segmentation-based keyword spotting. We introduce Zoning-Aggregated Hypercolumn features as pixel-level cues for document images. Motivated by recent research in machine vision, we use an appropriately pretrained convolutional network as a feature extraction tool. The resulting local cues are subsequently aggregated to form word-level fixed-length descriptors. Encoding is computationally inexpensive and does not require learning a separate feature generative model, in contrast to other widely used encoding methods (such as Fisher Vectors). Keyword spotting trials on machine-printed and handwritten documents show that the proposed model gives very competitive results.**

I. Introduction

In cases where full recognition is not necessary or high-quality recognition is not feasible, keyword spotting techniques allow the end-user to search a document for instances of a specific word [1], [2]. Keyword spotting and recognition, especially in handwritten documents, remain significant challenges compared to other forms of text, albeit also important recent advances [2]. The same is true for other document understanding actions such as layout analysis or text segmentation. Writing style variance and cursiveness in the documents of a single author, and variance between styles of different authors are important problems one has to face in processing of handwritten text, not found in machine-printed documents. Older handwritten manuscripts are related with extra difficulties due to degradations in quality of the digitized document, making all document understanding operations even more difficult.

Segmentation of a document in text components has been used in many document understanding systems as a basic pre-processing step. Text components that are used routinely as the target element of segmentation algorithms are text lines and text words. Word spotting can then be formulated as an image retrieval problem when the query is a word image. Another vein of techniques assume that the user uses a word string as a query. The two approaches are known as Query-by-Example (QbE) and Query-by-String (QbS) in the literature [2].

In this work, a novel QbE, segmentation-based keyword spotting method is proposed. We use a deep Convolutional Neural Network (CNN) pretrained for a character classification task [3]. Instead of using the CNN for the task it was originally trained for, i.e. character classification, we use it as an off-the-shelf feature extractor. It has been shown that per-layer activations can act as efficient local descriptors [4], [5]. The pool of resulting convolutional features is aggregated to a single descriptor per word image. Aggregation is done by combining simple sum-pooling, that has been recently demonstrated to be an appropriate encoding technique for convolutional features [6]. We combine this simple aggregation model with a zoning scheme, suitable for word images, to create the fixed-length word-level feature vector. We shall refer to this word-level descriptor as a ”Zoning-Aggregated Hypercolumns” descriptor (ZAH). Querying with the proposed descriptor is performed by nearest-neighbour search in the (Euclidean) descriptor space. Numerical results show that our approach leads to competitive KWS results.

The outline of the rest of this paper is as follows. In section II we discuss related works in the literature. In section III we present the proposed method in detail. In section IV we present numerical results comparing our method to other segmentation-based word spotting methods. In section V we present final conclusions and thoughts about future work.

II. Related work

Keyword spotting can be seen as a special form of an image retrieval problem. Like in image retrieval, suitable descriptors have to be created for the query and each word image in the document to be searched. As in all image understanding tasks, features matter, and powerful features have been used to build good descriptors for word spotting and recognition tasks. Such features range from low-level column-based profiles to more elaborate shape-based or patch-based features [1], [7], [2]. Given a dense set of feature vectors per image, matching is performed either by dynamic programming [7], direct comparison using a suitable metric (often the Euclidean) [1] or using an encoding technique first

[1] The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation Programme (H2020-EINFRA-2014-2015) under grant agreements n 674943 project READ.

2167-6445/16 $31.00 © 2016 IEEE
DOI 10.1109/ICFHR.2016.56

[8] before comparing the encoded vectors. Various state-of-the-art models that are based on the encoding of patch-based SIFT or HOG features have been proposed [2], [9], [8].

Interest in neural networks has recently been rekindled, with much effort being put to exploring their applicability in various tasks that include word spotting and recognition [10], [3]. Using activations of fully-connected layers given a pretrained convolutional network and an unseen image has been proved to be a powerful image feature extraction technique [4]. Activations of convolutional layers have also been used in this manner. The latter have been referred to as deep convolutional features or deep patch-based features [6] as opposed to shallow patch-based features (SIFT, HOG, etc.). In [5], combinations of convnet layers are concatenated into a single vector dubbed a ”hypercolumn”. Encoding of convolutional features can be efficiently performed [11], [6]. A simple sum-pooling operation of convolutional features has been shown to be appropriate to encode them into a single, powerful descriptor (Sum-pooled convolutional features, SPoC) [6]. This is an advantage over other encoders such as the state-of-the-art Fisher vectors or VLAD, which are comparatively more complex to compute. The former require fitting training data to a generative model with unsupervised learning. This process can be expensive and may also be a source of errors for the processing layers that follow in the event that the generative model is poorly estimated.

Motivated by the aforementioned results in machine vision, we propose a CNN-based feature extraction and encoding scheme suitable for keyword spotting. In our work, each word image is feed-forwarded through the CNN. The result, after proper zero-padding of the input, is a pixel-level map of hidden layer activations, that are aggregated in zones to form a single word image descriptor. Encoding is easy and fast since it only requires a sum-pooling operation.

Regarding to which of the layers should we use to extract activations and construct features, it is well-known that the deepest the layer, the lower-level and less task-specific and more general information it encodes [12]. In [12] for example, where a convnet trained for classification on ImageNet is studied, it is shown that the first layers are practically edge and corner detectors. Subsequent layers detect more complex patterns such as texture. Layers close to the output detect very high-level structure, like images of dogs, keyboards, and other classes that are specific to ImageNet.

In the current work we use convnets trained to classify characters [3], in order to extract features. Concatenating activations of multiple layers has been succesfully tried, where concatenated vectors are dubbed hypercolumns [5]. Inspired by the work of [5], we experiment in concatenating activations from multiple convolutional layers. As in the case of the ImageNet-trained convnet, we can expect that moving from deeper to shallower layers, corresponds to going from lower to higher-level detectors. Our character classifiers are pretrained mostly on ”street-view” text (fig. 2), which can differ significantly from our test set (fig. 3). For this reason we are not interested in using high-level activations, neither activations that capture too low-level information. Our experiments (sec. IV) confirm that intermediate layer activations are more important for the task in question.

The authors of [3] use their character classifier to extract outputs that are subsequently used for a different task (recognition). The word-level map they propose is based on the softmax response of the model instead of using the hidden layers as done in this work. The hypercolumns used in our work are much less task-specific and capture more abstract information than softmax outputs. This enables us to use the proposed model to perform KWS even on a script different than the one that the original network was pretrained with; we demonstrate this in section IV.

III. Proposed method

In this work, we propose a new, fixed-length descriptor for segmented word images. Each word image is first partitioned into a number of vertical zones. For each of the zones a set of local, pixel-level descriptors is extracted using an appropriate convolutional network. These pixel-level descriptors are called ”hypercolumns” in this work. These are subsequently aggregated and concatenated into a single, fixed-length, word-level descriptor. Keyword spotting is then expressed as nearest neighbour search in the space of the final descriptors. The processing pipeline is summarized in fig. 1.

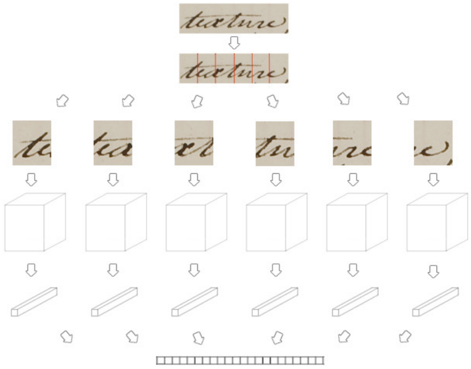


Figure 1. Processing pipeline to construct ZAH features. From top to bottom, we have: Input word image ; partition of image along its width into $Z = 6$ zones, with each zone covering an area of $h \times w_z$ pixels; extraction of pixel-level hypercolumns, for each zone a $h \times w_z \times d$, where d is hypercolumn dimensionality; aggregation of hypercolumns into a single $1 \times d$ hypercolumn per zone; concatenation of zone hypercolumns into a single $1 \times Zd$ vector.

We continue with describing the proposed pipeline steps in greater detail.

Zoning: The input is partitioned into Z vertical zones (fig. 1); we used $Z = 6$ in this work. Each zone overlaps its neighbouring zones by a width equal to a $\frac{1}{4Z}$ fraction of the total word width.

Setup of the pretrained CNN: To compute the proposed descriptor, we make use of a pretrained deep Convolutional Neural Network. We have used the character classifier CNNs of [3]. In principle, we can use any off-the-shelf CNN that has been trained to classify character images to character classes. We have used the case-insensitive CNN character classifier and the bigram classifier of [3], to which we shall refer here as unigram CNN and bigram CNN. These nets have been trained on a total of $186k$ and $92k$ samples respectively, coming from various "street-view" sources (see fig. 2).

Figure 2. Samples of training images that the CNNs we use in this work were trained with.

Feed-forward neural networks consist of a stack of layers, each one forwarding output to the layer that follows, until the output layer is reached. Consequently this applies also to convolutional networks, i.e. the nets we use. Input is pre-normalized so that the mean of the input is zero and its standard deviation equal to one. The architecture of the CNNs we have used in this work is as follows. Input is a box of 24×24 pixels, containing a single unigram or bigram. The output of the models contains 37 classes in the case of the unigram model (26 latin characters, 10 digits and a class for space) and 604 classes in the case of the bigram model. Intermediate layers are either sets of convolutional or fully-connected layers. We have three sets of convolutional layers topped by maxout layers, and one set of fully-connected layers topped by a maxout layer. The convolutional layers have window side of 9, 9, 8 pixels respectively. We shall refer to the maxout layers for each set as $conv1$, $conv2$, $conv3$ and $fc1$ correspondingly. In maxout layers a point-wise maximum over underlying layers of the same resolution is taken. In each maxout layer the maximum value is taken over tuples of 2, 4, 4, 4 underlying channels, respectively for $conv1$, $conv2$, $conv3$, $fc1$. Each maxout layer applies this operation over a number of input channels such that the output channels are 48, 64, 128 total channels respectively for the first three maxouts and 37(604) for the last maxout layer of the unigram(bigram) model. The last layer $fc1$ is followed by softmax activation to convert its values to a probability vector of class responsibilites.

Extracting Hypercolumns using the CNN: In test mode, the input can vary to any size larger than the size at training time. We resize input word images so that their height h is equal to a fixed number close to the original CNN input box size (we used 30 as a default value), and width w is such that the aspect ratio of the image remains fixed. With appropriate zero-padding, the convolutional network gives output as a $30 \times w$ map. The amount of zero-padding necessary depends on the number and size of the receptive fields of the net's convolutional layers. We zero-pad inputs with 11 zero pixels per dimension. The resulting map is of size $h \times w \times d$, where d is the total number of channels for all net layers. For the unigram model, this would be d = 48 + 64 + 128 + 37 = 277 (we do not take into account maxout inputs). We shall refer to this per-pixel concatenation of layer activations into a single vector as a "hypercolumn", following [5]. In practice, we do not concatenate all layers; concatenating a subset of all layers tends to be more efficient (cf. [5], and sec. IV in the current work).

Encoding Hypercolumns: Preprocessing and passing each word zone through the CNN gives Z $h \times w_z \times d$ maps of hypercolumns, one for each zone. [1]. We proceed by encoding each of the zone hypercolumns to a single descriptor per zone. We used sum-pooling for aggregation, formally:

$$d_z = \sum_{x \in W_z} \sum_{y \in H} v_{xy} \tag{1}$$

where d_z is the descriptor of zone d, set W_z contains all possible width values of zone z, set H contains all possible height values, and v_{xy} is the hypercolumn computed for the pixel with xy coordinates. We have also experimented using a center prior with a weighted-sum version of sum-pooling [6], formally

$$d_z = \sum_{x \in W_z} \sum_{y \in H} v_{xy} \exp^{-\frac{1}{2}\lambda(y-y_0)^2} \tag{2}$$

where λ is the precision (inverse variance) of the Gaussian kernel and y_0 is the height coordinate of the central horizontal axis of the word image window. Setting $\lambda = 0$ is equivalent to standard sum-pooling.

Zone-specific descriptors are then l2-normalized and concatenated to a single, fixed-length vector d:

$$d = [\frac{d_1}{||d_1||}, \frac{d_1}{||d_1||}, \cdots, \frac{d_Z}{||d_Z||}] \tag{3}$$

The full vector is itself subsequently l2-normalized [2].

IV. Numerical results

A. Document sets

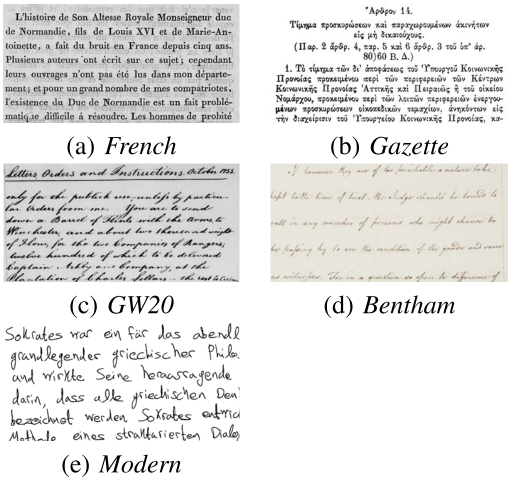
In order to test the proposed descriptor and method, we have ran a set of keyword spotting trials on various document datasets using a set of different model layouts. Our collections are two machine-printed and three hand-written sets, named here respectively "*French*" [1], "*Gazette*" [13][3]),

[1] We actually feed the whole word image to the CNN and then split into zones, which is more efficient; the two practices are in other respects equivalent.

[2] Code is available at https://github.com/sfikas/zah .

[3] This set is named "GR-POLY-DB-MachinePrinted-A" in its original publication.

"*GW20*" [14], "*Bentham*" [15] and "*Modern*" [15]. *French* is part of a book written in French, and consists of 10 pages segmented into 3258 word images [4]. *Gazette* consists of 5 pages segmented into 5004 word images and is written in polytonic Greek. *GW20* is the well-known collection of writings of George Washington, consisting of 20 pages segmented into 4860 words. *Bentham* consists of 50 pages segmented into 10648 words. *Modern* is a multi-writer, multi-script collection, consisting of 100 pages segmented into 14840 words. In figure 3 we show sample images from the collections.



(a) *French*

(b) *Gazette*

(c) *GW20*

(d) *Bentham*

(e) *Modern*

Figure 3. Samples from the document collections we used for evaluation in this paper. (a) French (machine-printed) (b) Gazette (machine-printed, greek script) (c) GW20 (handwritten) (d) Bentham (handwritten) (e) Modern (handwritten, multiple writers, multiple scripts)

B. Experiments

We have computed Mean-Average Precision (MAP) values for QbE trials on *French*, *Gazette* and *GW20*. MAP was computed as the mean over the result of querying selected images from the respective sets. Regarding sets *French* and *Gazette*, queries were selected according to a minimum length and minimum frequency criterion, following [15]. All instances of words with more than 5 letters and appearing more than 6 times in the set were used as queries. *trec_eval* was used to compute MAP. For trials in *GW20* we picked query instances in the same manner as [16]. Regarding sets *Bentham* and *Modern*, we used the automatic evaluation tool provided in the competitions site [15] to compute MAP and Precision at 5 (P@5) scores.

Choice of layers from which to extract features: We have run tests on *GW20*, each with a different choice of model parameters. We have experimented with using the three (maxout) convolutional layers and the (maxout) fully-connected hidden layer of the pretrained nets. We used the layer outputs by themselves, or concatenated in various combinations to a single hypercolumn [5]. We used the whole set for testing, and queries were chosen in the same manner as [16]. All query instances are also part of the test set. MAP results are reported in table I. We can conclude that the last convolutional layer ($conv3$) is the most important in providing a good result. In terms of compromising between descriptor calculation speed and efficiency, using $conv3$ or $conv2 + conv3$ is perhaps the best solution. Also, the bigram model gave in general better results than the unigram CNN. It is clear for both models that layer $conv3$, i.e. the last maxout/convolutional layer is the most important to include for good results. This result is consistent with our expectation that layer features coming from intermediate layers would be the more useful in our setup.

	MAP(%)	
Layer name	unigram CNN	bigram CNN
$conv1$	34.5	37.5
$conv2$	44.2	47.2
$conv3$	47.6	50.9
$fc1$	40.4	46.1
$conv1$+$conv2$	44.5	46.9
$conv2$+$conv3$	50.3	50.9
$conv1$+$conv2$+$conv3$	47.7	50.9
$conv1$+$conv2$+$conv3$+$fc1$	50.8	48.3

Table I
COMPARISON OF KWS RESULTS WHEN USING DIFFERENT CNN LAYERS TO EXTRACT FEATURES. CONV1 IS THE DEEPEST, I.E. CLOSER TO THE INPUT LAYER. FC1 IS THE CLOSEST TO THE OUTPUT.

Unigram model vs bigram model vs both: We have ran trials using the best combination of layers in the previous experiment, and using layers coming from the unigram character model, the bigram model, or a combination of the two. In the latter case we have concatenated layer outputs of $conv2$ and $conv3$ into a single vector. This gives a very slight boost to results, in particular the three corresponding MAP figures on *GW20* were 50.3%, 50.9% and 51.2%. However, in the latter case feature-extraction is twice as expensive, as twice the number of feed-forward passes are required.

Input size: Before performing the feed-forward pass for each word image, we resize it to a fixed size, keeping fixed its aspect ratio. The larger this size, the smaller the CNN receptive field becomes in comparison. We have found that using results from forward-passing with a bigger resize height does not improve results; this is not unexpected, since the CNN is trained to detect characters that are around a fixed size (in our case, a window of 24 pixels per edge. However, concatenating layer output of two different sizes, one at the standard character size, and one at a finer level, can give a small improvement. We have used window edge size of 30 and 60 pixels. The corresponding MAP results on *GW20* are 51.2% and 47.0%. Combining the two gives 53.1%. The intuition behind this is that since the CNN is

[4] In [1] a more extended version is presented. We used 10 pages out of the total 153 pages, with ids: 416957, 416976, 417022, 417032, 416962, 416994, 416969, 416997, 416921, 416978.

no longer used as a character classifier but as a feature extractor, using multiple input rescalings becomes tantamout to looking for features at multiple resolutions. In our case, size 30 represents the coarse level and size 60 represents the finer level.

Using multiple window resizes requires multiple passes on the convnet, thus is as expensive as the number of different window sizes, as is using the unigram and the bigram model together. Also, the reported improvement is only slight. For this reason, for the remainder of this section we use a bigram CNN and a single resize window of 30 pixels to extract features.

Center prior: We have used a center prior to prioritize pixels close to center height of the image. While such a prior did not give better results for natural image-related tasks [6], we felt that it could be appropriate for word images since important information may lie closer to the word center of mass. A Gaussian kernel is used to penalize hypercolumns close to vertical edges. Instead of standard sum-pooling (eq. 1) we used a weighted variant (eq. 2). Preliminary results with a center prior were mediocre, possibly because the center of mass of the segmented word does not always coincide with the same relative height in the word image. For this reason we have first normalized the input so that the image main zone is centered and covers a fixed percentage of the total height [17]. A comparative trial on the *GW20* set shows a dramatic increase from 51.2% to 66.4% by normalizing the main zone alone. Using a center prior with $\lambda = 6$ improves results even further, to 72.7%. Tests on other sets have shown that normalizing and prioritizing hypercolumns on the main zone also improves results and is always very time-efficient, the extend of the improvement though is largely set-specific.

C. Discussion of results

We have tested the proposed ZAH method using the bigram CNN to extract features, layer $conv3$, center prior with $\lambda = 2$ and a resize window equal to 30 pixels. MAP and Precision at 5 results can be seen at table II. The compared methods are: ZAH (proposed), Adaptive Zoning [1], [13], Profiles/DTW [14], [16], PAS features [16], Attribute-based model [2], HOG/LBP-based method [9], Inkball model [18], Projections of Oriented Gradients (POG) [17], BoVW-based [8].

It is worth noting that even the "easiest" of all databases, i.e. the machine-printed, latin-script *French*, is already different in many aspects than the sets that our convnets were pretrained with (fig. 3), i.e. street-view text. However, the proposed method manages to work well even though no extra refining of the net has been performed on any of the test sets. We tested the descriptor's robustness to handling a script different than the one the related CNN was trained with. In our case, the language of training was English, and we tried testing the proposed descriptor in the polytonic Greek *Gazette* set. These result validates our expectation that the deep layers encode task characteristics that are abstract enough to be transferred succesfully to even a different script.

We show keyword spotting results also on the two datasets of the ICFHR 2014 KWS competition, *Bentham* and *Modern*, in table II. The proposed method compares fairly against other reported results. While the benchmark is learning-free, the winning attributes/SVM-based model of [2] does require offline learning. Both results of [2] were achieved using training sets that are very close to the test sets (GW20 and IAM). In comparison, our method is trained on a very generic set of word images, that is not even handwritten. In other words, our method requires no test set-specific finetuning, unlike [2]. In general, the proposed method attains results in the same ballpark as most other reported results of the ICFHR 2014 sets. In terms of P@5 in particular, in *Bentham* we are second only to the recent POG method [17], with a difference of only 0.7%.

V. Conclusion and Future Work

We have presented a novel descriptor for word images, and used it succesfully for segmentation-based keyword spotting. The proposed descriptor is computed as a function of a pretrained CNN and the input word image. No extra finetuning of the CNN on the test set is required. While convnets have been used in numerous task and various setups in document processing, the novelty of the current work lies in that we use the convnet's hidden layer activations, instead of using its output, to extract features suitable to describe document / word images. We adapted this paradigm from generic machine vision into word spotting by combining CNN feature extraction with sum-pooling aggregation and vertical zoning. Extensive trials on a variety of both machine-printed and handwritten documents have validated the method's usefulness.

In our model, activations of layers are combined to form pixel-level cues that are called hypercolumns. We used a simple sum-pooling and zoning scheme to encode hypercolumns to a single word-level descriptor. Compared to other state-of-the-art encoding schemes (such as Fisher vectors), our approach is extremely fast and requires no extra learning procedures, like training a GMM for FVs.

We used the pretrained CNNs largely as off-the-shelf models, in the sense that any other classifier could have been used in their place. Various other layouts for the pretrained CNN could be investigated. A straightforward extension could follow the trend of trying to use even deeper models [19].

In [19] it has been shown that training a useful CNN-based classifier is possible even when available training data are scarce. With respect to our model, this result may be interpreted as motivation for one to use a CNN trained directly and specifically on the style of the document of

	%								
	ZAH (proposed)	[1]	[14]	[16]	[2]	[9]	[18]	[17]	[8]
French	91.5	81.2	89.7	–	–	–	–	–	–
Gazette	92.4	68.2	89.8	–	–	–	–	–	–
GW20	71.1	22.5	22.1	37.5	∗	∗	–	–	–
Bentham	53.6	–	–	–	51.3	52.4	46.2	57.7	46.5
Bentham(P@5)	76.4	–	–	–	72.4	73.8	71.8	77.1	62.9
Modern	32.1	–	–	–	52.3	33.8	27.8	35.5	38.9
Modern(P@5)	56.0	–	–	–	70.6	58.8	56.9	61.3	61.9

Table II

Comparison of spotting results using the proposed ZAH descriptor against other methods in the literature. Reported results are MAP figures unless denoted otherwise. See text for a discussion of results. (* [2] and [9] report results on *GW20*, but their results are not comparable to ours; [2] uses a fold from the same set for training, and [9] use a different evaluation protocol).

interest, instead of using a generic pretrained CNN. We leave this line of research as future work.

References

[1] B. Gatos, A. L. Kesidis, and A. Papandreou, "Adaptive zoning features for character and word recognition," in *International Conference on Document Analysis and Recognition (ICDAR)*, 2011, pp. 1160–1164.

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

[3] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Deep features for text spotting," in *IEEE European Conference in Computer Vision (ECCV)*. Springer, 2014, pp. 512–528.

[4] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *IEEE European Conference in Computer Vision (ECCV)*, 2014, pp. 584–599.

[5] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 447–456.

[6] A. Babenko and V. Lempitsky, "Aggregating local deep features for image retrieval," in *IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[7] T. M. Rath and R. Manmatha, "Word image matching using dynamic time warping," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2003, pp. 521–527.

[8] D. Aldavert, M. Rusiñol, R. Toledo, and J. Lladós, "A study of Bag-of-Visual-Words representations for handwritten keyword spotting," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 18, no. 3, pp. 223–234, 2015.

[9] A. Kovalchuk, L. Wolf, and N. Dershowitz, "A simple and fast word spotting method," in *International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2014, pp. 3–8.

[10] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke, "A novel word spotting method based on recurrent neural networks," *IEEE Transactions in Pattern Analysis and Machine Intelligence*, vol. 34, no. 2, pp. 211–224, 2012.

[11] Y. Kalantidis, C. Mellina, and S. Osindero, "Cross-dimensional weighting for aggregated deep convolutional features," *arXiv preprint arXiv:1512.04065*, 2015.

[12] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *IEEE European Conference in Computer Vision (ECCV)*. Springer, 2014, pp. 818–833.

[13] B. Gatos, N. Stamatopoulos, G. Louloudis, G. Sfikas, G. Retsinas, V. Papavassiliou, F. Sunistira, and V. Katsouros, "GRPOLY-DB: An old greek polytonic document image database," in *International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 646–650.

[14] V. Lavrenko, T. M. Rath, and R. Manmatha, "Holistic word recognition for handwritten historical documents," in *Proceedings of the First International Workshop on Document Image Analysis for Libraries*. IEEE, 2004, pp. 278–287.

[15] I. Pratikakis, K. Zagoris, B. Gatos, G. Louloudis, and N. Stamatopoulos, "ICFHR 2014 competition on handwritten keyword spotting (h-kws 2014)," in *International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2014, pp. 814–819.

[16] A. P. Giotis, G. Sfikas, C. Nikou, and B. Gatos, "Shape-based word spotting in handwritten document images," in *International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 561–565.

[17] G. Retsinas, G. Louloudis, N. Stamatopoulos, and B. Gatos, "Keyword spotting in handwritten documents using projections of oriented gradients," in *International Workshop on Document Analysis Systems (DAS)*. IAPR, 2016, pp. 411–416.

[18] N. R. Howe, "Part-structured inkball models for one-shot handwritten word spotting," in *International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2013, pp. 582–586.

[19] S. Sudholt and G. A. Fink, "PHOCNet: A deep convolutional neural network for word spotting in handwritten documents," *arXiv preprint arXiv:1604.00187*, 2016.