

# An Alternative Deep Feature Approach to Line Level Keyword Spotting

George Retsinas<sup>1,2</sup>, Georgios Louloudis<sup>1</sup>, Nikolaos Stamatopoulos<sup>1</sup>, Giorgos Sfikas<sup>1,3</sup>, Basilis Gatos<sup>1</sup>

<sup>1</sup> Institute of Informatics and Telecommunications, NCSR “Demokritos”, Greece

<sup>2</sup> School of Electrical and Computer Engineering, National Technical University of Athens, Greece

<sup>3</sup> Department of Computer Science and Engineering, University of Ioannina, Greece

{georgeretsi,louloud,nstam,sfikas,bgat}@iit.demokritos.gr

## Abstract

*Keyword spotting (KWS) is defined as the problem of detecting all instances of a given word, provided by the user either as a query word image (Query-by-Example, QbE) or a query word string (Query-by-String, QbS) in a body of digitized documents. Keyword detection is typically preceded by a preprocessing step where the text is segmented into text lines (line-level KWS). Methods following this paradigm are monopolized by test-time computationally expensive handwritten text recognition (HTR)-based approaches; furthermore, they typically cannot handle image queries (QbE). In this work, we propose a time and storage-efficient, deep feature-based approach that enables both the image and textual search options. Three distinct components, all modeled as neural networks, are combined: normalization, feature extraction and representation of image and textual input into a common space. These components, even if designed on word level image representations, collaborate in order to achieve an efficient line level keyword spotting system. The experimental results indicate that the proposed system is on par with state-of-the-art KWS methods.*

## 1. Introduction and Related work

A significant, as well as ever-increasing amount of digitized handwritten documents exists today all over the world, creating an urgent need for efficient and automatic document collection indexing. Keyword spotting (KWS) refers to the task of automatically retrieving instances of a given keyword query within a document image. KWS has been proposed as an alternative to full text recognition in cases where recognition is deemed to be very difficult or expected to give poor results [5].

Depending on how the query word is specified by the user, either using an example word image or just a text

string, we can distinguish Query-by-Example (QbE) from Query-by-String (QbS) keyword spotting methods, respectively. A different taxonomy of KWS systems is related with the existence of segmentation of the document collection resulting to the distinction into segmentation-based and segmentation-free systems. Segmentation-free approaches aim to locate the query instances on the entire document without the involvement of any segmentation procedure whereas segmentation-based approaches assume that a segmentation step into either line (line-based) or word (word-based) level has preceded. The proposed method is considered as segmentation-based (more specifically line-based) since it is assumed that the segmented lines are provided. The main innovation introduced is that, for the first time, both QbE and QbS search options on line level are allowed under the same KWS system. The proposed system, which relies on the success of Pyramidal Histogram of Characters (PHOC) embeddings [1], is based on the combination of three components, namely the Character Width Normalization, the Feature Extraction and the Common Space Encoder. It should be noted that the main component of each subtask is a neural network. Experiments conducted on the well-known data set IAM [9] prove the effectiveness of the proposed system.

Concerning word-based KWS methods, the seminal work of Almazán et al. [1], which introduced the PHOC representation, has influenced many recently proposed methods. In a nutshell, the PHOC representation encodes whether a specific character appears in a particular spatial region of a text string. The main idea of this method relies on the ability to embed both word images as well as text strings into a common vectorial subspace. The use of a common representation for both word images and text strings makes feasible a search using either a word image (QbE) or a word string (QbS). The abovementioned PHOC representation is learned using Support Vector Machines (SVMs) while Kernel Common Subspace Regression (KCSR) is used in order to find projections that max-

imize the correlation of the image space and the text space in a common latent subspace.

A number of recent methods have been inspired by the work of Almazán et al., further extending or adapting the base model [11],[12],[7],[8],[16]. All these word-based approaches for keyword spotting make use of Convolutional Neural Networks (CNNs) instead of SVMs and KCSR for the attribute and common subspace learning. Sudholt and Fink [11, 12] were the first to use CNNs in order to learn the PHOC representation given the raw word image content. The method is able to outperform Almazán’s method under both QbE and QbS scenarios. In Krishnan et al. [7], a word image representation is first learned using a CNN and subsequently used to learn a common subspace with KCSR as in [1]. Finally, in Wilkinson et al. [16] a triplet CNN is employed, accepting pairs of positive word matches together with a negative word match. Moreover, a new word embedding is proposed, dubbed DCT of Words (DCTofW).

Methods categorized to the line-based approach typically subscribe to the QbS paradigm. Line-based keyword spotting was first introduced by Fischer et al. [3]. It is a supervised method which is based on character Hidden Markov Models (HMMs). The main advantage of the proposed method is its ability to work on line images making a word segmentation step unnecessary. Furthermore, the system is making use of character models which can be trained very easily since line level transcription and segmentation of document images can be produced very easily, while enabling zero-shot learning at the same time. The authors extended this work in [2], where they incorporated character language models (unigrams and bigrams) at the decoding step leading to a better performance compared to their previous method. A major drawback of the methods presented by Fischer et al. is the large computational cost of the keyword-specific HMM Viterbi decoding process needed to obtain the confidence scores of each word to be spotted. Toselli and Vidal [14] presented a novel approach for the computation of confidence scores, directly from character lattices produced during a single Viterbi decoding process using only the filler model. Experiments showed that same spotting results were obtained using the proposed method, while requiring between one and two orders of magnitude less query computing time.

The best-performing approach for line-based keyword spotting was reported by Frinken et al. [4]. In this work, the authors make use of a recurrent neural network (BLSTM network) using a modification of the CTC token passing algorithm which was previously used for Handwritten Text Recognition (HTR). Experiments conducted on the IAM dataset showed that the method outperformed the classical HMM approach. Finally, Toselli et al. [15] presented a line-based keyword spotting method on the basis of frame-level word posterior probabilities. KWS confidence scores are

obtained by word graphs produced by a full-fledged statistical HTR system, based on stochastic optical character models, as well as probabilistic lexicon and language models. Their system achieved very promising results requiring less time for indexing when compared to the system of Frinken et al., which is still the best in terms of performance.

The rest of this paper is organized as follows. We present an overview of the proposed system and outline its main contributions in Section 2. Sections 3, 4 and 5 describe in detail the main components of the proposed system, i.e. Character Width Normalization, Feature Extraction and Common Space Encoder. In Section 6, the proposed line matching procedure is presented, while experimental results on KWS trials are reported in Section 7 highlighting the effectiveness of the proposed system. Finally, conclusions are drawn in Section 8.

## 2. System Overview and Contribution

In this work, we present a novel keyword spotting method that is based on deep feature extraction and fast query to text line matching. The key idea of the proposed method is that the features extracted from the convolutional output of a PHOC estimation network, trained on word images, should be similar in the case when the input is a line image or even a document image. We should stress that the convolutional layers are not constrained by the size of the input image. As a result, the keyword search procedure can be addressed as a feature matching procedure, 1-d search when the input is a line image or 2-d when the input is a document image (segmentation-free approach). Based on the success of line segmentation methods [6], we assume line images as input since the matching procedure is significantly simplified (1-d search).

Our processing pipeline consists of three components, all modeled as neural networks: (a) the *Character Width Normalization*. With this component we can normalize inputs, rendering horizontal zones of deep features commensurate to one another, and enabling fast query matching, since instances of the same word have similar width at the normalized line images. (b) the *Feature extraction*. This component is responsible for producing discriminative features for the query as well as the text lines of the document collection. Deep features are defined as the activations of an intermediate layer, produced when feeding an image to the network. Such features have proven to be robust on training-test content disparity; for example they have been successfully applied to cross-language KWS tasks without the need for model refinement [10]. (c) the *Common Space Encoder*. This component enables the QbS option, i.e. using a string as a query instead of a word image.

After having processed the query and the text lines, a step which is performed offline, the query can be compared against the set of text lines on the fly using a fast matching

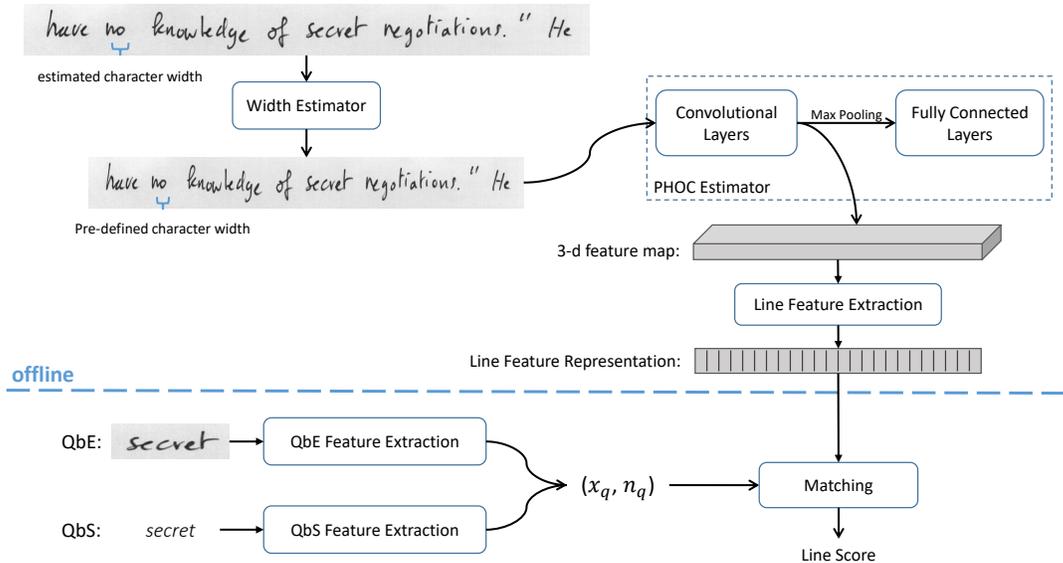


Figure 1. The proposed KWS system overview. Word image and string attribute inputs are encoded into a common space, suitable for performing an efficient line matching approach on the sequence of convolutional features representing the line image.  $x_q$  is the final feature vector and  $n_q$  is the number of line segments extracted from the query.

procedure. Matching of a specific query with a single text line can be accomplished very efficiently due to two factors: (a) matched features are extracted by max-pooling over a set number of horizontal zones (b) the size of the pooling filter is fixed given the word length of the query, due to the Character Width Normalization component. Matched tokens are then sorted and returned to the user. The proposed model processing pipeline together with the query to text line matching can be reviewed in Fig. 1.

One interesting aspect of this work is that, all the aforementioned components focus on the simplest word-level image representation and collaborate in order to efficiently address the complex problem of line-level KWS.

### 3. Character Width Normalization

The existence of different writing styles as well as different scales leads to considerable image width differences, even for the same word. This variance, closely related to scale, complicates the line matching procedure, since we cannot pre-define the space that a specific word, the query word, occupies. Therefore, in order to build an efficient KWS system, we propose an image normalization scheme, where different instances of the same word have the same width. Key step towards such a normalization is the estimation of the average character width which is performed by a convolution neural network since it can be easily correlated with visual features (e.g. curvature of strokes).

The normalization step should be able to estimate the character width irrespective of the size of the input image (word or line). One simple and efficient way to perform

the width estimation on both categories, word and line, is to extract a set of fixed-size patches from the images. The patches are randomly sampled from the image while their number is proportional to the size of the input image. The width estimation network  $f_w()$ , expects a fixed-sized image patch as input and provides an estimation of the character width for the specific patch. As the task at hand is a regression problem, the network  $f_w$  is trained using the mean squared error (MSE) loss function.

Even though the problem was clearly stated, the required labels for training the CNN, i.e. the character widths corresponding to image patches, are not known in advance. We follow a simple procedure for the assignment of the average character width (label) to each image patch. Given a word image and its transcription, the expected average character width (label) of every patch generated by this word image is the value  $w/n_c$ , where  $w$  is the image width and  $n_c$  is the number of characters of the word's transcription. Albeit the generated regression labels are not accurate, they suffice for this regression task since a few pixels difference will not affect the final result. In fact, the trained network provides an average loss of 3 pixels at test set, which is acceptable for the purpose of the normalization task.

Given an image  $I$ , a set of patches  $P_i, i = 1, \dots, n_p$  is generated and each patch has its own width estimation  $w_i = f_w(P_i)$ . It is possible that the system produces erroneous character width estimations, especially for the case of patches not containing representative information (e.g. patch which mostly contains blank space). This issue is addressed by the assignment of the median value

$w_c = \text{median}(\{w_i\})$  (more likely to correspond to the average character width) as the estimation for the whole image.

After estimating the average character width, we normalize the initial image to a fixed average character width  $w_{ref}$  (defined by the user). This normalization corresponds to a simple image resizing, for which the width is rescaled by a factor  $w_{ref}/w_c$ . The normalization process is depicted at Figure 1.

#### 4. Feature Extraction

A neural network, trained to learn a semantic prediction task can be used to easily extract so-called deep features [17]. We use deep features to describe both text lines and queries since such features are known to outperform traditional feature extraction methods on various vision tasks. These features are extracted as activations of a neural network with a feed-forward architecture, tuned to a semantic attribute prediction task.

In particular, the task is predicting word-level attributes that correspond to the existence or absence of a unigram or bigram at a specific section of the word. To this end, we adapt on the PHOCNet architecture inspired by the VGG architecture [11, 12]. The input is a word image of arbitrary size, processed by a convolutional backbone that leads to a fully-connected head topped by sigmoid outputs. Each sigmoid output predicts a word semantic attribute, e.g. whether the letter 'b' exists on the first half of the word, or whether the bigram 'st' exists on the second third of the word, and so on.

The proposed PHOC estimator network bears two differences w.r.t. to the PHOCNet architecture. The first novel point is the application of adaptive pooling on a specific region of interest (ROI), which corresponds to a part of the initial input image. We perform training using word images that contain information from neighboring words, i.e. not a tight crop of the word image as the word segmentation ground truth indicates (see Figure 2). This training strategy tunes the neural network in such a way that it can efficiently generalize to inputs corresponding to entire line images. In addition, it enables the creation of more robust filters, not affected by the existence of neighboring image elements. To this end, a bounding box with the region of interest is also defined for the pooling operation as it is shown in Figure 2. Hence, the training input consists of pairs of images and bounding boxes.

The second novel point is related with the pooling scheme used to create a fixed-size bridge with the fully-connected head. We employ a horizontal zoning-based scheme, dividing the text-line or word input into fixed-size horizontal segments. This scheme follows the rationale of zoning techniques that have been widely used for keyword spotting [5]. The use of pyramidal pooling is also avoided,

contrary to [11, 12]. We further simplify the above mentioned pooling scheme by completely ignoring the segmentation scheme over the convolutional map and instead, apply max-pooling over the entire map. This procedure will result to the creation of a feature vector whose size will be equal to the depth of the last convolutional layer. Furthermore, the application of a max-pooling operation over the entire convolutional map drastically reduces the size of the generated feature vector and therefore the number of parameters of the upcoming fully connected layer. The significance of this observation in the context of the proposed system, is that each word is represented using a collection of different filter responses instead of a sequence of feature vectors (zoning) or a set of sequences (pyramidal). The creation of a single feature vector simplifies the forthcoming matching procedure which can be accomplished using a nearest neighbor algorithm. The proposed pooling strategy hence results to the creation of a simplified and compact feature descriptor, suitable for fast matching.

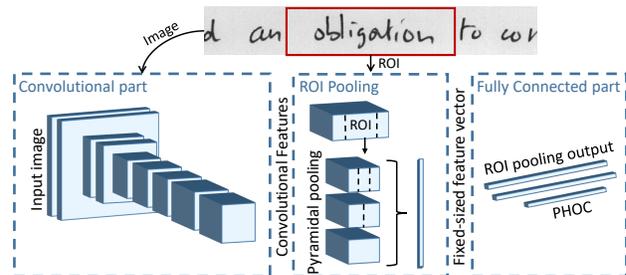


Figure 2. The PHOC estimation CNN consists of three distinct parts: 1) Convolutional 2) ROI Pooling and 3) Fully connected. Training is performed on an extended image containing a word instance along with the ROI bounding box of the word.

Given that  $f_c$  is the function consisting of the convolutional layers,  $f_p$  is the pooling operation over the ROI and  $f_{fc}$  is the fully connected part which results to the PHOC description, the PHOC estimation process can be formulated as follows:

$$x = f_p(f_c(I), b), \quad y = f_{fc}(x) \quad (1)$$

where  $I$  is the input image,  $b$  is the ROI bounding box,  $x \in \mathbb{R}^{n_d}$  is the extracted deep feature vector and  $y \in \mathbb{R}^{n_{phoc}}$  is the estimated PHOC. Note that  $n_d$  is an important hyper-parameter of the PHOC estimation network, since it defines the dimensionality of the extracted deep features.

#### 5. Common Space Encoder

The PHOC estimation network, after discarding the fully connected layers, provides discriminative features, ideal for a line-based approach. Nevertheless, the discarded layers create the connection between the convolutional features and the PHOC representation which is mandatory for the

QbS scenario. Therefore, an inverse function is required for the QbS scenario, i.e. a function that accepts a PHOC embedding (input) and produces a feature vector which is similar to the feature vector generated by the PHOC's estimator convolutional part (output).

One problem that may arise with this function is related with the fact that each input (PHOC representation) corresponds to a variety of outputs (word images of the same word written by different writers) and therefore the problem is ill-posed, since the function has to describe an one-to-many relation. This problem can be effectively solved by choosing to encode the PHOC embedding  $y$  as well as the convolutional feature vector  $x$  into a common space of fixed dimensionality  $n_e$ .

The encoding process is performed by two separate encoding functions, namely  $e_p$  (encodes the PHOC embeddings) and  $e_f$  (encodes the convolutional features), which are modeled by neural networks consisting of fully connected layers. Therefore, given a pair  $(x \in \mathbb{R}^{n_d}, y \in \mathbb{R}^{n_{phoc}})$  the resulting  $e_p(y) \in \mathbb{R}^{n_e}$  and  $e_f(x) \in \mathbb{R}^{n_e}$  must be identical in the encoding space.

A critical aspect concerning the training of the encoding functions is the creation of an appropriate loss function. The obvious goal is to minimize the distance between  $e_p(y)$  and  $e_f(x)$ , but such an approach may lead to trivial solutions (e.g. projection to the origin point). To address this problem, the training is performed using pairs of different words, i.e.  $(x_1, y_1)$  and  $(x_2, y_2)$ , and the goal is the minimization of the encoding distance within the same word while preserving an appropriate distance between different words.

Given a distance  $d(\cdot, \cdot)$ , the inner distance  $i$  and cross distance  $c$  are defined as:

$$\begin{aligned} i_1 &= d(e_f(x_1), e_p(y_1)), & i_2 &= d(e_f(x_2), e_p(y_2)) \\ c_1 &= d(e_f(x_1), e_f(x_2)), & c_2 &= d(e_f(x_1), e_p(y_2)) \\ c_3 &= d(e_p(y_1), e_f(x_2)), & c_4 &= d(e_p(y_1), e_p(y_2)) \\ i &= (i_1 + i_2)/2 & & (2) \end{aligned}$$

$$c = (c_1 + c_2 + c_3 + c_4)/4 \quad (3)$$

The obvious choice is to minimize the inner distance, while at the same time maximize the cross distance, i.e. the loss would be  $L = i - c$ . However, our goal is to preserve a cross distance at least as big as the reference distance  $m = d(y_1, y_2)$  in order to preserve the correlation of the initial PHOC representations. We call this reference distance as margin  $m$ . The margin term is crucial since the data distribution on the new space (with the help of the margin term) is expected to be similar to the data distribution on the PHOC space. In fact, the cross distance should be always greater than the inner distance by the aforementioned margin ( $c > i + m$ ) in order to ensure separable classes of

words. Therefore, the loss function is formulated as:

$$L = i + \max(0, i - c + m) \quad (4)$$

Implementation details: 1) The  $e_p$  consists of 4 fully connected layers with ReLU non-linearities, while  $e_f$  is a linear transformation. 2) The dimension of the encoding space is set to  $n_e = 128$ . 3) The cosine distance is selected as the distance function  $d$ . 4) The pairs  $(x, y)$  are selected such as their PHOC representations are not far apart, i.e.  $d(y_1, y_2)$  is less than a threshold  $t$ . The latter assists convergence, since unconstrained selection of pairs may generate dissimilar pairs which are easy to separate.

## 6. Line Image Features and Matching

Following the definitions of the subsystems and their respective networks, this section contains a detailed description on how the aforementioned systems are cooperating in order to perform line based KWS.

We can distinguish two main steps: 1) Extract compact features from a line image and store them. This step is performed off-line in a KWS application. 2) Given a query (either image or string), extract appropriate features and compute a matching score between the specific keyword and the line (see Fig. 1).

### 6.1. Line Features Extraction

The extraction of line features consists of three steps presented in Figure 1, as part of the overall system:

1. Resize the input line image ( $h_i \times w_i$ ) with respect to a pre-defined average character width  $w_{ref}$  using the Character Width Normalization component (see Section 3). The size of the resized image is  $h \times w$ .
2. Apply the convolutional part of the PHOC estimator network ( $f_c$ , see section 4) to the resized image produced in step 1. The output corresponds to a 3-d feature map, for which the height and width is proportional to the height and width of the resized line image ( $h/4 \times w/4$  - the two max pooling layers of kernel and stride 2 of  $f_c$  downsample the image by 4) while its depth corresponds to the number of filters of the final convolution layer ( $n_d = 512$ ).
3. Organize the aforementioned 3-d map ( $h/4 \times w/4 \times n_d$ ) of visual features into a sequence of feature vectors ( $n_l \times n_d$ ). The organization of the 3-d map into a sequence of feature vectors makes possible the efficient matching between a query word and a line image. The 3-d map is reduced by making use of local max-pooling performed on non-overlapping segments of size  $h/4 \times w_{step}$ , where  $w_{step} = w_{ref}/4n_{quant}$ . As a result, each line image is described by a short

sequence of feature vectors reducing the storage cost as well as the matching time. This step is depicted at Figure 3.

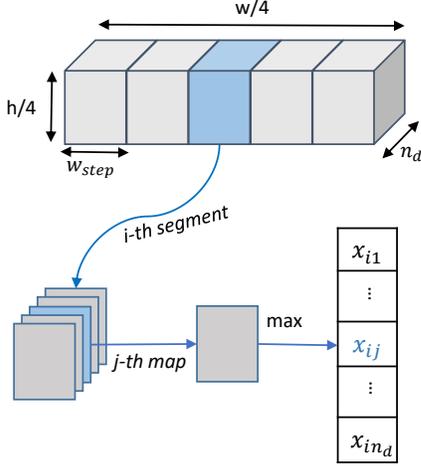


Figure 3. Extraction of line features from the 3-d convolutional feature map. Each line is represented by a sequence of  $n_l$  feature vectors of size  $n_d$ . Each feature vector occupies  $w_{ref}/n_{quant}$  width in the input line image.

## 6.2. QbE Feature Extraction

Given a query image  $I_q$ , we apply the following steps:

1. Estimate the average character width of the image and use it to resize the image accordingly (the resized image is denoted as  $I'_q$  and its size is  $h_q \times w_q$ ).
2. Calculate the number of line segments  $n_q$  by dividing the resized image's width by the step width  $w_{step}$  which is defined on the line features extraction section.
3. Apply the convolutional part of the PHOC estimator network ( $f_c$ , see section 4). This leads to the creation of a 3-d feature map whose size is  $h_q/4 \times w_q/4 \times n_d$ . Finally, extract a single feature vector by performing max-pooling for each channel (i.e. taking the maximum value over  $h_q/4 \times w_q/4$  values). It should be noted that during this step no ROI is defined.
4. Project the produced feature vector into the common encoding space using the encoding function  $e_f$  (see Section 5).

→ The final feature vector  $x_q$  is generated as:  $x_q = e_f(f_p(f_c(I'_q)))$

## 6.3. QbS Feature Extraction

Given a query string  $s_q$ , we apply the following steps:

1. Calculate the number of line segments  $n_q$ , by multiplying the number of query's characters with the character quantization parameter  $n_{quant}$  (each character should occupy  $n_{quant}$  line segments).
  2. Compute the PHOC embedding of the query string  $s_q$ .
  3. Encode the PHOC embedding into the common encoding space using the corresponding neural network  $e_p$  (see Section 5).
- The final feature vector  $x_q$  is generated as:  $x_q = e_p(PHOC(s_q))$ .

## 6.4. Line Matching

It is assumed that line features are represented as a sequence of vectors  $\{x_i\}$ ,  $x_i \in \mathbb{R}^{n_d}$ ,  $i = 1, \dots, n_l$ . The query (either image or string) is represented as a single feature vector  $x_q \in \mathbb{R}^{n_e}$ . A necessary parameter which should be known beforehand in order to perform the matching is the number of line segments  $n_q$  corresponding to the query width, i.e. how many line segments make up the query. The number  $n_q$  is calculated during the query feature extraction step (see Sections 6.2 and 6.3).

For each line, a set of scores is calculated using the cosine distance. In more detail, the scoring is performed using the cosine distance between the query feature vector and a feature vector created using a max-pooling procedure over consecutive  $n_q$  line features, i.e.:

$$\bar{x}_i[k] = \max_{j=0, \dots, n_q-1} x_{i+j}[k], \quad k = 1, \dots, n_d - n_q + 1 \quad (5)$$

$$d[i] = 1 - \cos(e_f(\bar{x}_i), x_q), \quad \bar{x}_i \in \mathbb{R}^{n_d}, x_q \in \mathbb{R}^{n_e} \quad (6)$$

The merge of consecutive  $n_q$  line features into a single feature vector, using a sliding window approach of max operations, is visualized at Figure 4. Since a set of feature vectors  $x_i$  can be extracted from a line image (the total number is equal to  $n_l - n_q + 1$ ), the final score is the minimum among all scores calculated using the set of feature vectors  $\bar{x}_i$ :

$$line\_score = \min_{i=1, \dots, n_l - n_q + 1} d[i] \quad (7)$$

For the matching step it is imperative for the two feature vectors to have the same dimensions. To this end, each feature vector  $\bar{x}_i$  is projected to the common encoding space using the network  $e_f$ . This task has been already performed for the query feature vector  $x_q$  during the feature extraction step. The simplicity of the matching procedure described above, is a considerable advantage of the proposed method, since it reduces the retrieval time.

## 7. Experimental Results

### 7.1. Experimental Setup

The performance of the proposed system was evaluated on the well-known IAM dataset<sup>1</sup>. It consists of 1539 hand-

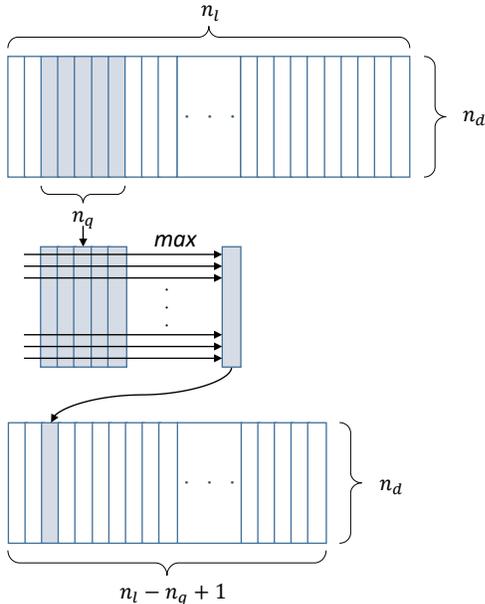


Figure 4. Line features transformation according to query size  $n_q$ . This step is essential in order to have comparable query and line features.

written document images of modern English written by 657 different writers and it has been partitioned into writer-independent training (6161 lines), validation (920 lines) and test sets (929 lines).

In order to be comparable with the results reported in the bibliography, we follow the two most widely used setups of the IAM dataset. **IAMDB1**: 882 queries selected as in [3] using all non-stop words that appear at least once in the training set as well as the test set. **IAMDB2**: All non-stop words among the 4000 most frequent words that also occur in the training set are selected as queries as in [4], resulting in 3421 queries in total.

In addition, for measuring the performance of the proposed system we consider two possible scenarios as in [3]. In the first scenario (**local**), a local threshold is used for each keyword separately. Concerning the second scenario (**global**), a global threshold is used that is independent of the keyword. For a vocabulary of common keywords, local thresholds can be optimized at training stage. On the other hand, for arbitrary out-of-vocabulary keywords, a global threshold has to be applied.

Another critical aspect for KWS techniques is the definition of the character set. Word-based approaches assume only lowercase and numeric characters [1]. On the contrary, line-based approaches [4] assume a wider variety of possible characters, including capital letters as well as some special characters (e.g. ‘/’, ‘-’ e.t.c). This variety also ex-

<sup>1</sup><http://www.fki.inf.unibe.ch/databases/iam-handwriting-database>

Approaches	QbE-PHOC	QbE-cfeat	QbS-PHOC
TPP	83.98	82.15	93.01
Zoning	83.66	82.24	92.47
Entire	83.27	81.73	91.11

Table 1. MAP (%) performance evaluation of different pooling strategies on the word-segmented IAM dataset.

ists in the aforementioned query lists and therefore, for our experiments, we adopt the same characters set (68 unique characters) and create 5-level PHOC embeddings with as many unigrams (1020 dimensional vectors).

The performance of the KWS methods is recorded in terms of the Mean Average Precision (MAP) since it is a retrieval problem. The retrieval list consists of the lines in the test set sorted according to their matching score with the query. A line that contains the requested query is considered as a hit.

## 7.2. Pooling strategies on PHOC estimation

One major difference of the proposed system with respect to PHOCNet is the usage of a PHOC estimation network with only one segment at the adaptive pooling layer, between the convolutional and the fully connected part. We experimented on the impact of the pooling operation to the system performance by evaluating three different strategies at the adaptive pooling layer: 1) Temporal Pyramidal Pooling - TPP (initial PHOCNet architecture [12]) 2) Zoning into 5 segments (without the pyramidal scheme) 3) Using max-pooling on the entire convolutional output (1 segment).

These strategies are evaluated on the segmented words as in [12] and the query list is extracted similarly to [1]. It should be stressed that the results are not directly comparable with the state-of-the-art word-level KWS techniques due to the fact that the characters set used in this work is significantly different (see Section 7.1). The results are presented at Table 1, where we distinguish the QbE scenario, both on PHOC level (QbE-PHOC) and on deep feature level (QbE-cfeat) and the QbS scenario on PHOC level (QbS-PHOC). As the experimental results indicate, the pooling strategy does not play significant role since the performance of the system is similar regardless the pooling scheme or the feature level. As a result, using max-pooling on the entire convolutional output simplifies the subsequent matching procedure while barely affecting the performance of the overall system.

## 7.3. Encoder Performance

A major contribution of this work is that it enables both QbS and QbE scenarios on deep features level, using an encoding neural network which projects convolutional features as well as PHOC embeddings into a common encoding

methods	local	global
QbS with ROI	88.73	83.15
QbS without ROI	87.13	79.14
QbE with ROI	84.25	73.16
QbE without ROI	83.01	71.58

Table 2. Impact of the ROI variation on the KWS system performance. MAP (%) results are reported at the line-level IAMDB1 setup ( $n_{quant} = 3$ ).

space. Therefore, we can evaluate both QbE and QbS on the new encoding space. For this experiment we used the pooling strategy over the entire convolutional map. The word level KWS with encoder component results to: **81.36%** and **89.43%** MAP for QbE and QbS respectively. These results are similar to the KWS scenario without the encoder even though the feature dimension is compressed (from 1020 of PHOC and 512 of convolutional features to 128 of the encoded features), while feature-based QbS is enabled.

#### 7.4. Line level KWS

Having confirmed the effectiveness of the systems components we proceed to the evaluation of the whole proposed KWS system using the IAMDB1 and IAMDB2 setups. It is worth to note that, even if QbE scenario is not considered on line-level by existing methods, we evaluated it using the query list of IAMDB1 setup since the corresponding query images can be detected at the training set. If there are more than one instance for a query string the query image is selected randomly (the reported QbE results are the average MAP of 10 runs). However, the IAMDB2 setup cannot be used for this case since query strings do not necessarily exist on the training set.

First, we evaluate the impact of the ROI variation of feature extraction network as shown in Table 2 using the IAMDB1 setup. The results indicate that the usage of a training scheme with expanded word images, as done in the ROI variation, assist the network to generate features that are more similar to the extracted line features.

The experimental results of the proposed method along with several state-of-the-art methods, for both IAMDB1 and IAMDB2 setups, are reported at Table 3. The proposed system, concerning the QbS scenario, significantly outperforms the majority of the line-based KWS methods reported in the literature and it achieves comparable performance compared to the best system [4]. Moreover, the experimental results indicate the validity and the effectiveness of the proposed method, which enables both QbS and QbE scenarios, since the QbE scenario achieves comparable results with the QbS scenario (especially for the local evaluation scenario). The gap in performance between QbS and QbE cases can be attributed to the fact that the QbE ap-

methods	IAMDB1		IAMDB2
	local	global	global
Fisher et al. [3]	68.92	47.75	-
Fisher et al. [2]	-	55.05	36.00
Toselli et al. [13]	-	-	61.03
Toselli et al. [15]	-	-	72.00
Frinken et al. [4]	-	-	76.00
proposed QbS	88.73	83.15	75.31
proposed QbE	84.25	73.16	-

Table 3. MAP (%) evaluation on the line-level IAMDB1 and IAMDB2 setups ( $n_{quant} = 3$ ).

proach may generates a feature vector that lies on the margin of the word cluster (defined by the feature vectors of all the instances of the same word), compared to QbS features that are unique per word and by default are in the center of the word cluster.

Another important observation is that even though we cannot straightforwardly evaluate the Character Width Normalization component, since we do not have accurate labels, we can conclude that it performs sufficiently well since the full pipeline provides noteworthy results.

Finally, we should stress that the proposed method can be used for real-time applications due to its storage and time efficiency. For clarity, we report some indicative storage and time requirements on the IAM dataset: the line image features, which computed offline, require 260KB storage (without any quantization), while it takes around 0.28msec to compare a query to a line.

## 8. Conclusions

In this work we have proposed a keyword spotting system that is capable of performing QbE and QbS KWS under a unified framework. The proposed system is built on the synergy of three distinct neural networks, respectively trained to normalize, encode image content into features, and construct a common text and image space. Even though the training is performed on word level, the comparative experimental results indicate that the proposed system is on par with state-of-the-art HTR-based QbS KWS methods, while it also enables successfully the QbE alternative. As future work, we envisage the generalization of the proposed framework for the segmentation-free scenario, i.e. not depending on the existence of document image segmentation at any level, but instead applying the method to the entire (unsegmented) document image.

**Acknowledgments:** *This work has been supported by the EU project READ (Horizon-2020 programme, grant Ref. 674943).*

## References

- [1] J. Almazán, A. Gordo, A. Fornés, and E. Valveny. Word spotting and recognition with embedded attributes. *IEEE transactions on pattern analysis and machine intelligence*, 36(12):2552–2566, 2014. [1](#), [2](#), [7](#)
- [2] A. Fischer, V. Frinken, H. Bunke, and C. Suen. Improving hmm-based keyword spotting with character language models. In *12th International Conference on Document Analysis and Recognition (ICDAR)*, pages 506–510, 2013. [2](#), [8](#)
- [3] A. Fischer, A. Keller, V. Frinken, and H. Bunke. Lexicon-free handwritten word spotting using character hmms. *Pattern Recognition Letters*, 33(7):934–942, 2012. [2](#), [7](#), [8](#)
- [4] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke. A novel word spotting method based on recurrent neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 34(2):211–224, 2012. [2](#), [7](#), [8](#)
- [5] Angelos P. Giotis, Giorgos Sfikas, Basilis Gatos, and Christophoros Nikou. A survey of document image word spotting techniques. *Pattern Recognition*, 68:310 – 332, 2017. [1](#), [4](#)
- [6] T. Gruening, G. Leifert, T. Strauss, and R. Labahn. A robust and binarization-free approach for text line detection in historical documents. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 236–241, Nov 2017. [2](#)
- [7] P. Krishnan, K. Dutta, and C. V. Jawahar. Deep feature embedding for accurate recognition and retrieval of handwritten text. In *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 289–294. IEEE, 2016. [2](#)
- [8] Praveen Krishnan and C. V. Jawahar. Hwnet v2: An efficient word image representation for handwritten documents. *CoRR*, abs/1802.06194, 2018. [2](#)
- [9] U.-V. Marti and H. Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, Nov 2002. [1](#)
- [10] George Retsinas, Giorgos Sfikas, and Basilis Gatos. Transferable deep features for keyword spotting. In *Multidisciplinary Digital Publishing Institute Proceedings*, volume 2, page 89, 2018. [2](#)
- [11] S. Sudholt and G. A. Fink. Phocnet: A deep convolutional neural network for word spotting in handwritten documents. In *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 277–282. IEEE, 2016. [2](#), [4](#)
- [12] S. Sudholt and G. A. Fink. Evaluating word string embeddings and loss functions for cnn-based word spotting. In *14th International Conference on Document Analysis and Recognition (ICDAR)*, pages 493–498. IEEE, 2017. [2](#), [4](#), [7](#)
- [13] Alejandro Héctor Toselli, Joan Puigcerver, and Enrique Vidal. Two methods to improve confidence scores for lexicon-free word spotting in handwritten text. In *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 349–354. IEEE, 2016. [8](#)
- [14] A. H. Toselli and E. Vidal. Fast hmm-filler approach for key word spotting in handwritten documents. In *12th International Conference on Document Analysis and Recognition (ICDAR)*, pages 501–505. IEEE, 2013. [2](#)
- [15] A. H. Toselli, E. Vidal, V. Romero, and V. Frinken. Hmm word graph based keyword spotting in handwritten document images. *Information Sciences*, 370:497–518, 2016. [2](#), [8](#)
- [16] T. Wilkinson and A. Brun. Semantic and verbatim word spotting using deep neural networks. In *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 307–312. IEEE, 2016. [2](#)
- [17] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *arXiv preprint*, 2018. [4](#)