LSDE: Levenshtein Space Deep Embedding for Query-by-string Word Spotting

Lluís Gómez and Marçal Rusiñol and Dimosthenis Karatzas
Computer Vision Center, Dept. Ciències de la Computació
Edifici O, Univ. Autònoma de Barcelona
08193 Bellaterra (Barcelona), Spain.

***Abstract*—In this paper we present the LSDE string representation and its application to handwritten word spotting. LSDE is a novel embedding approach for representing strings that learns a space in which distances between projected points are correlated with the Levenshtein edit distance between the original strings. We show how such a representation produces a more semantically interpretable retrieval from the user's perspective than other state of the art ones such as PHOC and DCToW. We also conduct a preliminary handwritten word spotting experiment on the George Washington dataset.**

I. Introduction

Handwritten word spotting refers to the image retrieval task focused to obtain a ranked list of word images that are relevant to a user's cast query. Since the seminal papers of Manmatha et al. [1], [2] that introduced the approach of handwritten keyword spotting as an alternative to recognition more than twenty years ago, many advances have been proposed. The performance achieved on public datasets has been steadily increasing with the proposal of better feature representations and retrieval strategies. In addition to the overall retrieval accuracy, many other advances have been made as well. In particular, two main aspects received a lot of attention lately. On one hand, segmentation-free methods have been proposed [3], [4], [5] and, on the other hand, query-by-string techniques have emerged [6], [7], [8], [9], [10], [11], [12].

A standard procedure for query-by-string word spotting is to define vectorial representations for both the word images and the text string queries. A subsequent embedding step is then used in order to map those two different representations into a common space in which distances can be computed between words in either modality. Several approaches have been proposed in order to compute these embeddings, such as LSA [6], CCA [8] or the use of CNNs [7].

Although in the literature there is a plethora of different visual feature proposals in order to represent word images, researchers scarcely focus on how the text strings are actually described. In [6], one of the first works to introduce such an embedding framework, Aldavert et al. proposed to encode strings by a bag of n-grams. Later, Almazán et al. [8] generalized this basic description and proposed the Pyramidal Histogram of Characters (PHOC) descriptor, that has been widely used since. In [11], Wilkinson et al. proposed the DCToW representation based on a discrete cosine transform over a character appearance vector. All these string representations were hand-crafted and might not be as powerful or

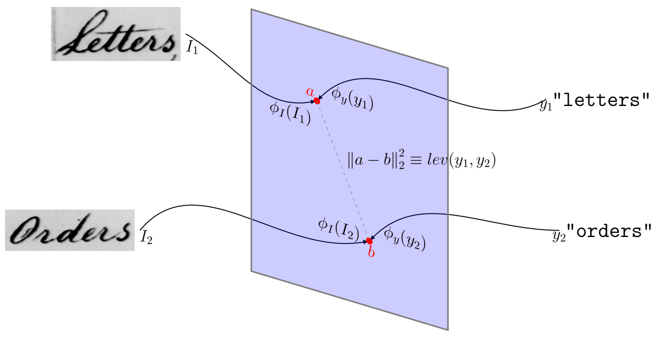


Fig. 1. LSDE learns an embedding space in which Euclidean distances correlate with the edit distance ($lev(\cdot,\cdot)$) between strings.

discriminative as desired. In the case of visual descriptions, we seek a representation that forces visually similar words (e.g. different instances of the same word) to nearby points in the description space. In the same sense, we should extend this notion of preserving distances to the string domain and seek representations that preserve the distance between strings. However, this is not guaranteed by existing representations, as small changes in a string might produce strong changes in both PHOC and DCToW representations.

The main motivation of our work is to enhance word spotting methods by introducing a better common embedding space, where string distances are preserved. Our research hypothesis is that an embedding space in which Euclidean distances correlate with the Levenshtein edit distance between strings, should have more descriptive power and be more generic, leading to more semantically interpretable, from the user's perspective, retrieval results when spotting words within a collection.

In this paper we present a novel embedding approach that *learns* a string embedding space in which distances between points are correlated with the Levenshtein edit distance [13] between the two original strings. Then we show how word images can be projected in the same space enabling thus multi-modal retrieval. Finally, we introduce a fine-tuning mechanism that adjusts the embedding space in a joint learning process. We compare our proposed LSDE (Levenshtein Space Deep Embedding) representation with the PHOC and DCToW representations and demonstrate that the LSDE representation is indeed much better correlated to the edit distance than the

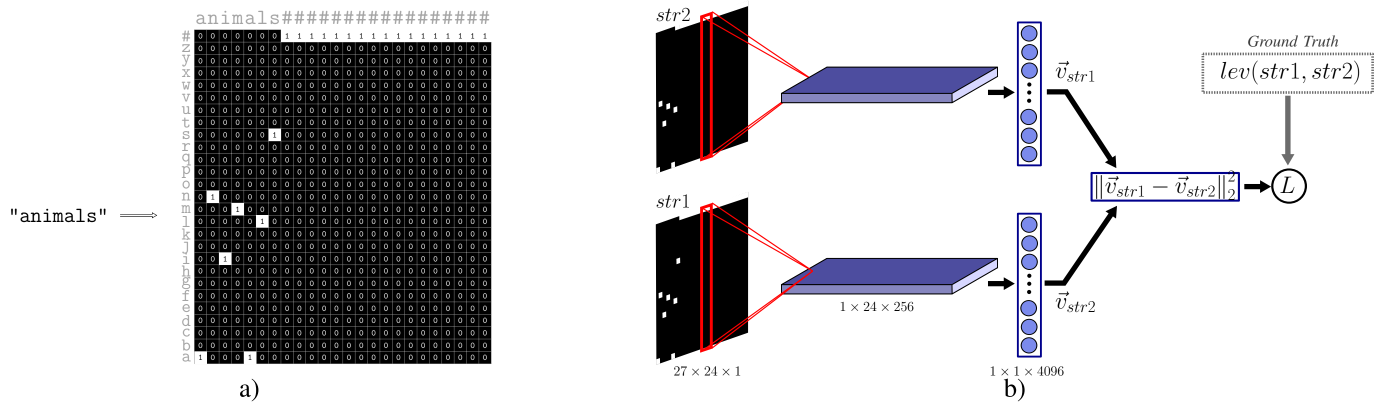


Fig. 2. a) Matrix representation of strings: a text string is encoded as a 27×24 matrix where columns correspond the one-hot vector representations of characters. b) A shallow CNN, composed by one convolution and one fully connected layer, transforms the input matrix representation into a 4096-d real valued vector. The network is trained with a siamese setup to output vectors whose Euclidean distance is equal to the edit distance of the input strings' pair.

rest. Then we conduct a preliminary handwritten word spotting experiment to test its performance.

The rest of this paper is organized as follows. Section II presents the proposed methodology for learning the Levenshtein Distance Space for string embedding, and the subsequent image embedding in the same space for the handwritten word spotting task. Section III provides the implementation details of the proposed networks and training procedures. Section IV presents the experimental results that were obtained while conclusions are drawn in Section V.

II. Architecture

Our goal is to learn a word representation in which both text strings and word images are embedded in a common Levenshtein space. Such a space has the following interesting property: the Euclidean distance between two points in this space is equivalent to the Levenshtein distance of the words they represent.

For this, we start by learning a string embedding model in which text strings are transformed into vectors with the aforementioned property. Then, we use the string embedding model as a teacher to train a convolutional neural network so that, given a word image as input, it predicts at its output the vector representation of its transcription as learned by the string embedding model. Finally, we construct a deep image-string embedding model by jointly fine-tuning the pre-trained image and string models. These three embedding models are detailed next.

A. Text String Embedding

In order to feed text strings into a convolutional neural network we represent them in a $N \times M$ matrix form, where N is the number of possible characters (e.g. $N = 26$ for the case of the lowercase Latin characters set) and M is the maximum desired word length (24 in our case). This way each encoded character corresponds to a one-hot vector in the corresponding matrix column for its particular position on the string. Strings are made fixed length M by adding trailing empty character symbols at the end. Figure 2 illustrates an example of this matrix representation for the text string "animals".

Upon this matrix representation of text strings our CNN model applies a convolutional layer with 256 kernels of size 27×3 and a fully connected layer with 4096 output neurons. Figure 2 depicts the architecture of our string embedding model. Despite its simplicity this architecture is the one that has provided a better trade-off between distance approximation and time performance among the many different architectures we have tried.

The network is trained with a siamese setup, presented with arbitrary pairs of text strings, using the following loss function:

$$L = \left(\sum(\vec{v}_{str1}, \vec{v}_{str2})^2 - lev(str1, str2)\right)^2 \qquad (1)$$

where $str1$ and $str2$ are two strings, $\vec{v}_{str1}$ and $\vec{v}_{str2}$ are the CNN output vectors for their respective matrix representations, and $lev(\cdot, \cdot)$ denotes the Levenshtein edit distance. As it can be seen, this simple loss function drives the CNN to learn the optimal transformation of strings into a vectorial form such that the squared Euclidean distance between vectors is equivalent to the Levenshtein edit distance of their originating strings.

It should be noted here that when training the string embedding model, we benefit of two important details: (1) the proposed CNN model is shallow and thus its inference time is rather small compared with deeper models, and (2) the available training data is unlimited. We can easily aim at a training procedure that randomly samples pairs of strings from a corpus with many thousands words and expect the network to see several millions of examples. We give concrete implementation details on how we have trained the string embedding model in the next section. We also provide empirical evidence of the effectiveness of the proposed model to approximate the true Levenshtein distance of arbitrary word pairs in section IV-A.

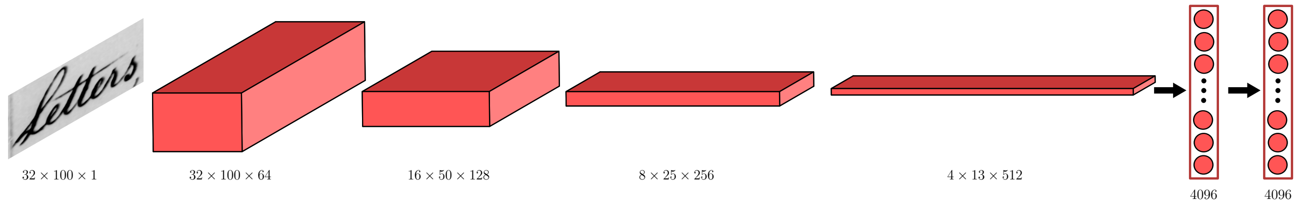


Fig. 3. Diagram of the CNN used for the image embedding model.

B. Image Embedding

Once the string embedding model is trained we can use it to teach an image embedding model so that, given a word image as input, it regresses at its output the LSDE representation of the corresponding string provided by the string embedding model.

For this second CNN model we adopt the architecture of the holistic word recognition network from Jaderberg *et al.* [14] as shown in Figure 3. To use it in our framework, we replace the softmax output layer by a fully connected layer with 4096 output neurons and Sigmoid activations, thus having the same dimensionality at the output (4096-d) as the LSDE representation. In this model, input images are resized to a fixed size of 32×100.

At training time either the Cross Entropy or Euclidean loss functions can be used to learn the optimal weights for embedding images into our learned Levenshtein space. In our case what proved to be more effective was to first train the network with a Cross Entropy loss and afterwards fine-tune it with an Euclidean loss. We give the implementation details in the next section.

C. Deep Image-String Levenshtein Space Embedding

Figure 4 illustrates the training architecture of the joint image-string embedding. We feed the joint model with pairs of word images (I_1, I_2) and their corresponding transcriptions $(str1, str2)$, that are transformed respectively by the image and string embedding models, thus providing four output vectors: two for the images $(\vec{w}_{I_1}, \vec{w}_{I_2})$ and two for the transcription strings $(\vec{v}_{str1}, \vec{v}_{str2})$. Under this design we define three different loss functions, (L_{ii}, L_{ss}, L_{is}), that are applied respectively to the vector representations of pairs of images (L_{ii}), pairs of strings (L_{ss}), and image-string combinations (L_{is}):

$$L_{ii} = (\sum(\vec{w}_{I_1}, \vec{w}_{I_2})^2 - lev(str1, str2))^2 \qquad (2)$$

$$L_{ss} = (\sum(\vec{v}_{str1}, \vec{v}_{str2})^2 - lev(str1, str2))^2 \qquad (3)$$

$$L_{is} = (\sum(\vec{w}_{I_1}, \vec{v}_{str2})^2 - lev(str1, str2))^2 \qquad (4)$$

The joint image-string embedding is initialized from the two trained models described in the previous sections. And the combined loss $L = L_{ii} + L_{ss} + L_{is}$ improves both models by fostering Euclidean-Levenshtein equivalence among all possible pairs of words representations.

At test time, the joint model can be used in a straightforward manner for query-by-string word spotting. When a new query

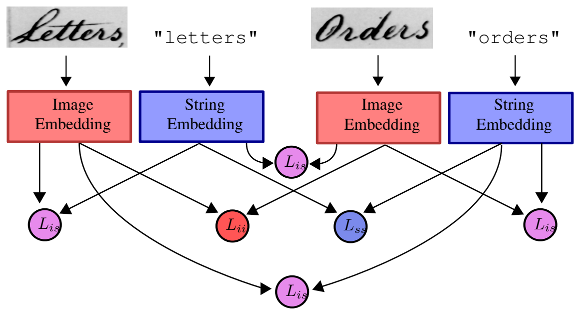


Fig. 4. The joint image-string embedding fosters the Euclidean-Levenshtein equivalence among all possible pairs of words representations.

string arrives, we first compute its vector representation using the string embedding model and then rank the set of word images according to the Euclidean distance in the embedding space of their vector representations with the query one. In the experimental section we show how the ranking produced in this way does not only perform well at retrieving relevant words in top rankings, but also obeys the Levenshtein space principle by ranking first the words with a smaller edit distance.

III. Implementation Details

We have implemented the embedding models using the Tensorflow [15] and Caffe [16] frameworks.

The text string embedding model is composed by a single convolutional layer with 256 kernels of size 27×3 and a fully connected layer with 4096 output neurons, with Rectified Linear Units (ReLU) activations in both layers. We have trained this model with a Stochastic Gradient Descent (SGD) optimizer for 5 million iterations, using a batch size of 32, and an initial learning rate of 0.00001 that is halved every million iterations. The training pairs of text strings are randomly sampled from the "Words" corpus of the NLPT Python library [17], which contains $236,736$ English words. On every batch we force half of the training pairs to be variations of the same word: the first word is randomly sampled and its couple is set to be the same word with a random number of edit operations. E.g. if the first word is "letters", the second word may become "xetters", "litters", "cutters", etc.

The image embedding CNN model is adapted from the holistic word recognition model proposed by Jaderberg *et al.* [14]. It has four convolutional layers and two fully connected layers, with ReLUs after each layer except for the last one in which we plug a Sigmoid activation.

The convolutional layers have 64, 128, 256, and 512 square filters respectively with kernel size of 3×3, stride $= 1$ and padding $= 2$. Max-pooling layers with kernel size 2×2 and stride $= 2$ follow the first three convolutional layers. The two fully connected layers have 4096 units.

We train the image embedding network using the RMSProp[1] optimizer in two stages. First we initialize the network with a standard multinomial logistic regression loss with dropout [18], following that we fine-tune the weights with an Euclidean (sum-of-squares) loss. In both stages we perform $250,000$ iterations with a batch size of 16 and an initial learning rate of 0.0001 that is decreased using the following decay policy: $lr = lr_0 * (1 + \gamma \cdot iter)^{(-\beta)}$, where lr_0 is the initial learning rate, $iter$ is the current iteration, and γ, β are two parameters set to $\gamma = 0.0001$ and $\beta = 0.75$. At training time we adopt the data augmentation strategy described in Sudholt *et al.* [7].

Finally, the joint image-string embedding model is initialized from the two pre-trained models described so far. The weights of both models are fine-tuned jointly using the multiple loss strategy detailed in section II. We train the joint model with SGD for one million iterations, using a batch size of 16, and an initial learning rate of 0.00001 that is halved every $250,000$ iterations. The training pairs of image-strings are randomly sampled from the training data but we force half of the samples on every batch to be words with a maximum edit distance of 3, thus focussing the learning on the most difficult examples.

IV. Experiments

We present two different sets of experiments, the first one dealing just with the LSDE text string representation and the other ones dealing with the query-by-string spotting application.

A. LSDE string representations

In order to test the performance of the LSDE representation we carried out an initial string retrieval experiment aimed at demonstrating that our learned text string embedding is more meaningful than other handcrafted string representations such as PHOC and DCToW. We considered the well-known Brown, Gutenberg and Reuters corpora [17] composed of 46.275, 41.504 and 29.190 unique words respectively. For this experiment we randomly selected 1000 words from each corpus to be used as queries and then ranked the rest of the corpus in terms of similarity in the LSDE space. What we expect to demonstrate is that our proposed embedding ranks words that are "similar" to the query in terms of their Lvenshtein edit distance better than when using PHOC or DCToW descriptions.

In order to assess such performance, we can not use the classical *mean average precision* measure since it only considers the ranking order of the true positives by using a *binary* relevance assessment. Either a word is deemed to be relevant to the query or not. In this experiment we are trying to evaluate how good the rankings are produced in terms of string similarity, and thus a binary relevance assessment is not enough. We choose thus to use the normalized discounted cumulative gain (nDCG) measure [19], that defines a gradual relevance score, computed as follows:

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p},$$

where

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL|} \frac{2^{rel_i} - 1}{\log_2(i+1)},$$

and

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)},$$

being p a rank position (the full corpus length in our case), $|REL|$ being the retrieved elements that are graded according to a relevance score rel_i. In our setup, we gave the following rel scores of 15, 10, 5 and 3 to all words with edit distances 1, 2, 3 and 4 respectively, and a null relevance to words with edit distances greater than 4. We present the nDCG measures in Table I obtained with the LSDE, PHOC and DCToW methods as well as with a random sorting of the words, just to put into perspective that small differences in the nDCG score are really meaningful. We observe that the proposed LSDE method outperforms both PHOC and DCToW in all the three corpora.

TABLE I
nDCG measures for the string retrieval experiment

Method	Brown	Gutenberg	Reuters
LSDE	99.022	98.895	98.791
PHOC	98.211	98.193	96.583
DCToW	97.584	97.562	95.868
Random Sorting	94.424	93.942	92.194

TABLE II
MSE between LSDE space distances and true Levenshtein edit distances

Method	Brown	Gutenberg	Reuters
LSDE	0.7121	0.6301	0.6786

In order to further prove that the LSDE string representations are more meaningful than PHOC and DCToW, we computed all the pair-wise distances of words within the Brown corpus and present in Figure 5 how well they correlate with the true Levenshtein edit distance. We appreciate that both PHOC and DCToW *do not* correlate with the edit distance, even yielding very high scores when comparing strings that only differ in a few characters. However, the Euclidean distance in the LSDE space provides a good approximation of the Levenshtein distance for a wide word spectrum, just showing a small decay for words with very high edit distances. We present in Table II

[1]http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

the Mean Square Errors (MSE) between LSDE representations and true unnormalized Levenshtein edit distances showing how the LSDE distance is a good approximation of the Levenshtein distance.

B. Word Spotting

For the query-by-string word spotting task we use the George Washington (GW) dataset [20]. The GW dataset was created from the George Washington letters at the Library of Congress, a collection dated from the 18th century. It consists in a set of 20 pages, and $4,860$ word instances of $1,124$ unique words. While there is no official partition for the GW dataset, we follow the approach of [21], [6], [8] that splits the dataset in two sets at word level containing 75% of the words for training purposes and the remaining 25% for test. This data partition is repeated four times for cross validation purposes, here we make use of the same cross validation fold as in [8], and provide average results.

In query-by-string word spotting, given a string query, the goal is to retrieve all word image instances in the test set that match the query. Following the standard evaluation protocol for the GW dataset we use all unique transcription strings in the test partition as queries. Then, we rank all word images in the test set by their similarity with the query string and report the mean average precision (mAP) for the entire set as the final performance measure.

Table III compares our method with the state of the art in query-by-string word spotting on the GW dataset. We appreciate the competitive performance of the proposed method, while using a notably smaller CNN model for the image embedding than the ones used in some other recent methods [7], [11].

But using mAP to assess the retrieval performance might be misleading, since mAP does not care at which edit distance the representation of a false positive is from the query. Even if we obtain a lower mAP than other methods, our method is able to provide more human understandable ranking by proposing earlier words that are not an exact match to the query, but are close in the string distance sense. The LSDE ability to retrieve "similar" words in terms of Levenshtein distance is first evaluated showing some qualitative results in Figure 6, in which we can see that the edit distances of the false positive words are usually quite small. In addition, Figure 7 plots the average edit distance between the query and the transcriptions of the word images retrieved in the first top-200 results, where we see that even considering the topmost 100 results, we are still no more than 4 character operations apart from the query string.

TABLE III
MEAN AVERAGE PRECISIONS FOR STATE-OF-THE ART QUERY-BY-STRING METHODS IN THE GW DATASET

Method	QBS mAP
Aldavert *et al.* [6]	56.54
Frinken *et al.* [21]	84.00
Almazán *et al.* [8]	91.29
Sudholt *et al.* [7]	92.64
Krishnan *et al.* [10]	92.84
Wilkinson *et al.* [11]	93.69
LSDE	91.31

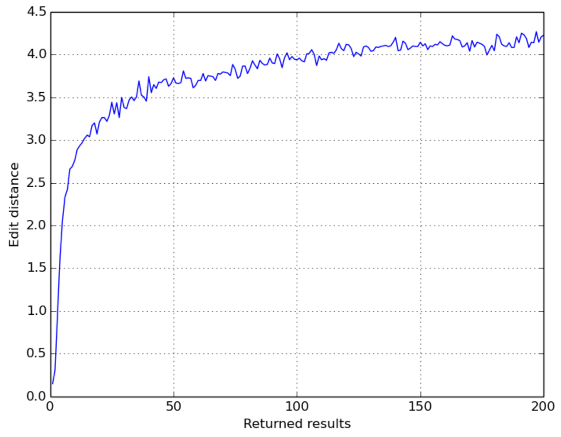


Fig. 7. Average edit distance between query and transcriptions of the query-by-string experiment

V. CONCLUSIONS

In this paper we have presented the LSDE string representation and its application to handwritten word spotting. LSDE is a novel embedding approach for representing strings that learns a space in which distances between projected points are correlated with the Levenshtein edit distance between the original strings. We have experimentally demonstrated that the proposed representation has better semantic interpretation and superior behaviour than other state of the art ones such as PHOC and DCToW. Moreover, we have also shown a preliminary handwritten word spotting method using LSDE that yields competitive results to the state of the art on the George Washington dataset while using a smaller CNN model for image embedding. As further improvements, we plan to work on proposing a segmentation-free approach with the use of sliding windows and to avoid the image resize step by using a Spatial Pyramid Pooling layer in the image embedding net.

ACKNOWLEDGMENTS

This work was supported by the Spanish project TIN2014-52072-P and by the CERCA Programme / Generalitat de Catalunya. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

REFERENCES

[1] R. Manmatha, C. Han, E. Riseman, and W. Croft, "Indexing handwriting using word matching," in *Proc. of the ACM Internal Conference on Digital Libraries*, 1996, pp. 151–159.
[2] R. Manmatha, C. Han, and E. Riseman, "Word spotting: a new approach to indexing handwriting," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 1996, pp. 631–637.
[3] M. Rusiñol, D. Aldavert, R. Toledo, and J. Lladós, "Efficient segmentation-free keyword spotting in historical document collections," *Pattern Recognition*, vol. 48, no. 2, pp. 545–555, 2015.

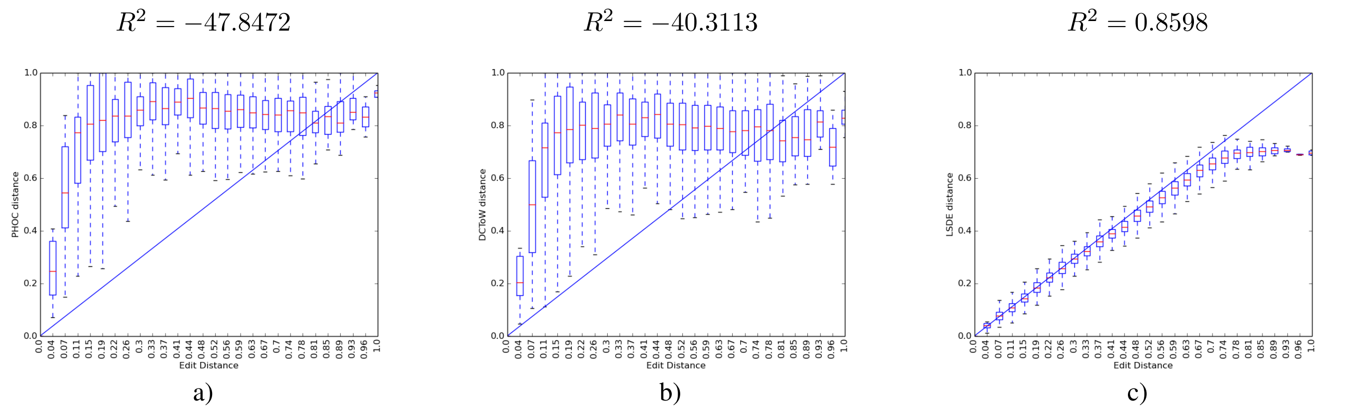


Fig. 5. Boxplot comparing true Levenshtein edit distances and distances computed for the string representations with associated coefficient of determination R^2 values for a) PHOC, b) DCToW and c) LSDE

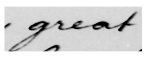
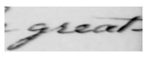
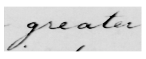
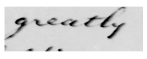
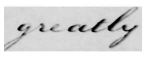
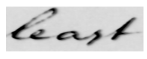
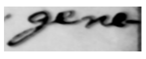
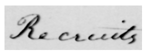
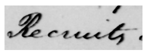
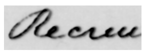
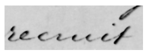
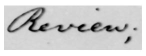
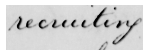
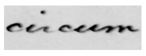
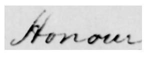
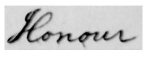
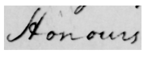
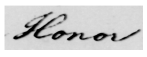
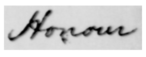
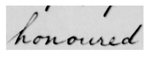
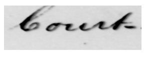
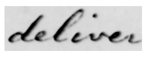
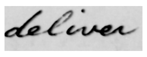
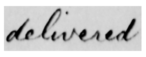
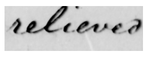
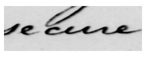
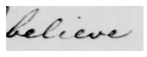
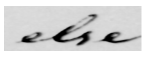
great	 great ED=0	 great ED=0	 greater ED=2	 greatly ED=2	 greatly ED=2	 least ED=3	 gene ED=3
recruits	 recruits ED=0	 recruits ED=0	 recru ED=3	 recruit ED=1	 review ED=5	 recruiting ED=3	 circum ED=7
honour	 honour ED=0	 honour ED=0	 honours ED=1	 honor ED=1	 honour ED=0	 honoured ED=2	 court ED=4
deliver	 deliver ED=0	 deliver ED=0	 delivered ED=2	 relieved ED=3	 secure ED=5	 believe ED=3	 else ED=4

Fig. 6. Qualitative results of the LSDE query-by-string experiment

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

[5] L. Rothacker, M. Rusiñol, and G. Fink, "Bag-of-features HMMs for segmentation-free word spotting in handwritten documents," in *Proc. of the 12th International Conference on Document Analysis and Recognition*, 2013, pp. 1305–1309.

[6] D. Aldavert, M. Rusiñol, R. Toledo, and J. Lladós, "Integrating visual and textual cues for query-by-string word spotting," in *Proc. of the 12th IAPR International Conference on Document Analysis and Recognition*, 2013, pp. 511–515.

[7] S. Sudholt and G. Fink, "PHOCNet: A deep convolutional neural network for word spotting in handwritten documents," in *Proc. of the 15th International Conference on Frontiers in Handwriting Recognition*, 2016, pp. 277–282.

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

[9] L. Rothacker and G. Fink, "Segmentation-free query-by-string word spotting with bag-of-features hmms," in *Proc. of the 13th International Conference on Document Analysis and Recognition*, 2015, pp. 661–665.

[10] P. Krishnan, K. Dutta, and C. Jawahar, "Deep feature embedding for accurate recognition and retrieval of handwritten text," in *Proc. of the 15th International Conference on Frontiers in Handwriting Recognition*, 2016, pp. 289–294.

[11] T. Wilkinson and A. Brun, "Semantic and verbatim word spotting using deep neural networks," in *Proc. of the 15th International Conference on Frontiers in Handwriting Recognition*, 2016, pp. 307–312.

[12] A. Poznanski and L. Wolf, "CNN-N-Gram for handwriting word recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2305–2314.

[13] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[14] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," *arXiv:1406.2227*, 2014.

[15] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. of the 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv:1408.5093*, 2014.

[17] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python. Analyzing Text with the natural language toolkit*, 3rd ed. O'Reilly Media, 2009.

[18] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv:1207.0580*, 2012.

[19] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. on Information Systems*, vol. 20, no. 4, pp. 422–446, 2002.

[20] T. M. Rath and R. Manmatha, "Word spotting for historical documents," *International Journal on Document Analysis and Recognition*, vol. 9, no. 2, pp. 139–152, 2007.

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