High Performance Text Recognition using a Hybrid Convolutional-LSTM Implementation

Thomas M. Breuel
NVIDIA Research
Santa Clara, CA, USA
tbreuel@nvidia.com

***Abstract*— Optical character recognition (OCR) has made great progress in recent years due to the introduction of recognition engines based on recurrent neural networks, in particular the LSTM architecture. This paper describes a new, open-source line recognizer combining deep convolutional networks and LSTMs, implemented in PyTorch and using CUDA kernels for speed. Experimental results are given comparing the performance of different combinations of geometric normalization, 1D LSTM, deep convolutional networks, and 2D LSTM networks. An important result is that while deep hybrid networks without geometric text line normalization outperform 1D LSTM networks with geometric normalization, deep hybrid networks with geometric text line normalization still outperform all other networks. The best networks achieve a throughput of more than 100 lines per second and test set error rates on UW3 of 0.25%.**

I. Introduction

The field of optical character recognition (OCR) has advanced greatly with the introduction of Long Short Term Memory (LSTM) recurrent neural networks; these networks are now routinely used in OCR engines for many different scripts; they have greatly simplified the development of OCR engines and made them much more widely applicable. Nevertheless, OCR performance can still be improved in a number of areas, including lowering error rates further, increasing throughput, and simplifying training.

OCR is generally carried out as a multi-step process, starting with page segmentation and text line detection, followed by text line recognition, and language modeling. Traditional OCR systems treat text line recognition as a problem of character segmentation, character recognition, and a search through a recognition lattice. When using recurrent neural networks like LSTMs for text line recognition, each text line is scanned in reading order (from left to right for Latin script), feature vectors are extracted, and the sequence of feature vectors is given as the input sequence to the recurrent neural network. The neural network outputs a sequence of posterior probabilities, one for each input feature vector. During training, these output feature vectors are aligned with the ground truth using the Connectionist Temporal Classification (CTC) algorithm. The CTC algorithm can be understood as a special case of the forward backward algorithm [8], applied to the output from the recurrent neural network and a transformed ground truth model in which a target string like ABC is transformed into a regular expression of the form $\epsilon^+A^+\epsilon^+B^+\epsilon^+C^+\epsilon^+$.

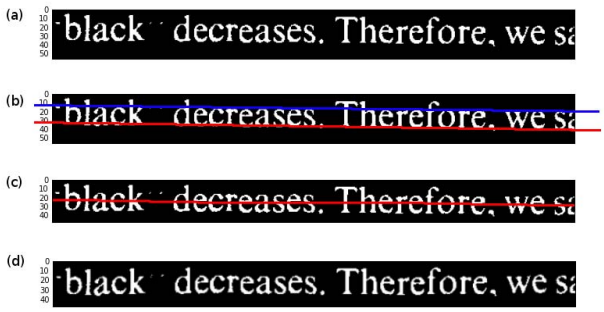


Fig. 1. An illustration of geometric text line normalization: (a) input text line, (b) baseline/x-height estimation, (c) centerline, (d) geometrically normalized line, obtained by straightening out the centerline.

This general approach to text line recognition was used for handwriting recognition by Liwicki *et al.* [9] and Graves *et al.* [7], using architectures based on one or more layers of LSTMs and pooling. Hybrids of deep convolutional networks and recurrent neural networks have also been used in other settings, such as scene text recognition[12, 6]. But handwriting recognition and scene text often differ from OCR in one or more important aspects, such as the amount of text being recognized, the use of word error rates, the use of language models, and throughput requirements.

In contrast to handwriting recognition, printed text recognition applications require character accurate transcriptions, including for novel words, with error rates of less than 1% without a language model. In addition, character sets tend to be larger and image resolutions tend to be smaller in printed text recognition than in handwriting recognition. Furthermore, large scale digitization efforts have high throughput requirements for text line recognition, which tends to be the most compute-intensive part of the OCR process. For printed texts in Latin scripts, character position relative to the baseline is also a crucial feature. To meet these requirements, LSTM-based OCR systems tend to perform geometric normalization operations on each text line image. When geometric normalization is reliable and predictable, even simple 1D LSTM-based recognizers can give excellent recognition performance.

2379-2140/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDAR.2017.12

With Normalization:	
Lstm1d(200)	0.40%
Lstm1d(512)	0.43%
Cr(64) Mp Concat Lstm1d(100)	0.36%
Cr(64) Mp Cr(128) Mp Concat Lstm1d(100)	0.25%
Cr(64) Mp Cr(128) Mp Cr(256) Mp Concat Lstm1d(512)	0.25%
C(16) Lstm2d(16) Mp Concat Lstm1d(100)	0.54%
Cr(16) Mp Cr(32) Mp Concat Lstm1d(100)	0.31%
Without Normalization:	
Cr(64) Mp MaxRed Lstm1d(100)	1.2%
Cr(64) Mp Cr(128) Mp MaxRed Lstm1d(100)	0.54%
Cr(64) Mp Cr(128) Mp Cr(256) Mp MaxRed Lstm1d(512)	0.43%

Fig. 2. Best results for each network architecture. Key: Cr(64): 3x3 convolutional layer with ReLU and output depth of 64; Mp: 2x2 max pooling with 1x2 stride; Lstm2d(64): 2D LSTM with 64 outputs; Lstm1d(100): 1D LSTM with 100 outputs; Concat: concatenation of feature vectors along the vertical; MaxRed: maximum reduction of feature vectors along the vertical.

Some of the most widely-used neural network-based OCR systems, OCRopus [3, 2, 15, 14] and Tesseract [13] both implement custom deep learning frameworks (Tesseract 4.0 alpha adds support for LSTM networks, [10]). That is because, until recently, deep learning frameworks used for object recognition and other applications, were not flexible or scalable enough to deal with the large images and variable image sizes that occur in document analysis and textline recognition. Neither of these custom frameworks has supported GPU acceleration either, meaning that neither of these systems could take advantage of the order of magnitude speedups that customarily result from the use of GPU-based deep learning implementations.

This paper describes a new text line recognizer based on the PyTorch [11] deep learning framework. PyTorch has excellent support for variable sized inputs and utilizes optimized cuDNN [5] kernels directly, making it a good match for OCR applications.

The rest of the paper will describe the PyTorch-based recognizer. Furthermore, given the compute power available with GPU-based implementations, this paper also explores the performance of combinations of convolutional and recurrent neural networks for text line recognition and tries to answer the question of whether deep convolutional architectures allow us to dispense with the geometric normalization step.

II. Methods

A. Network Architectures

All the networks described in this paper use a 1D LSTM followed by CTC training for the final layer.

For comparison with the original OCRopus text line recognizer, the input to the 1D LSTM consists of geometrically normalized text lines. The text line normalization process takes any input text line and turns it into a text line image of height 48 and variable width (an average of 854 pixels for the UW3 database; the details of this normalization process are described below. For the 1D LSTM-only text line recognizer, each 48 pixel input column is used directly as a 48-dimensional input vector to the LSTM.

For the deeper network architectures explored in this paper, we use one or more layers of 3×3 convolutions followed by ReLU non-linearities and 1×2 max pooling. Alternatively, we use a variant of multi-dimensional LSTMs that run bidirectional 1D LSTMs first over each row and then over each column of the input image, again, followed by max pooling. This, of course, represents only a small sample of the kinds of deep architectures we might explore; future, large-scale experiments would benefit from varying the footprint of the convolutions and the max pooling and the non-linearities.

The output of the convolutional layer gives rise to a $h \times w \times d$ feature map. A 1D LSTM requires a sequence of $w \times d'$ feature vectors. To transform the output of the convolutional layer into the input of the final 1D LSTM, we consider two primary transformations: concatenation and max-reduction. Concatenation takes the h feature vectors of dimension d and concatenates them into a feature vector of size $d' = h \cdot d$. Max reduction takes the maximum over each element of the d-dimensional feature vector along the h dimension. Concatenation preserves spatial information along the vertical dimension, while max reduction makes the resulting feature vector independent of vertical position.

In the Results section, we use a simple notation for representing the layers in the different models: $Cr(48)$ is a convolutional layer with ReLU nonlinearity and an output depth of 48; Mp to represent 1×2 max pooling, $Lstm2d(64)$ for a separable 2D LSTM, $PrMax$ for max projection along the vertical, $Concat$ for concatenation, and $Lstm1d$ for the final 1D LSTM (this notation can actually be used for specifying networks in the software). All models had a final linear layer to map the outputs of the LSTM into a class vector for each state. All outputs were treated as softmax outputs prior to the CTC.

B. Geometric Line Normalization

Scripts based on the Latin alphabet are some of the most important scripts for OCR engines to recognize. These scripts have the unusual property that many character pairs are nearly identical in shape but are distinguished from one

another primarily on their position and size relative to an imaginary *baseline* and *x-height*; examples of such character pairs are “p”/“P”, “o”/“O”, and even single quote vs. comma. The baseline and x-height of a line of text are typesetting concepts but are not explicitly indicated on text lines. On long text lines, it is usually easy to infer the baseline and the x-height of a line of text; for example, the character shape for lower-case “m” is unique and different from the corresponding upper case character. The lower left corner of that character will be on the baseline, and the height of its bounding box will correspond to the x-height. Baseline finding is further complicated by the fact that many text lines in scanned documents are not entirely axis aligned.

We can use both machine learning and optimal geometric matching for finding baselines and x-heights quite accurately in most cases and normalize the position and size of a text line prior to passing the text line into a 1D LSTM for recognition. However, when geometric text line normalization fails, it tends to lead to a large number of recognition errors on the entire line, meaning that errors in text line normalization tend to contribute substantially to overall OCR error rates. An alternative approach, and one that has turned out to be more robust is text line normalization based on directional image smoothing[4]. That is, we smooth the input image using anisotropic filters, then find ridges in the smoothed image. The resulting text lines look distorted, and introduce some context dependency into the feature vectors, but LSTM-based recognizers appear to be able to model this well. These two alternative normalization methods are illustrated in Figure 1.

It might also be possible to dispense with geometric normalization altogher, since LSTM-based handwriting recognizers generally do not used precise text line normalization as part of the recognition process; instead, they alternate convolutional or MDLSTM layers with max pooling, followd by a final projection of the feature vectors onto the axis corresponding to the time dimension of the recurrent networks. The combination of convolutions and max pooling has been found to be quite effective for achieving translation invariance in visual object recognition and handwriting recognition, so it is reasonable to ask whether we can dispense with any kind of geometric text line normalization and simply rely on combinations of convolutional and max pooling layers instead.

C. Training

All results in this paper are derived using text lines from the University of Washington Database III, split randomly into 95190 text lines representing 4.5 million characters for training, and 1253 text lines representing 60000 characters for testing. Although the library supports it, no data augmentation or synthetic data was used in the experiments reported in this paper in order to be able to train many models fast (for optimal performance in a deployed system, we would use both data augmentation and artificial data generation). Learning rates were chosen between 10^{-5} and 10^{-4}, with a momentum parameter of 0.9.

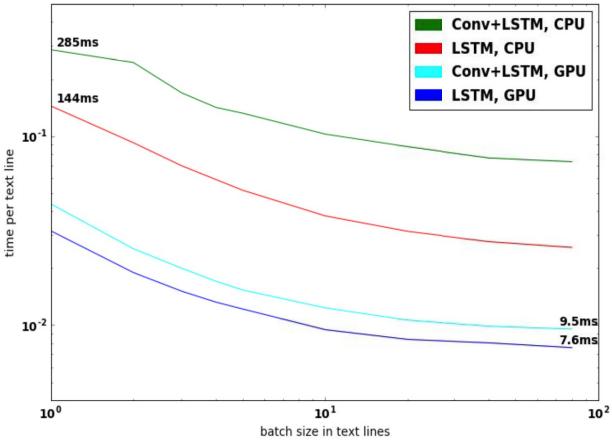


Fig. 3. Running times for the recognizer for different models and batch sizes. Conv+LSTM is a Cr(64) Mp Cr(128) Mp Lstm1d(100) model, LSTM is an Lstm1d(200) model. Text line inputs are an average of 48×854 pixels large after size normalization.

Count	True	Predicted
13	,	.
12	SPACE	ϵ
10	ϵ	SPACE
5	ϵ	m
5	ϵ	i
4	ϵ	w
4	ϵ	-
3	ϵ	l
2	ϵ	k
2	ϵ	S
2	l	l
2	ϵ	u
2	I	l
2	t	ϵ
2	ϵ	/
2	ϵ	b
2	ϵ	a
2	’	ϵ

Count	True	Predicted
15	.	ϵ
14	SPACE	ϵ
13	ϵ	SPACE
10	’	ϵ
8	,	.
8	’	,
7	l	ϵ
6	0	o
6	ϵ	d
5	ϵ	m
5	S	s
5	ϵ	l
5	c	e
3	ϵ	a
3	l	i
3	ϵ	.
3	I	l
3	i	ϵ

Fig. 4. The most common errors made by the recognizers are confusions between period and comma, insertions and deletions of spaces, and insertions of spurious characters (ϵ-to-something errors). The left results are for Cr(64) Mp Cr(128) Mp Concat Lstm1d(100); the right model is Cr(64) Mp Cr(128) Mp Cr(256) Mp MaxRed Lstm1d(512), and both evaluations are on the same randomly selected test set from UW3.

Training sequences are batched into batches of size 5 consisting of sequences of similar lengths. This achieves significantly better efficiency than training one sample at a time, but still keeps any batch size effects on training small. Larger batch sizes also end up requiring more GPU memory. For prediction, batch size has no effect on error rates, and generally batch sizes of 20-50 were used.

Networks were trained for a total of up to 5 million presentations of text lines. Text input lines were shuffled so that the order of presentation was randomized for each epoch.

For experiments without geometric text line normalization, text lines were cropped to the bounding box of non-zero pixels. If the resulting image was taller than 64 pixels, it was scaled down to 64 pixels.

D. Grayscale Degradations

For the experiments on degraded grayscale images, inputs were degraded by Gaussian blurring with randomly chosen sigmas in the range $[0.5, 3]$, addition of noise from between 5% and 20%, and addition of low-pass filtered (sigma in the range $[3, 100]$) background and foreground structure of maximum values between 40% and 60% of the total image range. Examples of text line image degradation with this model are shown in Figure 5.

E. Implementation

PyTorch provides excellent high level abstractions for both multicore and GPU implementations, making the implementation of different network architectures fairly easy. The CTC loss is implemented via a multicore CTC implementation in C++.

All timing experiments were carried out on an i7-6600 desktop running Ubuntu 16.04 and CUDA 8.0 with a NVIDIA GTX 1080 graphics card; networks were also trained on AMD desktop machines with NVIDIA GTX 1060 cards.

III. Results

A. Models with Normalization

Experimental results are shown in Figure 2. The first set of results gives the performance of models with normalization. The Lstm1d results use the same architecture as [2], but normalize text lines to a larger text height (48 pixels vs 32 pixels in the prior paper) use a larger number of hidden units (200 units instead of 100 units); this explains the lower error rate of 0.4% vs the error rate of 0.6% in [2].

As we can see in the table, the use of convolutional layers and max pooling prior to 1D LSTM results in substantial reductions in error rates. However, deeper networks and larger number of hidden units were not able to reduce error rates below 0.25%. Examining the errors in Figure 4 suggests that this may be close to the intrinsic error rate for the UW3 testset we were using in these experiments.

We can also replace the convolutional layers with 2D LSTM layers; this is the kind of model that has previously been used extensively in handwriting recognition. Although this model performs slightly better than the original 1D LSTM OCR method[2], it performs worse than most of the other models tested, including all 1D LSTM methods with normalization and even several of the unnormalized models.

B. Models without Normalization

As indicated in the introduction, we also wanted to compare models with and without geometric normalization to see whether normalization was still beneficial in the presence of deep convolutional layers. Experimental results show that the combination of deep convolutional layers and 1D LSTMs without normalization can perform similarly to simple 1D LSTMs with normalization. Somewhat unexpectedly, however, deep convolutional layers with 1D LSTMs still perform significantly better with normalization than without.

C. Speed

Figure 3 shows the amount of time required for recognizing an average text line using a simple 1D LSTM and a two layer convolutional network followed by a 1D LSTM, both on a 16 core CPU and on the GPU. All measurements were carried out using the PyTorch implementation, but existing open source LSTM-based OCR systems perform about as well as the batch size 1 CPU implementation in the figure. For medium-to-large batch sizes, we see that the GPU implementation performs about 20 times faster than existing systems for recognition. Even relative to multicore CPU implementations, the GPU implementation results in substantial speedups notice the logarithmic scale). Altogether, the GPU-based implementation permits recognition at a rate of over 100 average text lines per second on a modern desktop computer. Note that high-end GPUs are not fully utilized by these networks, since model complexity and batch size is often memory constrained.

D. Degraded and Blurred Images

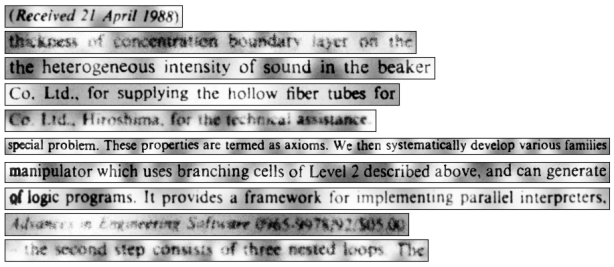
Recognition results on noisy, blurred, and degraded images are shown in Figure 5. This is a much harder problem than the recognition of clean images, with some inputs entirely unrecognizable. The images are similar to those used in [1], though the dataset is different. In these experiments, networks were run directly on the grayscale inputs, as in [1]. Results reproduce what we see on binary LSTMs, with 1D LSTMs performing significantly worse than hybrid networks.

E. Training Progress

As had been observed in other experiments training LSTM networks for OCR, training curves contain large spikes in error rates during training. The combination of convolutional neural networks with 1D LSTMs seemed less prone to diverging than simple 1D LSTM networks for OCR.

F. Most Common Errors

Figure 4 shows the most common errors made by the recognizers. The left table is for a recognizer with line normalization. The most common errors made by that recognizer are comma/period confusions, the insertion and deletion of spaces, and the insertion of spurious characters. Comma/period confusions are common just because the two characters are often nearly indistinguishable in noisy images. Correct recognition of spaces can be difficult in proportionally spaced text, but might improve with more training data. The only error class that is unusual is the the insertion of spurious characters (ϵ-to-something errors). These are an artifact of the decoding process; for example, the letter "m" is occasionally inserted between an "r" and an "n" as an alternative interpretation of the "rn" combination. Recognizers using a recognition lattice or language model avoid such errors by imposing the constraint that every portion of the image can only be part of one valid interpretation. Such errors can potentially be reduced further by using attentional models for decoding, similar to what is used in speech recognition using deep learning, or by



Lstm1d(200)	2.4%
Lstm1d(512)	2.6%
Cr(32) Mp Cr(64) Mp Lstm1d(200)	1.1%

Fig. 5. Recognition of noisy, blurred, and degraded text line images. Left: a random sample of degraded text lines used in the experiments. Right: error rates with different architectures. All recognizers are run directly on the unprocessed grayscale inputs.

introducing language modeling and beam search decoding. However, it should be remembered that these errors are very rare overall.

The right hand figure shows the confusions for the best model not using line projection. The classes of errors are broadly similar to those found in models using line normalization. However, we also see some confusions that obviously result from the lack of absolute size or position information (e.g., "O" vs "o", "S" vs "s"). In addition, some characters for which positional information is important (e.g., single quotes, commas, etc.) are deleted from the output; this is also a result of the decoding process: such characters often have a posterior probability of around 0.5 for each of the two recognition alternatives, and the current, simple decoding process drops both interpretations instead of picking one.

IV. Conclusions

The paper has described a new implementation of a text line recognizer for printed text based on the PyTorch deep learning framework. The implementation can be used to replace the existing text line recognizer in systems like OCRopus, and yields an order of magnitude speedup, allowing around 100 text lines per second to be recognized on a standard desktop machine with a GPU. The greater speed and flexibility of the framework has allowed us to explore and evaluate more complex models for text line recognition than previously feasible.

We have shown that the combination of convolutional layers, max pooling, and 1D LSTM can result in substantially lower error rates than the older 1D LSTM networks. Hybrids of convolutional and LSTM networks also result in substantially better performance on blurred and noisy text. In a simple side-by-side comparison, replacing convolutional layers with 2D LSTM layers in a convolutional-recurrent hybrid network did not result in improved performance.

The paper also addressed the question of whether geometric text line normalization is still helpful when hybrid convolutional-recurrent networks are used, since traditionally, convolutional neural networks with max pooling are often believed to obviate the need for such preprocessing techniques. Experimental results show that hybrid convolutional-recurrent networks without normalization indeed outperform simple 1D LSTM networks with normalization. Nevertheless, the combination of line normalization with convolutional-recurrent hybrids still performs significantly better with line normalization than without.

Tesseract and OCRopus are two commonly used open source OCR systems, both implementing "traditional" segmenting and LSTM-based recognizers. Prior work had already shown that LSTM-based recognizers clearly outperform the segmenting recognizers. The results on different LSTM architectures described in this paper are directly applicable to the LSTM-based versions of OCRopus and Tesseract. The major difference is that the recognizer described in this paper is based on a standard deep learning framework and can take direct advantage of GPU computing, resulting in substantially higher throughput than CPU-based recognizers.

This paper has primarily been concerned with describing a new OCR engine based on PyTorch and cuDNN and characterizing it on a dataset commonly used for OCR research. Although as part of this, we have seen that convolutional-recurrent hybrid networks are now suitable for high performance printed OCR, this paper only explored a small number of possible architectures. Much larger scale explorations is warranted, also in order to determine the best architectures for other scripts; that work remains for future work. The availability of a fast, GPU-based, open source text line recognizer should make it much easier to explore new architectures in the future, and it permits much more complex models to be explored than previously possible.

References

[1] F Asad et al. "High Performance OCR for Camera-Captured Blurred Documents with LSTM Networks". In: *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*. ieeexplore.ieee.org, Apr. 2016, pp. 7–12.

[2] T M Breuel et al. "High-Performance OCR for Printed English and Fraktur Using LSTM Networks". In: *2013 12th International Conference on Document Analysis and Recognition*. ieeexplore.ieee.org, Aug. 2013, pp. 683–687.

[3] Thomas M Breuel. "The OCRopus open source OCR system". In: *Electronic Imaging 2008*. International Society for Optics and Photonics. 2008, 68150F–68150F.

[4] Syed Saqib Bukhari, Faisal Shafait, and Thomas M Breuel. “Towards generic text-line extraction”. In: *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE. 2013, pp. 748–752.

[5] Sharan Chetlur et al. “cudnn: Efficient primitives for deep learning”. In: *arXiv preprint arXiv:1410.0759* (2014).

[6] Ian J Goodfellow et al. “Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks”. In: (20 12 2013). arXiv: 1312.6082 [cs.CV].

[7] Alex Graves and Juergen Schmidhuber. “Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks”. In: *Advances in Neural Information Processing Systems 21*. Ed. by D Koller et al. Curran Associates, Inc., 2009, pp. 545–552.

[8] Alex Graves et al. “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 369–376.

[9] Marcus Liwicki et al. “A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks”. In: *Proc. 9th Int. Conf. on Document Analysis and Recognition*. Vol. 1. 2007, pp. 367–371.

[10] *Neural Nets In Tesseract 4.00*. https://github.com/tesseract-ocr/tesseract/wiki/NeuralNetsInTesseract4.00. Accessed: 2017-03-01.

[11] *PyTorch Home Page*. http://www.pytorch.org/. Accessed: 2017-03-01.

[12] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: (Apr. 2014). arXiv: 1409.1556 [cs.CV].

[13] Ray Smith. “An overview of the Tesseract OCR engine”. In: *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*. Vol. 2. IEEE. 2007, pp. 629–633.

[14] Uwe Springmann and Anke Lüdeling. “OCR of historical printings with an application to building diachronic corpora: A case study using the RIDGES herbal corpus”. In: *arXiv preprint arXiv:1608.02153* (2016).

[15] Uwe Springmann et al. “OCR of Historical Printings of Latin Texts: Problems, Prospects, Progress”. In: *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. DATeCH ’14. New York, NY, USA: ACM, 2014, pp. 71–75.