Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?

Joan Puigcerver
Pattern Recognition and Human Language Technology Research Center
Universitat Politècnica de València
46022 Valencia, Spain
Email: joapuipe@prhlt.upv.es

***Abstract*—Current state-of-the-art approaches to offline Handwritten Text Recognition extensively rely on Multidimensional Long Short-Term Memory networks. However, these architectures come with quite an expensive computational cost, and we observe that they extract features visually similar to those of convolutional layers, which are computationally cheaper. This suggests that the two-dimensional long-term dependencies, which are potentially modeled by multidimensional recurrent layers, may not be essential to achieve a good recognition accuracy, at least in the lower layers of the architecture. In this work, an alternative model is explored that relies only on convolutional and one-dimensional recurrent layers that achieves better or equivalent results than those of the current state-of-the-art architecture, and runs significantly faster. In addition, we observe that using random distortions during training as synthetic data augmentation dramatically improves the accuracy of our model. Thus, are multidimensional recurrent layers really necessary for Handwritten Text Recognition? Probably not.**

I. Introduction

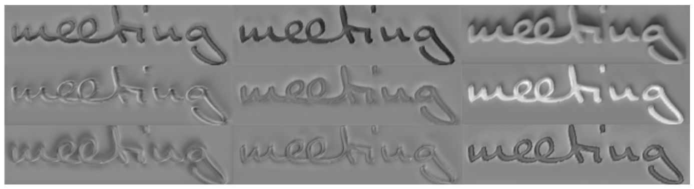
Recurrent neural networks, and particularly Multidimensional Long Short-Term Memory (MDLSTM) networks [1], have been widely adopted by the Handwritten Text Recognition (HTR) community. MDLSTM currently hold the state-of-the-art performance on most (if not all) of the benchmarks used to measure the advances in the field.

The difference between regular Long Short-Term Memory (LSTM) networks [2] and MDLSTM is that, the former introduce a recurrence along the axis of one-dimensional sequences (for instance, the time-axis in speech, or the writing-direction in images), while the latter have a recurrence along two axes (typically the x-axis and y-axis in images). This allows the latter to process two-dimensional data of unconstrained size, potentially capturing long-term dependencies across both axes.

In the HTR field, these networks are typically used to transcribe images of text lines. Multiple two-dimensional MDLSTM layers (2D-LSTM) are stacked in order to extract meaningful features from the images (usually, in combination with other types of layers, like convolutional and pooling layers). Then, the 2D data is transformed into a 1D sequence to obtain the character-level transcription of the input line image. This architecture is, essentially, the core of all the successful approaches to line-level HTR in the recent years [1], [3], [4].

Nevertheless, 2D-LSTM networks are quite computationally expensive, specially when compared to other types of operations like convolutions. When *using a large number of parallel processing units* (i.e. GPUs), the best parallel implementation of a 2D-LSTM layer has a computational complexity of $\mathcal{O}((W+H) \cdot D + C)$, while the computational complexity of a naive implementation of a convolutional layer is $\mathcal{O}(C \cdot S)$. Here, H and W are the height and width of the image, C and D are the number of input/output channels and S is the size of the receptive field of the convolution operation (e.g. 3×3). Typically, in the lower layers of the neural networks, which dominate the computational cost of traditional models, $(W+H) \cdot D + C$ is much larger than $C \cdot S$ (typically by one or two orders of magnitude).

In addition, an inspection of the features extracted by the 2D-LSTM in the lower layers shows that they are visually similar to the two-dimensional convolutional ones (see Fig. 1). This suggests that, in order to extract meaningful features from the images, one does not probably need the large context provided by recurrent layers, at least in the lower layers of the stack.


(a) 2D-LSTM

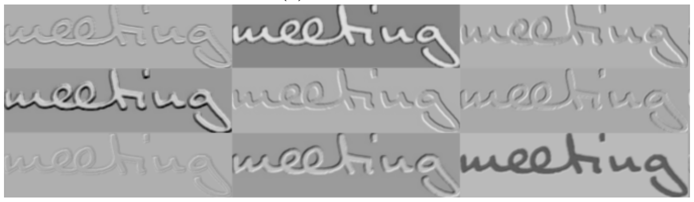

(b) Convolutional

Fig. 1: Randomly selected features extracted after a 2D-LSTM and after a convolutional layers, at comparable positions of the model.

These two observations indicate that 2D-LSTM could be replaced by convolutional layers, at least at some extent in the lower layers, reducing the required computational resources with no (or little) loss on accuracy. Finally, it is reasonable to assume that in the upper layers of the network, one-dimensional recurrent layers might be powerful enough to model the language dependencies, given that languages are well modeled by one-dimensional sequences (of characters).

A. *Contributions*

In this work, we aim to investigate whether MDLSTM networks are strictly required to achieve state-of-the-art performance for line-level HTR, or cheaper layers (in terms of computational resources) can be used instead. We first review the basics of HTR to the less-familiar readers (Section II) and then:

- We present a neural network architecture based on convolutional and 1D-LSTM layers to perform line-level HTR (Section III).
- We provide statistically-sound empirical study showing that our architecture provides similar or better accuracy, when compared to the current state-of-the-art 2D-LSTM architecture, and it is strictly superior in terms of speed (Section IV-B and IV-C).
- We show that by performing adequate random distortions on the training images one can significantly reduce the error rates (Section IV-D).
- We release our software and experimentation pipeline allowing other researchers to fairly reproduce our results[1].

II. Handwritten Text Recognition with Recurrent Neural Networks

HTR aims to recover the handwritten text represented in an input signal. Particularly, in the case of offline HTR, the input signal typically is a (manually or automatically) segmented line of a scanned handwritten document (e.g. handwritten forms, historical manuscripts, etc). In the recent decades, this problem has mostly been addressed statistically by trying to solve the following optimization problem:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \Pr(\mathbf{y} \mid \mathbf{x}) \tag{1}$$

Where $\mathbf{x}$ is the input signal, represented as a sequence of *frames*, x_t, and $\mathbf{y}$ is a sequence of textual symbols, y_l (typically words or characters). Since the real probability distribution $\Pr(\mathbf{y} \mid \mathbf{x})$ is unknown, it is modeled by a parametric distribution $P_\theta(\mathbf{y} \mid \mathbf{x})$, whose parameters are fit according to the Maximum Likelihood Estimation (MLE) criterion, over a set of training samples. Then, the solution to Eq. (1) is approximated by:

$$\hat{\mathbf{y}} \approx \arg\max_{\mathbf{y}} P_\theta(\mathbf{y} \mid \mathbf{x}) \tag{2}$$

Traditional models for $P_\theta(\mathbf{y} \mid \mathbf{x})$ are the composition of Hidden Markov Models (HMM) with Gaussian Mixture Model emissions and n-gram Language Models (LM). Nevertheless, current solutions widely rely on Recurrent Neural Networks. Particularly, Connectionist Temporal Classification (CTC) allows to train a neural network modeling $P_\theta(\mathbf{y} \mid \mathbf{x})$ end to end[2] [5].

[1] https://github.com/jpuigcerver/Laia/

[2] CTC is used to locally optimize $\hat{\theta} = \arg\max_\theta \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} P_\theta(\mathbf{y} \mid \mathbf{x})$

TABLE I: Details of the configuration used in the convolutional blocks of our architecture.

Configuration	Values
Conv. filters	16 – 32 – 48 – 64 – 80
Maxpool (2 × 2)	Yes – Yes – Yes – No – No
Dropout	0 – 0 – 0.2 – 0.2 – 0.2

III. Proposed system

A general overview of the architecture is depicted in Figure 2. In this section we explain the details of each of the blocks of our architecture, and some other details of our system.

A. *Architecture*

1) Convolutional blocks: Each convolutional block contains a two-dimensional convolutional layer (Conv) with a kernel size of 3×3 pixels, with both horizontal and vertical stride of 1 pixel. The number of filters at the n-th Conv layer is equal to $16n$. In order to reduce overfitting, we apply Dropout [6] at the input of some Conv layers (with dropout probability equal to 0.2). After the Conv layer, batch normalization [7] is used in order to normalize the inputs to the nonlinear activation function. We use Leaky Rectifier Linear Units (LeakyReLU) [8] as the activation function in the convolutional blocks. Finally, the output of the activation function is fed to a Maximum Pooling layer (Maxpool) with non-overlapping kernels of 2×2 pixels. The Maxpool layer is commonly used to reduce the dimensionality of the input images. Table I shows the configuration used in each convolutional block. We use a total of 5 convolutional blocks in our architecture.

2) Recurrent blocks: Recurrent blocks are formed by bidirectional 1D-LSTM layers, that process the input image columnwise (see “Columnwise concat” in Figure 2) in left-to-right and right-to-left order. The output of the two directions is concatenated depth-wise (see “Depth concat” in Figure 2). Thus, if D is the number of hidden units in each direction, the output of the BLSTM block has a depth of $2 \cdot D$ channels. Before each 1D-LSTM layer, Dropout is also applied here (with probability 0.5). The number of hidden units of all 1D-LSTM layers is fixed to $D = 256$. We use a total number of 5 of recurrent blocks.

3) Linear layer: Finally, each column after the recurrent 1D-LSTM blocks must be mapped to an output label. In order to do so, the depth is transformed from $2 \cdot D$ to L using an affine transformation (where L is equal the number of characters + 1, for the CTC blank symbol). As we did in the recurrent blocks, Dropout is applied before the Linear layer to prevent overfitting (also with probability 0.5).

B. *Learning*

All parameters of the neural network are trained by minimizing the CTC loss. We use the RmsProp algorithm [9] to incrementally update the parameters of the model using the gradients of the CTC loss on each batch of 16 images.

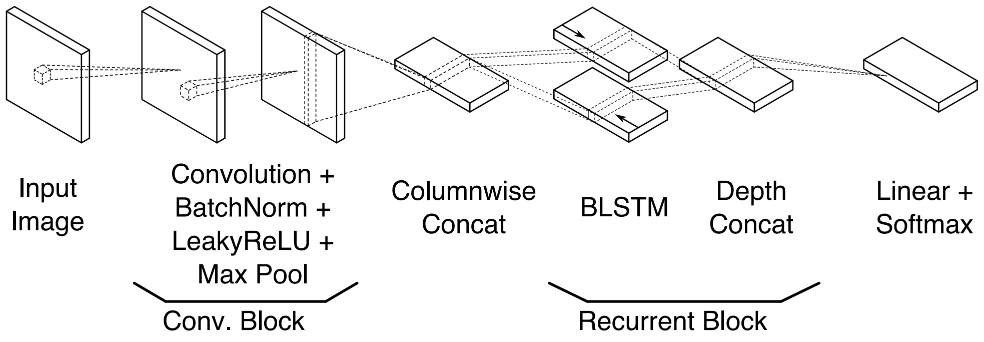


Fig. 2: Architecture of the neural network presented in this work using bidirectional 1D-LSTM.

We use a fixed learning rate equal to 0.0003. In all the in-paper experiments. Training is stopped when the character error rate (CER) on the validation set does not improve after 20 epochs. *We increased this maximum to 80 epochs for the final experiments in Section IV-E.*

C. Dynamic data augmentation

We perform adequate random distortions on the input images, in order to artificially augment the training samples and reduce overfitting. These distortions include: rotation, translation, scaling and shearing (all performed as a single affine transform) and gray-scale erosion and dilation. Each of these operations is applied dynamically and independently on each image of the training batch (each with 0.5 probability). Thus, the exact same image is virtually never observed twice during training.

The parameters controlling each distortion (e.g. rotation angle, scaling factor, erosion kernel, etc.) are sampled from a fixed distribution. Tuning each of them to minimize CER, for each dataset, would be too time consuming. Instead, we use the default parameters provided by our toolkit which have worked considerably well on a variety of scenarios[3].

D. Implementation details

Our toolkit was built using the Lua scripting language. It uses Torch, a well-known deep learning framework. Torch supports both CPU and GPU devices. We also use existing implementations of several operations, when available. This reduced the cost of implementation and debugging. Most notably, we use NVIDIA's convolution and LSTM[4] and Baidu's CTC[5] implementations. For the comparison with the 2D-LSTM models, we used RETURNN [10] and the state-of-the-art architecture used in [4].

IV. Experimental evaluation

A. Datasets

We chose two widely used datasets to experimentally validate our work: First, the IAM database (modern English) [11], compiled by the FKI-IAM Research Group. The dataset is composed of 1 539 scanned text pages, handwritten by 657 different writers and partitioned into writer-independent training, validation and test partitions of 6 161, 966 and 2 915 lines, respectively. The original line images in the training set have an average width of 1 751 pixels and an average height of 124 pixels. There are 79 different characters in the dataset, including the white-space. In order to build a better LM, we also used the Brown, Wellington and LOB text corpora (excluding the text paragraphs present in the IAM test set).

In addition, we used the Rimes dataset (modern French) [12]. Rimes consists of 11 333 training lines and 778 test lines. The original release does not include a separated validation partition, but we sampled 10% of the total training lines for validation purposes. Thus, the final division of the dataset into training, validation and test consists of 10 203, 1 130 and 778 lines, respectively. The original line images in the training set have an average width of 1658 pixels and an average height of 113 pixels. There are 99 different characters in this dataset.

B. Methods for statistically-sound empirical testing

One of the most popular tools for statistical testing, is the comparison of the confidence intervals (CI) of the studied statistic (e.g. the word error rate, CER, etc.). There are several ways of computing CI, depending on the assumptions that one makes about the statistic's distribution. For instance, a usual assumption is that error rates in lines or paragraphs follow a normal or a binomial distribution. Unfortunately, they require assumptions that do not hold in many scenarios, and/or do not compute a CI over the error rate of the whole partition, which is the statistic commonly used in the field. Instead, we compute the CI using non-parametric bootstrapping [13], which has the advantage that does not make any assumption about the distribution, and presents an interval over the error rates at a partition level.

Nonetheless, CI can lead to misinterpretations: it is possible that two CI *are not disjoint* and the differences are still significant. In these cases we compute the p-value over the differences in the error rates at paragraph-level, using the standard Student's t-test[6]. We followed this approach because

[3]In fact, these parameters were chosen by visually inspecting a few training lines from IAM and never changed.

[4]https://developer.nvidia.com/cudnn

[5]https://github.com/baidu-research/warp-ctc

[6]We also used non-parametric bootstrapping and Wilcoxon signed-rank tests to compute p-values, with identical significance results.

a) it is well understood and known, b) it is robust with non-normally distributed data, and c) it is included in all statistical software packages. Although the error rates are far from being normally distributed, the *differences of the error rates at paragraph-level are closer to a normal distribution*.

Finally, in some cases it is sufficient to show the non-inferiority of one method. This requires to choose a non-inferiority margin, Δ. According to [14], a commonly used *strict* relative margin is 10%. Instead, we used an even stricter relative margin of 5%, meaning that the null hypothesis assumes that our method performs at least 5% worse than the 2D-LSTM.

Thus, the null and alternative hypotheses in our tests are:

$$\underbrace{\begin{cases} H_0 : e_1 = e_2 \\ H_a : e_1 \neq e_2 \end{cases}}_{\text{Superiority test}} \qquad \underbrace{\begin{cases} H_0 : e_1 - e_2 \geq \Delta \\ H_a : e_1 - e_2 < \Delta \end{cases}}_{\text{Non-inferiority test}}$$

Here, e_1 is the (character or word) error rate achieved by our 1D-LSTM architecture, e_2 is the achieved by the 2D alternative, and $\Delta = 0.05e_2$.

C. Comparison with state-of-the-art 2D-LSTM architecture

Notice that comparing 1D-LSTM against 2D-LSTM architectures is not a trivial task, for many reasons. We did our best to fairly compare both architectures, trying to isolate the effect of other variables. The following aspects were considered:

- We did not use batch normalization nor random distortions in our architecture, for a fair comparison.
- We resized the images only for the 1D architecture. Since the height that we use (128 pixels) is slightly larger than the original average height, we are in a slightly worse average scenario against our architecture in terms of computing/memory requirements, in comparison to the 2D-LSTM.
- One could argue that the reduction in CER/WER is because our architecture has many more parameters than the 2D-LSTM. However, the original authors [4] showed that the accuracy of *the 2D-LSTM model did not benefit from more parameters*. Additionally, the memory used by the parameters is insignificant compared to the available GPU memory (40MB vs. 12GB), and most of the space is actually used to allocate activations and buffers during back-propagation. Consequently, we do not consider the number of parameters an important factor, but the accuracy and speed achieved, and the memory needed.
- We assume the independence of some factors. In both cases we use a fixed learning rate of 0.0003 and the RmsProp algorithm [9]. We also assumed that the effect of an explicit LM is independent of the architecture, thus in order to simplify the comparison, we did not use any explicit LM in this section (i.e. no external n-gram LM).

Table II shows the character and word error rates (without any external LM) on the training and validation partitions and the average training runtime per epoch.

TABLE II: Comparison of the state-of-the-art 2D-LSTM architecture (see [4]) and our proposed model. We show the error rates on each partition, the average duration of one training epoch, the maximum amount of GPU memory used, and the total number of parameters of the architecture. No explicit language model was used. Bootstrapped confidence intervals at 95%.

		1D-LSTM	2D-LSTM
CER	Validation	**5.1** [4.6–5.7]	5.7 [5.1–6.3]
	Test	**8.2** [7.6–8.9]	8.3 [7.7–9.0]
WER	Validation	**17.9** [16.3–19.7]	20.7 [18.8–22.5]
	Test	**25.4** [23.9–27.0]	27.5 [26.0–29.0]
Avg. Runtime (min.)		**3.8** [3.8–3.8]	24.3 [24.3–24.3]
# of parameters (Mi.)		9.3	2.6
Max. Memory (GB)		10.5	10.6

(a) IAM

		1D-LSTM	2D-LSTM
CER	Validation	**3.0** [2.6–3.5]	3.7 [3.1–4.4]
	Test	**3.3** [2.7–4.1]	4.0 [3.3–4.8]
WER	Validation	**12.4** [10.9–14.0]	16.5 [14.6–18.6]
	Test	**12.8** [10.9–14.8]	17.7 [15.4–20.2]
Avg. Runtime (min.)		**5.0** [5.0–5.0]	45.9 [45.8–46.0]
# of parameters (Mi.)		9.6	2.6
Max. Memory (GB)		10.3	11.5

(b) Rimes

First, observe that the CI over the average running times do not overlap, being the running times of the 1D-LSTM architecture much smaller than the 2D-LSTM (6×–7× speedups). Also, the maximum memory used during training is similar in both cases. One could argue that the differences in running times are due to using different interpreted languages (i.e. Lua vs. Python). However, most of the running time is due to code executed in the GPU, written in CUDA in both cases.

Although the CI are overlapping in most cases, the WER p-value is lower than 0.0001 in all partitions and datasets, under the null hypothesis that both models have the same mean (see Section IV-B). This is far below the standard $\alpha = 0.05$, meaning that we can assume that the architecture based on 1D-LSTM has a significantly lower WER in both datasets. On the other hand, the CER p-value under $H_0 : e_1 = e_2$ is also below $\alpha = 0.05$, except for the test partition of IAM. However, the non-inferiority test gives a p-value equal to 0.0013. Thus, although the superiority of our method cannot be accepted with a sufficiently low α (i.e. Type-I error), the non-inferiority can. This scenario (the CER on the test set of IAM) is the only where our architecture has not been shown to be superior.

D. Effect of dynamic data augmentation and batch normalization

Table III shows the effect of batch normalization and the data augmentation strategy described in Section III-C. We did not find any statistically significant difference when using batch normalization alone, with respect to the 1D-LSTM baseline. In fact, the results were slightly worse on the validation partitions. However, batch normalization helps to reduce the error rates when applied with random distortions on the input images. On IAM, which has a smaller training set than Rimes, the error rates are significantly reduced in all cases, when compared to the baseline. In summary, the CER on the test set decreases from 8.2% to 6.2% on IAM, and from 3.3% to 2.6% on Rimes. Furthermore, WER is reduced from 25.4% to 20.2% on IAM, and from 12.8% to 10.7% on Rimes.

TABLE III: Effect of using random distortions during training and batch normalization (BN) on top of the baseline 1D-LSTM architecture. No explicit language model was used. Bootstrapped confidence intervals at 95%.

System	CER (%) Validation	CER (%) Test	WER (%) Validation	WER (%) Test
Baseline	5.1 [4.6–5.7]	8.2 [7.6–8.9]	17.9 [16.3–19.7]	25.4 [23.9–27.0]
+ BN	5.2 [4.7–5.8]	8.3 [7.9–8.6]	18.5 [16.9–20.2]	24.9 [23.4–26.4]
+ Distortions	4.4 [3.9–4.9]	6.4 [5.8–6.9]	15.5 [14.0–17.0]	20.8 [19.6–22.1]
+ BN + Distortions	**4.1** [3.6–4.5]	**6.2** [5.7–6.8]	**14.6** [13.1–16.1]	**20.2** [19.0–21.4]

(a) IAM

System	CER (%) Validation	CER (%) Test	WER (%) Validation	WER (%) Test
Baseline	3.0 [2.6–3.5]	3.3 [2.7–4.1]	12.4 [10.9–14.0]	12.8 [10.9–14.7]
+ BN	3.1 [2.6–3.7]	3.2 [2.5–3.9]	12.6 [11.0–14.3]	12.7 [10.9–14.6]
+ Distortions	**2.4** [2.0–2.8]	**2.6** [2.1–3.3]	10.8 [9.3–12.3]	10.8 [9.0–12.6]
+ BN + Distortions	2.5 [2.1–2.8]	**2.6** [2.0–3.2]	**10.5** [9.3–11.8]	**10.7** [9.0–12.5]

(b) Rimes

E. Comparison with previous publications

In order to fairly compare our contributions with previously published works that use explicit language models. First, We transform the label-posteriors output by the neural network, $P(\mathbf{l}_i \mid \mathbf{x}_i)$, into *pseudo*-likelihoods, according to Eq. (3). Then, we combine them with an interpolated n-gram model built using SRILM. Decoding is then performed using Kaldi [15], with a beam width equal to 65.

$$p(\mathbf{x}_i \mid \mathbf{l}_i) = \frac{P(\mathbf{l}_i \mid \mathbf{x}_i) \cdot p(\mathbf{x}_i)}{P(\mathbf{l}_i)} \approx \frac{P(\mathbf{l}_i \mid \mathbf{x}_i)}{P(\mathbf{l}_i)^{\gamma}} \tag{3}$$

For both IAM and Rimes we used $\gamma = 0.2$. In addition, we scale the likelihoods $p(\mathbf{x}_i \mid \mathbf{l}_i)$ in order to combine them with the LM scores. The acoustic scale factor for Rimes was set to 1.90, combined with a 4-gram word LM. For IAM, it was set to 1.79, for a 3-gram word LM. These parameters were adjusted to minimize the WER on the validation set.

We used word n-grams for both datasets, with modified Kneser-Ney discounting [16] and interpolation. For IAM we used 3-grams and a vocabulary of 50 000 tokens, which achieved a perplexity of 272.2 and 304.0, and an out-of-vocabulary (OOV) rate of 3.0% and 2.9%, on the validation and test sets, respectively. For Rimes, 4-grams were used with the whole training vocabulary (5 048 tokens) achieving a much lower perplexity, 21.9 and 21.1, and OOV rates, 2.9% and 2.8%, on the validation and test sets.

Table IV shows the results of the comparison between our system and previously published results. Notice that the use of an explicit LM reduced the error rates dramatically for IAM, but not much for Rimes. In the latter scenario, our system has a lower CER when compared to the previous best publication, even without using an explicit LM (2.3% vs. 2.8%), and the WER with no LM is similar to the state-of-the-art (9.6% in both cases). When the 4-gram LM is used, the WER decreases to 9.0%. However, notice a small increase in the CER (from 2.3% to 2.5%). These differences are very likely not significant, but previous publications did not offer data to make this comparison. The achieved CER and WER on IAM are considerably larger than the current state-of-the-art. However, our system performed better when the LM was not considered (recall that both architectures were compared under same conditions in Section IV-C). Hence, these differences are very likely due to the distinct LM. In both [4], [17], they use a combination of word and character n-grams, in order to reduce the effect of OOV words. However, we used just a word n-gram LM aiming to keep the experiments simple.

TABLE IV: Comparison of the character and word error rate (%) on IAM and Rimes paragraphs achieved in this work with previously published competitive state-of-the-art results. Bootstrapped confidence intervals at 95%.

System	CER (%) Validation	CER (%) Test	WER (%) Validation	WER (%) Test
Ours (no LM)	3.8 [3.4–4.3]	5.8 [5.3–6.3]	13.5 [12.1–14.9]	18.4 [17.4–19.5]
Ours (word LM)	2.9 [2.5–3.3]	4.4 [3.9–4.8]	9.2 [8.1–10.2]	12.2 [11.4–13.2]
Voigtlaender et al. [4]	**2.4**	**3.5**	**7.1**	**9.3**
Doetsch et al. [17]	2.5	4.7	8.4	12.2
Pham et al. [3]	3.7	5.1	11.2	13.6

(a) IAM

System	CER (%) Validation	CER (%) Test	WER (%) Validation	WER (%) Test
Ours (no LM)	**2.2** [1.8–2.5]	**2.3** [1.8–3.0]	9.6 [8.3–11.0]	9.6 [8.1–11.3]
Ours (word LM)	2.3 [2.0–2.7]	2.5 [1.9–3.2]	**8.9** [7.9–10.0]	**9.0** [7.5–10.4]
Voigtlaender et al. [4]	—	2.8	—	9.6
Doetsch et al. [17]	—	4.3	—	12.9
Pham et al. [3]	$3.3^{\dagger}$	3.3	$13.1^{\dagger}$	12.3

(b) Rimes

V. Discussion and related works

Convolutional layers and 1D recurrent layers have already been used in the past for HTR. For instance, in [18] they combined ConvNets with HMMs, using an hybrid architecture instead of using CTC for training the system end to end. In addition, 1D-LSTM and CTC have also been used on top of handcrafted features [17]. The most similar published work to ours is [19], where they use a similar architecture and train with CTC. However, they only applied it for isolated word recognition.

In this work, we provided multiple evidences that multidimensional recurrent layers may not be necessary to achieve good accuracy for HTR. We provided some intuitive explanation: although MDLSTM are in principle more powerful than ConvNets, this additional power seems not be necessary for HTR, given that similar features are learned. Moreover, a statistically-sound analysis of the experimental results also supports this claim, for two widely used datasets.

Nevertheless, a word of caution is required: Notice that the databases that we used are just a (very small) subset of all possible scenarios for HTR. Also, the system architecture might not be independent with other factors (e.g. LM), but we assumed it for practical purposes. Thus, many more experiments need to be conducted to strengthen the statistical power of our conclusions. Anyhow, the experimental results support the intuitive explanation, and are very positive due to the huge reduction in computational cost.

Finally, we would like to encourage future publications to make use of the (or a similar) statistical analysis that we conducted. We are aware that statistical testing is not free of controversy: p-hacking and publication bias are just two examples. However, it is the most effective way (if properly used) of knowing whether some observations are due to a fundamental difference between two approaches, or just due to the particularities of the data.

Acknowledgment

I am in debt with Paul Voigtlaender and Patrick Doetsch, who generously gave additional details from their work. I would also like to thank my supervisors, Enrique Vidal and Alejandro H. Toselli, and my long-time labmate Dani Martín-Albo, for their comments on this article. NVIDIA Corporation kindly donated the Titan X GPU used for this research. Finally, this work was partially supported by the Spanish MEC under grant FPU13/06281, and through the EU projects: HIMANIS (JPICH program, Spanish grant Ref. PCIN-2015-068) and READ (Horizon-2020 program, grant Ref. 674943).

References

[1] A. Graves and J. Schmidhuber, "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 2009, pp. 545–552.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout Improves Recurrent Neural Networks for Handwriting Recognition," in *2014 14th International Conference on Frontiers in Handwriting Recognition*, Sept 2014, pp. 285–290.

[4] P. Voigtlaender, P. Doetsch, and H. Ney, "Handwriting Recognition with Large Multidimensional Long Short- Term Memory Recurrent Neural Networks," in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, Oct 2016, pp. 228–233.

[5] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 369–376.

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[7] Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[8] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, no. 1, 2013.

[9] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, 2012.

[10] P. Doetsch, A. Zeyer, P. Voigtlaender, I. Kulikov, R. Schlüter, and H. Ney, "RETURNN: the RWTH extensible training framework for universal recurrent neural networks," *CoRR*, vol. abs/1608.00895, 2016.

[11] U.-V. Marti and H. Bunke, "The IAM-database: an English sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, 2002.

[12] E. Augustin, J.-m. Brodin, M. Carré, E. Geoffrois, E. Grosicki, and F. Prêteux, "RIMES evaluation campaign for handwritten mail processing," in *Proc. of the Workshop on Frontiers in Handwriting Recognition*, no. 1, 2006.

[13] B. Efron, "Better bootstrap confidence intervals," *Journal of the American Statistical Association*, vol. 82, no. 397, pp. 171–185, 1987.

[14] S. Wellek, *Testing statistical hypotheses of equivalence and noninferiority*. CRC Press, 2010.

[15] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi Speech Recognition Toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011, iEEE Catalog No.: CFP11SRW-USB.

[16] S. F. Chen and J. Goodman, "An Empirical Study of Smoothing Techniques for Language Modeling," in *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ser. ACL '96. Stroudsburg, PA, USA: Association for Computational Linguistics, 1996, pp. 310–318. [Online]. Available: http://dx.doi.org/10.3115/981863.981904

[17] P. Doetsch, M. Kozielski, and H. Ney, "Fast and Robust Training of Recurrent Neural Networks for Offline Handwriting Recognition," in *2014 14th International Conference on Frontiers in Handwriting Recognition*, Sept 2014, pp. 279–284.

[18] T. Bluche, H. Ney, and C. Kermorvant, "Feature extraction with convolutional neural networks for handwritten word recognition," in *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE, 2013, pp. 285–289.

[19] B. Shi, X. Bai, and C. Yao, "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, no. 99, pp. 1–1, 2016.

†Rimes does not have a predefined validation set, so the validation sets are likely to be different.