

Offline Handwritten Recognition : Influence of the Baseline information

Killian Barrere * and Florent Bartoccioni* *supervised by* Bertrand Coüasnon†

*ENS Rennes, Univ Rennes, France

†Univ Rennes, CNRS, IRISA, France

Abstract—Known as Handwritten Text Recognition (HTR), the task of transcribing handwritten inputs into a sequence of numerical character is still an open problem. The offline variant of HTR, that is to say, making transcription from an image of a text, has been extensively researched. But one point that has not been studied is whether or not the position of the text baseline is useful for this problem. Therefore, we provide in this paper different experiments that aim to test the impact of the text baseline’s informations. Then we show our results and discuss lengthily about what can be improved on these experiments.

Index Terms—Deep Learning, Handwritten Text Recognition, Offline Text Recognition, Baseline.



1 INTRODUCTION

Even in our digital age there is still a non-negligible number of documents that are handwritten. Storing handwritten documents and searching through them efficiently are both difficult tasks. We define Handwritten Text Recognition (HTR) as the ability of transcribing handwritten inputs (paper documents, touch-screens or other devices) into a sequence of numerical character. Two variants of HTR exist, *Online* and *Offline*. For the former the input is captured while the person is writing. For example the movements of the pen tip is registered by a graphics tablet. Meanwhile, for the later, the input is a picture of the entire text (the raw values of pixels). Online HTR is known as being easier since more information is available such as the velocity of the tip, the pressure applied and the points drawn as a sequence through time. We narrow down the scope of the project by only working on the offline variant of HTR. Therefore, by simplicity and from this point, we will refer *Offline HTR* as only *HTR*.

Like other subjects, HTR was struck by the Deep Learning wave. Before, HTR used to work with Hidden Markov Model (HMM) as an entire system, but the new Deep Learning pipeline outperform these methods [1] both in result and in practicality. Nowadays, almost every HTR system uses Neural Networks. Specific types of Neural Networks have been created and modified for the HTR problem.

However, one thing that seems to be not so researched is the influence of the text baseline’s position. We think that adding the knowledge of a the text baseline could help the Neural Network to recognize letters (and texts at the end), especially those who cross the standard character size such as a p, l, etc.

In this project, we will first try to build an architecture that stick to state of the art results. Then we will evaluate the influence of the text baseline position. Especially, we will have to test different ways to add the text baseline’s

information to the architecture.

The section 2 of the paper consists of an historical background of Text Recognition. It gives an overview of the progress made in the domain from Hidden Markov Models to Deep Learning. We then discuss about the material we used and explain the experiments we wanted to test in section 3. Finally, in section 4 we lengthily discuss about the obtained results and how our experiments could be improved.

2 RELATED WORKS

The HTR systems studied here all follows the classical architecture composed of three main modules:

- Preprocessing module, in charge of deslant, deskew, enhance degraded images and reduce variability of text styles. Data augmentation is also done in this module.
- Feature extraction module, where a feature vector sequence is computed as the representation of the handwritten text line image.
- Decoding module, which estimate the most likely character sequence for the extracted feature vector sequence.

2.1 Feature Extraction

Hidden Markov Models (HMM), combined with the Viterbi’s algorithm [2], used to be the most competitive solution for HTR. But HMMs applied as in old pipelines have two major flaws. Firstly, the features extraction had to be designed by hand which required more development time. Secondly, HMMs worked at the level of a pixel column (i.e columnwise). Hence, the prediction could only make use of a limited context as a character or a word is made up of several columns of pixels.

To address this issue new pipelines using Neural Networks (NN) have been introduced [1]. This allowed an automatic extraction of features and led to outperforming the prior methods [1] [3]. Nowadays, almost every pipeline for HTR uses Deep Learning as a feature extraction module.

Specific Neural Networks architectures have been created and improved according to the specifications of HTR. To extract features from images, the Convolutional Neural Networks (CNN) replaced hand crafted feature extraction [4] [5] showed huge improvements. A CNN will learn to recognize local features (e.g., lines, curves, etc.) across space and then learn to combine these components to recognize larger structures (e.g., letters, comma, etc.).

After the CNN comes a Recurrent Neural Networks (RNN) which use recurrent connections that allow a ‘memory’ of previous inputs to persist in the network’s internal state, which can then be used to influence the network output. In our case we use a specific kind of RNNs called Long Short Term Memory (LSTM). The key point of LSTMs is that at each step in the sequence it will decide what “remember” and what to “forget”. That is to say, which extracted feature will affect the state of the recurrent neuron and which won’t. This allows us to identify long term dependencies among the extracted features.

Intuitively, the convolutional layers extract local features while the recurrent layers captures long-term dependencies between these extracted features which allow us to make predictions on the word or line level.

2.2 Bidirectional RNN and Multi-Dimensional RNN

However, RNNs (and hence LSTMs) don’t make a full use of all the available context. Actually, the current state is only computed from the previous state while we could use the posterior content. In this regards, Bidirectional RNN (BRNN) [6] and Bidirectional LSTM (BLSTM) [7] [8] [9] [10] [11] have been introduced. Basically, BRNNs are two RNNs put in parallel and in opposite directions. This allows the network to express the current state in function of the prior and posterior states. Alex Graves took it further by introducing Multi-Dimensional LSTM (MDLSTM) [12] where the network use the context in both directions across all dimensions. This kind of architecture achieved the most competitive results in the field [1] [3] [13] [14] [15]. Nevertheless, particularly for MDLSTMs, these kind of results are achieved at the cost of an increase in training time.

System	CER (%)		WER (%)		Training time (h)	
	Validation	Test	Validation	Test		
BLSTM	Laia [16]	2.9	4.4	9.2	12.2	37
	Doetsch et al. [17]	2.5	4.7	8.4	12.2	/
MDLSTM	Voigtlaender et al. [14]	2.4	3.5	7.1	9.3	237
	Pham et al. [13]	3.7	5.1	11.2	13.6	/

TABLE 1: Results on IAM dataset

The table 1 presents a comparison of the results and training time between MDLSTMs and BLSTMs architectures. The results are significantly better for the MDLSTMs. However, because of the extensive use of LSTMs in MDLSTMs, the architecture in [14] has a training time increased by a factor of 6 compared to the one in [16]. Last but not least, the question of the usefulness of MDLSTMs in

HTR compared to BLSTMs have been lengthly discussed in [16].

2.3 Decoding and Language Model

Finally, the output of the RNN has to be transcribed into a sequence of character, this step is called *decoding*. To this end, HMMs were used as the so-called Hybrid HMM [18] model. However, this method could not be trained as a whole and required presegmented training data. Nowadays, Connectionist Temporal Classification (CTC) [19] is generally used as an output layer. It allows temporal classification with RNNs without an explicit segmentation of the input. Furthermore, the CTC provides an objective function which gives an end-to-end model (image to text) that can be trained with backpropagation. Deep BLSTM an MDLSTM trained by using a CTC objective function will learn both local character image dependency for character modeling and long-range contextual dependency for implicit language modeling.

To further improve the model results, we usually use a Language Model (LM) [11]. This part is able to decrease the error rate of the network by constraining its output using the knowledge of the vocabulary on which the LM as been trained on (see table 3 and 2). Some examples of methods used as LMs are Weighted Finite-State Transducers (WFST) [11], Neural Network LM [20] and LSTM LM [21].

However, due to the significant impact that LMs have on final results (see tables 2 and 3) and because of the diversity in existing methods we will only focus on the results at the output of the Network (i.e CTC prediction without LM). By doing this we will keep the results on the influence of the text baseline more fair as the methods used for the LMs are not always detailed in research papers.

System	CER (%)		WER (%)	
	Validation	Test	Validation	Test
BLSTM Laia [16]	3.8	5.8	13.5	18.4
MDLSTM Pham et al. [13]	7.4	6.3	27.3	43.9

TABLE 2: Results on IAM dataset **without LM**

System	CER (%)		WER (%)	
	Validation	Test	Validation	Test
BLSTM Laia [16]	2.9	4.4	9.2	12.2
MDLSTM Pham et al. [13]	3.7	5.1	11.2	13.6

TABLE 3: Results on IAM dataset **with LM**

3 EXPERIMENTAL SETUP

In this section we will describe our choices for the recognition system considered, more particularly the NN architecture. The dataset we tried is introduced, and we will also discuss about how we injected the information of the text baseline in the architecture.

3.1 Chosen architecture

For the advantages cited in section 2 we chose a Deep Neural Network (DNN) architecture with CTC as an output

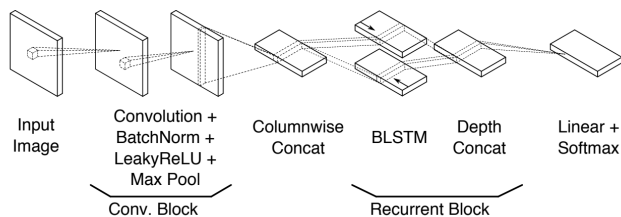


Fig. 1: LAIA architecture [22]

layer. This allowed us to have an end-to-end model that can be trained without an explicit segmentation of the text image input.

Our final choice was LAIA [22], which is a DNN architecture consisting of a block of convolutions followed by a block of BLSTMs. Although it has a simpler architecture, its results are really close to the best performing architectures which use hierarchical MDRNNs [23] (see table 1). It is an easy-to-use, fast-to-train network (approximately 40h on a Nvidia GeForce GTX 1080Ti). The author provides an implementation in Lua with several scripts to prepare the dataset and to compute the character/word error rate. Therefore, this allowed us to easily reproduce the paper’s results from which we compared our experiments.

Moreover, we believed that such a simple architecture could be more influenced by the injection of the text baseline information. Actually, since MDLSTMs use the context on both directions in both dimensions of the image, it is less sensitive to variation in input. Whereas BLSTMs could be more influenced by the vertical position of the words. According to this assumption, the evaluation of our experiments should be more significant with BLSTMs than MDLSTMs.

But with the choice of simplicity also comes some issues. It is sometimes hard to understand what really happens during the preparations of the images and the training of the networks. We encountered a lot of difficulty when it comes to use complex experiments. This encouraged us to re-implement the architecture with TensorFlow (a framework we knew). Firstly, we intended to do it as an educative purpose (to have a better understanding of how the whole pipeline works). Secondly, to have a better control over the architecture. We initially wanted to integrate recent advances such as inception module [24] [25] or skip connections [26] as well as a way to inject the text baseline information directly in the architecture. We tested our implementation on MNIST (28x28 images of digits) and got an accuracy > 98%. However, we learned the hard way, on other datasets with bigger images, that TensorFlow’s LSTM implementation is inefficient in terms of memory and execution time. Because of the specific conditions we were under during this project, we could not continue with our work on TensorFlow and we chose to stick with the original architecture and implementation of Laia [22].

3.2 Database

As it was already implemented in the LAIA framework, we worked with the IAM [27] database. It contains more than 10000 lines of handwritten text in modern style English and is used as a reference for HTR. The number of writers

is also high enough for the network to not overfit to this particular dataset. We have used, what Laia’s author calls, Aachen’s partition of the dataset. That is to say, the dataset was splitted according to the following statistics:

- Train: 6161 lines from 747 forms.
- Validation: 966 lines from 115 forms.
- Test: 2915 lines from 336 forms.

3.3 How to represent the baseline

In this section we will discuss about how we injected the text baseline information. Each of the proposed methods have been tested in 4 in order to evaluate their relevance. To detect the different lines of the text we used a software developed by IRISA’s team Intuidoc which have been tested in cBAD competition.

3.3.1 Add the baseline on the images

Our first idea was to modify the images of the database so that they will include the baseline position in the value of the image’s pixels. With that also comes many ways to change the pixels values of the image.

Our text images are initially encoded in grayscale, that is to say in a single channel image. A first possibility is to work directly on this channel to include the line on it as pixel’s values. We can test different parameters on how to introduce the line on a single channel but with a training of two days and a limited amount of time, it is difficult to test all of these parameters. As the text is written black (pixel value close to 0) on white (pixels value close to 255), we can therefore try several values for the baseline. A value close to text, a medium value, or another one. Also, we can think of whether or not the baseline should overwrite the text. By doing that, some informations are hidden by one layer.

A second idea that could solve the problem of the merged text and text baseline would be to use additional channels. We could use separate channels for the text (unchanged) and other ones to add the baseline information. The second channel could only contain the baseline while a third one could also work as the first experiment, with text and baseline on this same channel. By doing that, because the convolutions use different weights for each channel, we hope that the network could retrieve the informations of the text, the baseline, or other informations without worrying about the loss of informations caused by the overlapping of text and baseline.

One extra parameter for these experiments is the thickness of the baseline. A too thin baseline could be ignored whereas if it is too thick, it can overlap with the text.

3.3.2 Normalize the text

In this method, the idea is to reduce the variability in the data inputs. Rather than modifying the images by adding the values where the baseline should be placed, we can try to modify the scale and positions of different parts of the text. We use the detected text baseline and the text midline to straighten the text, fix its position and scale it uniformly. By doing so, we reduce the vertical variability of the text.

This should improve the accuracy of the BLSTM as it can't modelize along the vertical axis like an MDLSTM would.

The two detected lines split the text in three part: the bottom (crossed by the letters q,p,j, etc.), the body and the top (crossed by t,l,k,etc.). We could apply many different transformations to these parts, with different risks. For example, we could apply a stronger scaling factor to the text body in order to emphasize this part. But if the transformation is not the same in the different parts of text, it may be distorted which could lead to a loss of informations on the connections of the text.

3.3.3 Align the text

On the same principle, we can use the text baseline to align the text. We could only use the text baseline and not the midline anymore. This time rather than scaling the text, we only shift each column so that the detected text baseline is aligned with a fixed vertical position.

This much simpler solution applied less transformation to the image of the text. Therefore, the results could be totally different from the last experiment in which we could have a lot of noise introduced in the images.

3.3.4 Include the baseline in the Network

One last idea, instead of modifying the images, would be to add the baseline position information directly in the architecture. Not in the image but rather represented as numbers concatenated with some of the network layers. This numerical input should represent the baseline position and may somehow affect the results of the pipeline. We could add this information directly as an input of the network or latter on the architecture. But in order to do that we have to be able to modify the architecture.

4 EXPERIMENTS AND RESULTS

In this section we provide a detailed analysis of the experiments and a critical point of view of the different results.

4.1 The experimentation Protocol

In order to measure the impact of the added text baseline, we first wanted to have a reference without the information of the baseline position to which we can compare our experiments. We chose to run all experiments with LAIA, which is presented in section 3.1. We also decided to run our tests on the IAM database (introduced in section 3.2). This offers us a light architecture with a fast training of 40 hours on this database. Also, some preprocessing of the images of the dataset is already available in LAIA to better start our experiment. This preprocessing is trying to deskew the images, rescale them while also reducing the amount of noise in the images.

To test an experiment, we started with an unmodified dataset IAM. We then use the provided scripts to improve the quality of the dataset. On this preprocessed dataset we then applied our desired modification according to the specifications of our experiments. Finally, we trained a new model, tested it and compared its results against the reference.

4.2 Add the line on the same channel as the text

This experiment tries to introduce the text's baseline as it is presented in section 3.3.1. We wanted to add the baseline by keeping the text unchanged and modify the pixels of the same channel where the line should be added.

As described in our protocol 4.1, we first applied a preprocessing of the text image, which aims to improve the quality of the dataset. Then, we detected the text baseline and added it on the text images. The images are encoded in grayscale with the text in dark gray (with a value of the pixels close to 0). The text baseline was added by changing the corresponding pixel to a fixed value of 128 (in gray). Only the pixels without text on it are changed to this specific value, as if the line was behind the text. The figure 2 shows an image of the IAM dataset with an added line.

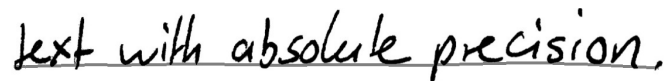


Fig. 2: Example from the IAM [27] database on which the baseline was added by changing the value of the pixels to a fixed gray value (here 128).

We then trained the Neural Network architecture, and compared the results to the reference. The figure 3 compares the Confidence Intervals of both of them.

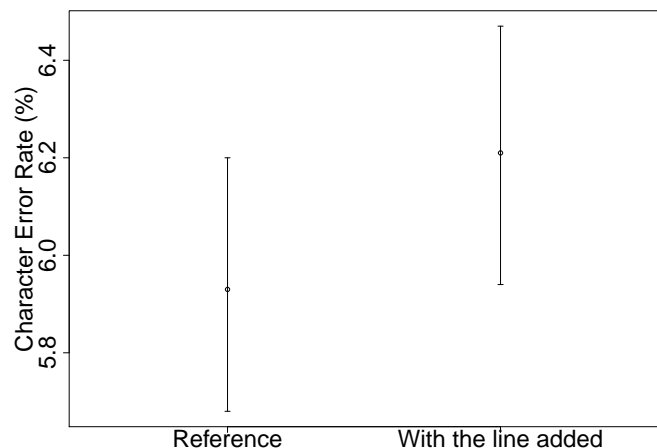


Fig. 3: Comparison between the reference's results and the first experiment 3.3.1. We compared the Confidence Intervals of the Character Error Rate for both of them. A lower value is better.

At first sight, adding the baseline this way seems to increase the error rate, but the difference is only of 0.2% may not be significant.

We compared our results with a *Student t test*. To do that we first introduced the null hypothesis: **Null Hypothesis 1**. There is no particular difference between the repartitions of the two results.

We tested it and got a *p-value* of 0.14 which is superior to the desired error risk of 5%. Therefore, we could not reject the null hypothesis. It cannot be accepted, but we admitted that the observed differences were due to some random factors. The random initialization of the NN's weights seems to be the main random factor. We also thought that adding

text with absolute precision,

text with absolute precision

Fig. 4: From top to bottom, the first channel contains an unmodified text image from the IAM [27] database, the second only the baseline in black (pixel value of 0) and the third is both the text and the line still in black.

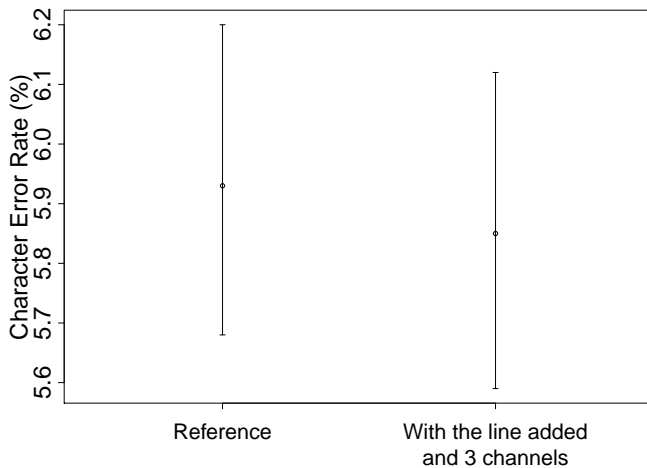


Fig. 5: Comparison of the Character Error Rate Confidence Intervals for both the reference and the experiment in which we added the baseline by using multiple channels. A lower value is better.

the line may confuse the recognition process because the line is on the same channel as the text.

4.3 Add the line on other channels

We then tried another experiment where this time the line is represented in other channels. Figure 4 shows how we separated the line information from the text. We started with the first channel containing only the text. The second channel contains only the text’s baseline. Lastly, we decided to use a third channel that contains both the text and its baseline. By doing that, we hoped to have a significant gain compared to the last experiment, where a single channel could be a problem. The pixel’s value of the text baseline has been set to value of 0 (black). We could also try a different value of the text baseline like 128 in the last experiment, but we lacked time for further experimentations.

Training the architecture with the same parameters gave us the results presented in the figure 5. This time, we observed that the error rate is a bit lower, which mean the architecture may be better this time. But still, the difference is very small. We, again, made a comparison with a *Student t test*, with the same *Null hypothesis*. With these parameters, we got a *p-value* of 0.66. According to this value, we cannot reject the *Null hypothesis*. But we can still admit that the difference is due to the weight initialization.

4.4 Normalize the text based on the text baseline and midline

As adding the line by modifying the pixels’ value seemed to led to close to reference results, but non significant, we decided to try something different.

In this experiment, we will not add the line by modifying the value of the pixels, but rather apply some modifications of the images based on the known position of the line. We still started with images that have been preprocessed using LAIA’ scripts. We then detected both the text’s baseline and midline. After that, we used the position of both lines in each column of pixels to compute the size of the text’s body, the size of ascendant and descendant letters. Finally, we rescaled each part of the text to a fixed height and placed those parts at a specific height so that they are at the same position between all the images. We decided to give a bigger size to the text’s body as we thought that this part contains more informations and is useful to distinct letters like *a*, *e* or *o*. An example of the modified images is presented in figure 6.

~~text with absolute precision.~~
text with absolute precision.

Fig. 6: The upper image shows the detection of both the midline and the baseline on a text image from the IAM [27] database. The second image is obtained after applying a normalization to this image. The part of the text above the midline has been rescaled to a fixed height. The same modification is applied to the text body (between the midline and the baseline) and to the part below the text baseline.

We then trained the architecture on the modified IAM dataset. Figure 7 shows the results of the trained model and compares it with the results we got on the reference. The results show an increase in the error rate compared to the reference. With an higher error rate, the architecture struggle more to recognize characters and words.

The confidence intervals this time were far enough from each other to say that there was a difference between the results of the two experiments. Using an another *Student t test* led to a *p-value* of 0.0015, which is inferior to 0.05, our error risk of 5%. Hence, we can conclude that the *Null Hypothesis* could be rejected. That means that the results of the two experiments are significantly different, and that the difference is essentially due to others factors than randoms factors such as the weight initialization.

Viewing some of the images of the modified dataset makes us think that the modifications made by this experiment are too harsh. The images contain a lot of noises. One possible origin of such a noise could probably be introduced by the lines themselves. As the baseline is the same in the first experiments, the midline would be the problem. In fact the midline is less stable than the baseline. It is probably due to the fact that there is more black pixels in the bottom of the text’s body than in the upper part, which makes it easier to detect the baseline than the midline.

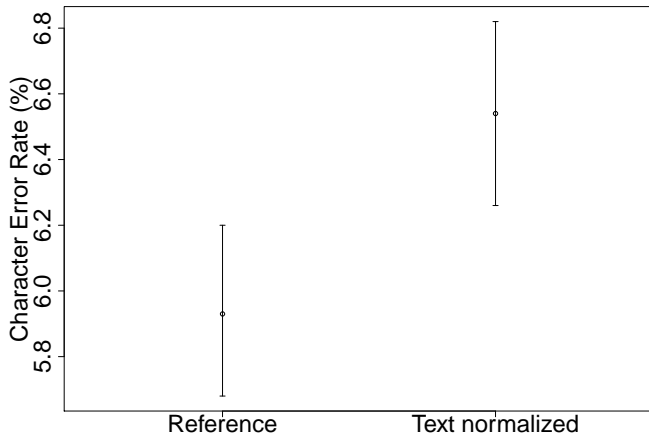


Fig. 7: Comparison of the results for both the reference and the experiment in which we normalize the different parts of the text. We compared the Character Error Rate Confidence Intervals for both the reference and the experiment. A lower value is better.

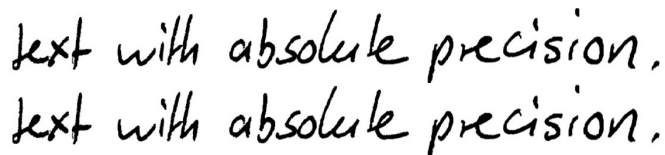


Fig. 8: The first image is an example of a line of text before any of our modification from the IAM [27] database. The second image represent the same example but after shifting the columns of pixels to align the baseline to a fixed value.

4.5 Align the text based on the text baseline position

In response to these problems, we tried another solution that no longer makes use of the midline. We started with the preparation of the dataset. Then we used the text’s baseline in order to align the text. With the position of the line known in each column of the image, we then shifted the columns, without any scaling. The idea was still to straighten the text put it to a fixed height. Figure 8 shows an example of an image just after the application of this process.

Then we trained the network with this dataset. Figure 9 shows a comparison of the confidence intervals for both the reference and the experiment where we shifted the columns.

We observed that the results of our experiment seemed to match closely with the results of the reference. Again, we performed a *Student t test* and it returned a *p-value* of 0.86. This value is far from the critical value of 5%. Therefore, we cannot conclude that there is a significant difference between the two observations. But we can still admit that the tiny difference is due to the weight initialization and that adding the line this way may have no impact.

However, by taking a look at most images of the dataset, it seems that the images remain practically unchanged. Figure 8 shows that well. That fact could be explained by the nature of the IAM dataset which is a very stable one. There is not so much noise in it, the images are already almost aligned (i.e the text baseline is almost straight and horizontal). In order to see a significant impact

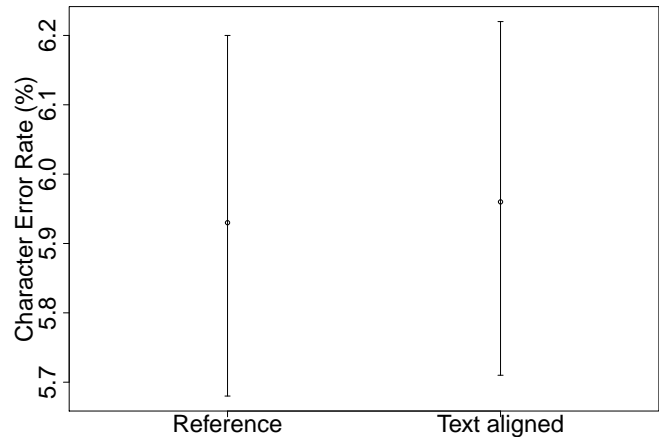


Fig. 9: Comparison of the results for both the reference and the experiment in which we shifted the pixels columns so that the text is aligned. We compared the Character Error Rate Confidence Intervals for both the reference and the experiment. A lower value is better.

	CER (%)	WER (%)
Reference	5.93 [5.68 – 6.20]	18.89 [18.29 – 19.54]
Line on 1 channel	6.21 [5.94 – 6.47]	19.26 [18.65 – 6.47]
Line on 3 channels	5.85 [5.59 – 6.12]	18.71 [18.09 – 19.33]
Normalized images	6.54 [6.26 – 6.82]	20.23 [19.57 – 20.88]
Aligned images	5.96 [5.71 – 6.22]	18.68 [18.07 – 19.33]

TABLE 4: This table shows the results of the reference and all our experiments. CER stands for Character Error Rate and WER for Word Error Rate.

of the baseline we think that we need to try the experiment on another dataset where text images are more noisy and less aligned.

5 CONCLUSION

In this paper, we presented our work on testing if adding the baseline information to the architecture could change its recognition performance. We introduced different methods to add the baseline to the images or directly on the pipeline (in section 3.3). We then presented our different experiments in section 4. We also discussed of the results of all experiments. Table 4 shows the results of the reference and our experiments.

While most of the time we cannot conclude that there is a significant difference between those results, except for the experience where we tried to normalize the images, the results are close to reference results. According to our experiments results, adding the baseline is not necessary as it costs time to get the baseline’s position and add it to the images, while not improving the recognition performances. Moreover, between all the experiments, we did not observe any particular difference in the training time.

However, due to the specific conditions we were under during this project, we could only experiment on the IAM dataset. It is a rather stable one, with well aligned text and a modern writing style. We believe that it could be interesting

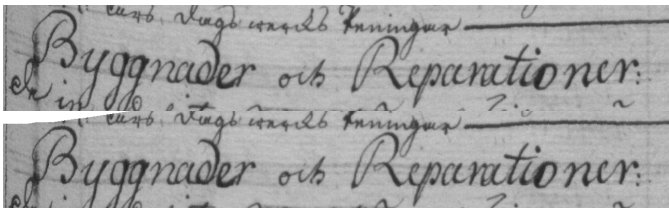


Fig. 10: This figure shows an example where we tested to align the detected baseline with a horizontal pixels line. The upper image is an image with a curvy line, while the bottom one is obtained after aligning the baseline.

to test our methods, especially the text alignment on another dataset with images that are more variable. Figure 10 shows a line from an ancient document where we tested our last experience which consists to align the detected baseline. As the image was really curvy, the resulting image has been far more impacted than the majority of the IAM dataset in our experience (Figure 8).

REFERENCES

- [1] Alex Graves and Juergen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 545–552. Curran Associates, Inc., 2009.
- [2] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.
- [3] B. Moysset, T. Bluche, M. Knibbe, M. F. Benzeghiba, R. Messina, J. Louradour, and C. Kermorvant. The a2ia multi-lingual text recognition system at the second maurdor evaluation. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 297–302, Sept 2014.
- [4] D. Suryani, P. Doetsch, and H. Ney. On the benefits of convolutional neural network combinations in offline handwriting recognition. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 193–198, Oct 2016.
- [5] Yi-Chao Wu, Fei Yin, and Cheng-Lin Liu. Improving handwritten chinese text recognition using neural network language models and convolutional neural network shape models. *Pattern Recognition*, 65:251–264, 2017.
- [6] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, Nov 1997.
- [7] A. Ray, S. Rajeswar, and S. Chaudhury. Text recognition using deep blstm networks. In *2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR)*, pages 1–6, Jan 2015.
- [8] Marcus Liwicki, Alex Graves, Horst Bunke, and Jürgen Schmidhuber. A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks. In *In Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR 2007*, 2007.
- [9] V. Frinken and S. Uchida. Deep blstm neural networks for unconstrained continuous handwritten text recognition. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 911–915, Aug 2015.
- [10] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, May 2009.
- [11] Q. Liu, L. Wang, and Q. Huo. A study on effects of implicit and explicit language model information for dblstm-ctc based handwriting recognition. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 461–465, Aug 2015.
- [12] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Multi-dimensional recurrent neural networks. *CoRR*, abs/0705.2011, 2007.
- [13] 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*, pages 285–290, Sept 2014.
- [14] 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)*, pages 228–233, Oct 2016.
- [15] T. Bluche, J. Louradour, M. Knibbe, B. Moysset, M. F. Benzeghiba, and C. Kermorvant. The a2ia arabic handwritten text recognition system at the open hart2013 evaluation. In *2014 11th IAPR International Workshop on Document Analysis Systems*, pages 161–165, April 2014.
- [16] Joan Puigcerver. Are multidimensional recurrent layers really necessary for handwritten text recognition? 2017.
- [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*, pages 279–284, Sept 2014.
- [18] Morgan Nelson and Boulard Hervé. An introduction to hybrid hmm/connectionist continuous speech recognition.
- [19] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.
- [20] F. Zamora-Martínez, V. Frinken, S. España-Boquera, M.J. Castro-Bleda, A. Fischer, and H. Bunke. Neural network language models for off-line handwriting recognition. *Pattern Recognition*, 47(4):1642–1652, 2014.
- [21] V. Frinken, F. Zamora-Martínez, S. España-Boquera, M. J. Castro-Bleda, A. Fischer, and H. Bunke. Long-short term memory neural networks language modeling for handwriting recognition. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 701–704, Nov 2012.
- [22] Joan Puigcerver, Daniel Martín-Albo, and Mauricio Villegas. Laia: A deep learning toolkit for htr. <https://github.com/jpuigcerver/Laia>, 2016. GitHub repository.
- [23] Alex Graves et al. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.
- [24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [25] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [26] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.
- [27] U-V Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, 2002.