Results of a PyTorch implementation of an Handwritten Text Recognition Framework

Killian Barrere[1], Aurélie Lemaitre[2], and Bertrand Coüasnon[2]

[1] École Normale Supérieure de Rennes, Univ Rennes, France
killian.barrere@ens-rennes.fr
[2] Univ Rennes, CNRS, IRISA, France
aurelie.lemaitre@irisa.fr; bertrand.couasnon@irisa.fr

Abstract.

The task of automatically extracting transcription from images of handwritten text is still motivating a lot of people. In this report, we explain in a comprehensive manner, some points of a Handwritten Text Recognition framework we implemented. As an extension of DeepDIVA, which is built upon PyTorch, it is able to perform well on the IAM dataset and on ICFHR 2018 READ dataset.

1 Introduction

There exists a large number of handwritten documents that awaits to be scanned and transcribed, so that they could be used for a private or public usage. Since manual transcriptions are expansive and time consuming, more and more efforts are made to design systems that are able to extract a correct transcription. This is especially true with large datasets and historical datasets in which commercial Optical Character Recognizer are not performing very well. The term Off-line Handwritten Text Recognition (HTR) designates such systems that are able to extract a transcription from only images of handwritten text (as shown in figure 1). While some systems have been using Hidden Markov Models to perform that task, almost every recent system is now using Deep Learning architectures. They are mainly composed of Recurrent Neural Network and Convolutional Neural Networks.

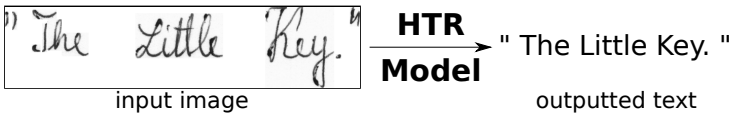


Fig. 1. An example of a transcription from an HTR model on a text line of IAM.

However, only a few frameworks for HTR are currently able to train models and extract text. Moreover, we think that they lack of customization, especially when a user wants to try new architectures.

To that end, we propose to implement an HTR framework and a working architecture along with it, as an extension of DeepDIVA [1]. DeepDIVA is a Deep Learning framework working with PyTorch that aims to bring an easier installation, reproducibility of the experiments and a modular system that allows the users to customize their tasks and architectures.

This report starts by giving an insight into related works in section 2. We then present our framework as well as our Deep Learning architecture that we used in section 3, before showing results obtained on several datasets in section 4. We finish the report by concluding on the work achieved.

2 Related Works

Handwritten Text Recognition Architectures Hidden Markov Models (HMM) [28] used to be the most competitive solution for Handwritten Text Recognition (HTR) 15 years ago. But there are major drawbacks in using HMMs. The users must provide informative, discriminating and independent features to build an effective solution. Features are representation of images attributes. A second problem is the difficulty to have links between what the system is currently seeing and what it has already seen.

Neural networks help to overcome both difficulties. As a result, they have been increasingly used in HTR competitions and are now best performing architectures for this task.

Convolutional Neural Networks (CNN) solve the first problem. They are able to learn how to extract features which are relevant to the task. They replace some of the human works, and improve the performance of HTR models in general. Bluche et al. propose a different version of CNN called Gated CNN [3]. They improve CNNs by adding a possibility to automatically select which features to use based on the area the networks is currently processing. This new method shows promising results by outperforming current state of the art.

Recurrent Neural Networks (RNN) have been used for HTR to increase the ability of the whole system to learn longer dependency. They use a memory and recurrent connections to propagate information through time. However, their ability to memorize useful information is not enough for HTR. It leads to the use of Long Short Term Memory (LSTM) introduced by Hochreiter et al. [14] and improved with forget gates [7]. LSTM has the ability to learn when to take into account the input, when to keep it in memory, when to forget it and when to output the state of its memory. They are used in every recent HTR system to learn long-range context.

In the case of an image of a text line, the image could be seen as unidimensional with RNNs scanning from left to right (for European languages). However, each point has only information from the past. Bidirectional RNN (BRNN) [26], is composed of two RNNs, one scans left to right and the other reversely. In that case each point potentially has information from both past and future context. BLSTMs [11] are used for speech recognition [8] [32] and later for HTR with a model named LSTM-RNN and show improvements in the recognition scores [16] [6] [24]. They are also coupled with CNNs to create CRNN architectures [27] [30] [23] [3].

It is also possible to take the input images as matrices of pixels instead of vectors (or line) of columns. In that case, the images could be seen as bi-dimensional. For that purpose, Graves proposed Multidimensional RNN (MDRNN) [12] [13] [9] and its variant MDLSTM which are the extension of RNN and LSTM for multiple dimensions. Following the same technique used for regular RNN, we can extend the known context to every other direction with Multi-directional MDRNN. They combine four RNNs, one scanning from top left to bottom right, another from top right to bottom left, etc. The multiple directions are always used for HTR, and we will use the term MDRNN to designate Multi-directional MDRNN. MDRNNs tend to give better results for the task of HTR than BRNNs and their equivalents BLSTMs. They are widely used for HTR alone [2] or more often combined with convolutions [18] [20] [31]. However, they are far slower to train than their 1D version. Depending on the available resources, both 1D and 2D RNN are relevant.

Graves et al. introduce Connectionist Temporal Classification (CTC) [10], a custom layer that is able to translate the output of the network into text labels. It has the advantage of allowing end to end models, which are models able to take raw images as input and outputting text. This step is called decoding. Furthermore, with the abilities of CTC, pre segmented data are no more required as CTC is able to map outputted labels with the positions in the ground truth. It saves a lot of time by adding the possibility when creating the data to no more give the position of every character and words.

To further improve the result of a model, Language Models (LM) are used. LMs are able to learn constraints on the vocabulary and then correct the network on these errors. There exist many different architectures of LM with their specificities. However, they are not always so useful. Furthermore, because LMs work independently of the Neural Networks architectures, some systems do not include them in their pipeline.

Available Frameworks LAIA [23] is a deep learning framework especially built for the task of HTR and tasks that require an HTR model. It provides useful functions to train a model and get the transcription of a set of images. Moreover LAIA has entire pipelines starting from download and preparation of the images to applying an LM for many existing datasets. Both CRNN and MDLSTM models are implemented and in our opinion it is a good starting point

when getting results quickly is crucial. However, it is constructed on top of Torch, which make the toolkit no longer updated.

RETURNN [5] [33] is a Deep Learning framework mainly built to optimize at the GPU level RNN, LSTM, and their multidimensional variants. It takes advantage of the use of Theano, Tensorflow and CUDA fast implementations. Models can run on both CPUs and GPUs, and RETURNN already comes with a good number of examples.

DeepDIVA [1] is a Deep Learning framework using both Python and PyTorch that aims to ease the reproducibility of the experiments. It helps by saving time during the installation process (by installing automatically CUDA for instance), having great compatibility, the reproduction of experiments or by providing useful visualizations during the experimentation process. More details can be found in the original paper [1]. However, at the moment of writing, the tasks already implemented inside the framework does not include HTR.

Our point of view In our mind, it could get difficult to customize the task and the architectures with the few existing frameworks. Also because of the numerous advantages of DeepDIVA [1], we decided to build our own implementation of the HTR task and associated architecture as an extension of DeepDIVA.

We retain two architectures: CRNNs [27] [30] [23] and MDLSTMs [18] [20] [31]. Both models have good results, with MDLSTMs slightly better. We choose to implement a CRNN architecture compared to MDLSTMs because CRNN architectures are faster to train and also easier to implement. Our model does not include an LM, because we want to focus on other parts and because it could always be added later.

3 Addition of the Handwritten Text Recognition task to DeepDIVA

We choose to use DeepDIVA [1] in the objective of implementing a pipeline to complete the task of HTR. By adding that implementation, we hope that it will ease future research in HTR.

In section 3.1 we will first describe how images of text are loaded and stored along with their transcription. Then section 3.2 will present our implementation of a custom CRNN model. We will discuss about how the CTC loss is included in the whole process in section 3.3 and how we get transcriptions. The whole section aims to explain some points while being the most comprehensive as possible.

Meanwhile, the other important points of the whole toolkit (such as backpropagation, customizations, etc.) are already provided by DeepDIVA and were left mostly unchanged.

3.1 Data Loading

HTR datasets often comes with their specific structure and representation. To avoid writing a specific function for each dataset, we choose to use the Pivot File Format (PiFF) [19] to represents datasets in a common format. We then use a single function to load images and their transcriptions from the PiFF file.

As HTR datasets contain a great number of high-resolution images and we do not want to lose any useful information by reducing the size of the images, we have no choice but to load dynamically images.

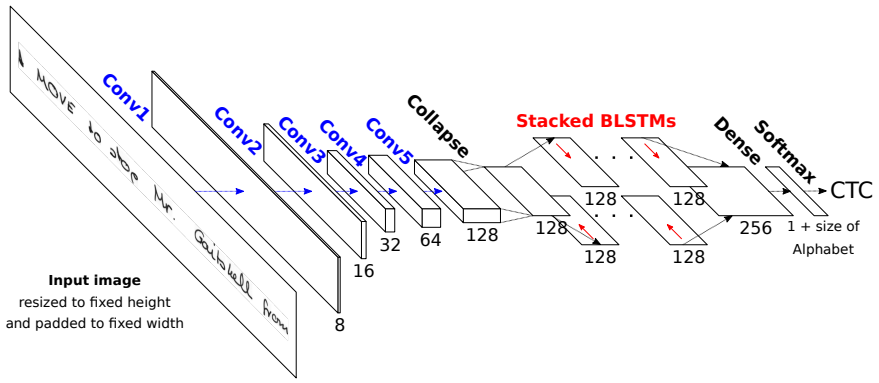
Images of HTR datasets usually have different sizes. However, it is always useful to work with batches of images to have a speedup. To that end, we resized all images to a fixed size. As we are using RNNs that are working on the horizontal dimension, our architecture is able to handle horizontal variations. Images are first rescaled to a fixed height, and then padded with black pixels, which is not a problem for Bidirectional RNNs.

3.2 CRNN Architecture

In this section, we explain our implementation of a CRNN architecture (first presented in section 2). Our architecture (a visualisation is available in figure 2) is largely inspired by existing CRNN architectures [27] [23] [3]. It is first composed of a series of convolutional layers that aims to extract more and more features as the image is processed through the different convolutional layers, followed by stacked BLSTMs that are in charge to generate characters probabilities.

The model takes as input images of three channels (RGB), but the architecture could easily be changed so that it takes a different number of channels (such as one channel for grayscale images). Each of the first three convolutions (Conv1, Conv2 and Conv3 in figure 2) are followed by a 2x2 max pooling to reduce the size of images and features, by covering a larger size on the input image. The last 3 convolutions (Conv3, Conv4 and Conv5 in figure 2) use dropout. Dropout allows the network to avoid overfitting by randomly selecting neurons that will not be taken into account during the forward and backward pass of the training phase. Every convolution is directly followed by a LeakyReLU activation function and by a Batch Normalization as in LAIA [23]. Normalizing the batch features after each convolution has a similar positive effect as when normalizing the input images at the beginning of the architecture. The CNNs are in charge of producing 2D feature maps that are later converted to a sequence of features in the Collapse layer by taking the maximum value for each column and feature.

Then a fixed number of stacked BLSTMs are in charge of producing values that will predict a character. In the HTR experiments, we tried from two to five stacked BLSTMs. We expect better results with a deeper network, but at a cost



Conv1	Kernel: 3x3; Features: 8; MaxPooling: 2x2; Batch Normalization
Conv2	Kernel: 3x3; Features: 16; MaxPooling: 2x2; Batch Normalization
Conv3	Kernel: 3x3; Features: 32; Dropout: 0.2; MaxPooling: 2x2; Batch Normalization
Conv4	Kernel: 3x3; Features: 64; Dropout: 0.2; Batch Normalization
Conv5	Kernel: 4x2; Features: 128; Dropout: 0.2; Batch Normalization
Collapse	MaxPooling on the height of each column
BLSTMs	Multiple Stacked BLSTMs; 128 Features for each direction; Dropout: 0.5
Dense	Fully Connected Layers; Input: 256; Output: Number of characters + 1
Softmax	Transforming outputs in Character probabilities.
CTC	Convert Character probabilities in a text and provide a loss function.

Fig. 2. Our CRNN architecture. More details can be found in section 3.2

of an increase in the time required to converge. The LSTMs have a hidden size of 128 and a dropout value of 0.5.

A dense layer then takes the outputs of the BLSTMs column by column to produce a value per character of the alphabet and the blank label.

3.3 Connectionist Temporal Classification

Finally, a Softmax is applied to obtain character probabilities for each block of the processed image. These probabilities are then given to the CTC to compute the loss. As it is mostly the case in all Python implementation, we choose to use warp-ctc [25] and its binding with PyTorch [21] because of its speed to compute the loss. It computes the loss in a parallel and quick fashion using a negative likelihood.

In parallel, we use character probabilities to generate the predicted sequences. This step is called decoding. As warp-ctc is only in charge of computing the loss, we implemented and tested two algorithms.

The first one is called Best Path Decoding and introduced by Graves et al. [10]. In this algorithm, we simply construct the predicted sequence frame by frame, by taking the most probable label at each step. We then apply a post-

processing to this sequence, by removing the repetition on consecutive frames. The blank labels are removed from this sequence to obtain the final result. This method has the advantage of being very quick to compute and therefore produces satisfying results without spending a lot of time. For these reasons it is widely used in HTR nowadays. But this is a greedy algorithm and only produces an approximation of the best decoding.

We also implemented Beam Search Decoding (which is a more precise heuristic, that consists to explore the best parts of the tree of possible texts at each step). However, some tests show that in comparison to Best Path Decoding, it is taking a very long time and does not bring any differences on the text produced.

To measure the performance of one model, the most two common metrics are Character Error Rate (CER) and Word Error Rate (WER). For CER (resp. WER), that score corresponds to the edit distance, which is the minimal number of additions, deletions, or substitutions of characters (resp. words) between the prediction and the ground truth, normalized by the number of characters (resp. words) in the ground truth. We implemented and tested both CER and WER computation. These scores can be understanded as the percentage of wrong characters or words.

4 Experiments

In this section, we will describe the different experiments we tried and their results. We train on text line images as showed in figure 1. In every experiment, we use a batch size of 32, a learning rate of 0.001,Stochastic Gradient Descent as the optimizer and an Nvidia Tesla P100 with 16 GB of memory. However, like many Deep Learning experiments we manage to do only one run per experiment. The main reasons for that choice are that training takes a long time (For instance each experience on the IAM dataset in section 4.1 takes 2 days.) , a lack of available time to complete everything and the high cost of GPUs altogether. Therefore, results could be due to some random factors and are not representative. DeepDIVA allows the user to choose a fixed seed, however when using GPUs and CUDA kernels, at the moment of writing and to the best of our knowledge, the results are random and cannot be seeded. As training of an HTR architecture requires a lot of computing powers, GPUs are almost the only possibility. Therefore, we cannot have a perfect reproduction of an experiment that is one of the main objectives of DeepDIVA.

4.1 Experiment on the IAM Dataset

IAM [17] is an HTR dataset. The text is written in English by several writers that have been asked to handwrite a specific text. The text is available in entire forms, lines or single words images.

We choose that dataset to build, optimize the hyper-parameters and test our architecture. It is composed of a large number of images of text lines (6181, 966 and 3030 images for the training, validation and test set on the Aachen partition) and because it is also widely used to test models, it allows us to compare our results with others.

A first step concerns the preparations of the images of the dataset. As it is already well covered by others, we decided to reuse an existing pre-processing of the images. The authors of LAIA [23] used a de-skewing to remove misalignment, replacing white borders with a border of a fixed pixel size and also resizing the image to a fixed size. Meanwhile, Kozielski et al. proposed to do a different preparation of the images [15]. They start by normalizing the contrast of the images, then apply a correction in the slant of the images. They use a system applying modifications on the repartition of the black pixels, in order to get a fixed mean and standard deviation, frame by frame on the input images. We choose to reuse the pipeline from LAIA as it is simpler, easy to install and also tested for a similar Deep Learning architecture. Moreover, the different scripts from the pipeline of LAIA [23] also come with a preparation of the ground truth. It removes some inconsistency contained in the text, so that they match what is really written on the images. We decided to also use it as it better match the text images and the text predicted by a model, however, to have fair comparisons, we would need to evaluate each model with that same transcription. Lastly, we used the Aachen custom partition of the dataset to train and evaluate models as it is widely used.

First experiments aim to build a good CRNN Architecture. We start by optimizing the Convolutions. In our opinion, the number of max pooling is what makes big differences. Those layers divide the size of the image by 2 on each axis. In a direct conclusion, it reduces the time taken to compute forward and backward pass. It also allows features of the next layers to have a summary of information of a greater area of the input image, and potentially leading to better results. However, using too many max pooling could resume too many values in too few and could make the entire network unable to learn anything.

Number of	Validation		Test		Training time
Max Pooling	CER	WER	CER	WER	(in hours)
2	7.54	26.33	10.59	33.33	23
3	**7.05**	**24.89**	**10.34**	**32.35**	36
4	12.12	36.65	16.00	44.29	24

Table 1. Influence of the number of Max Pooling layers in our architecture on the Error Rates and on the training Time. A number of N means that the first N convolutional layers are followed by a 2x2 max pooling. For the rest of the network, we used 2 stacked BLSTMs. We ran the experiment on the IAM dataset.

Table 1 shows the results we obtained with a different number of CNN layers using 2x2 max pooling. With two max pooling, the training seems to be a bit unstable and converges after 23h. By using three max pooling, we observe a convergence after 36h that leads to better results. Adding a fourth max pooling is not increasing the performance, however. We choose to use three max pooling as it gives the best error rates.

We optimize the number of stacked BLSTMs to get the best possible performance. We have been training CRNN models from two to five consecutive BLSTMs and compare the training time and their error rates in table 2.

Number of Stacked BLSTMs	Validation CER	Validation WER	Test CER	Test WER	Training Time (in hours)
2	7.05	24.89	10.34	32.35	36
3	5.75	20.07	8.54	27.00	41
4	**5.41**	**19.07**	**8.17**	**25.59**	**26**
5	6.09	21.78	9.09	28.65	55

Table 2. Influence of the number of stacked BLSTMs in our architecture, on the Error Rates and on the training Time. For the rest of the network, we used max pooling on the first three layers of convolutions. We ran the experiment on the IAM dataset.

Results show that the more BLSTMs we are using, the more time it takes to complete one epoch. But error rates are lowering as we increase that number. However, by adding a fifth BLSTM layer we observe a short increase in the error rates. It could be explained by the fact that the network needs more and more time and epochs to get good results, or by the fact that the rest of the network is no longer well suited for as many BLSTMs layers. In any case, we decided to use a model with three max pooling and four stacked BLSTMs as our best model and to compare with others.

Table 3 shows the results obtained with our model as compared with other works. The error rates obtained by our model are higher than the best performing models. However, the scores are not that far and mainly demonstrate that the implementation of an HTR system is working as expected. We believe that to get lower error rates we would need to spend more time on optimizing the remaining hyper parameters of the network. We did simple experiments to pick a good learning rate, we did not test the influence of the number of features in each Convolutional layer and did not test how the number of hidden units in BLSTM can affect the results. We could also try a better pre-processing of the image as it impact a lot the results of one model [15]. In addition, working on an LM would bring better results as table 3 shows that well. That could be achieved with additional time, but we also wanted to test different architectures and other datasets.

Model Name	Model Description	Validation CER	Validation WER	Test CER	Test WER
Our Model	CRNN	5.4	19.1	8.2	25.6
LAIA [23]	CRNN	3.8	13.5	5.8	18.4
LAIA [23]	CRNN + LM	2.9	9.2	4.4	12.2
Kozielski et al. [15]	LSTM-RNN	5.5	16.6		
Kozielski et al. [15]	LSTM-RNN + LM	2.7	9.5	5.1	13.3
Doetsch et al. [4]	LSTM-RNN + LM	2.5	8.4	4.7	12.2
Pham et al. [22]	MDLSTM	7.4	27.3	6.3	43.9
Pham et al. [22]	MDLSTM + LM	3.7	11.2	5.1	13.6
Voigtlaender et al. [31]	MDLSTM + LM	**2.4**	**7.1**	**3.5**	**9.3**

Table 3. Comparison between the results of our model and the state of the art on the IAM dataset. Here different models are compared, and some are a lot more complex. While CRNN and LSTM-RNN requires a similar time to train a model, MDLSTM are far longer to train.

4.2 ICFHR2018 Competition on Automated Text Recognition on a READ Dataset

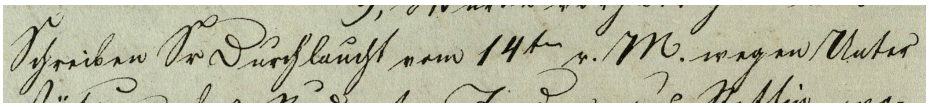


Fig. 3. An example from the READ retraining dataset.

The objective of the Competition on Automated Text Recognition on a READ Dataset [29], introduced at the ICFHR 2018, is to evaluate how much labelled data are needed to obtain low error rates when applying transfer learning. The provided dataset contains multiple documents. Each document contains several pages and is written by a different writer in a specific language. Pages are composed of several text lines that are used as models inputs.

The basic idea is to start by training a model on a wide group of datasets. The training data are composed of pages from 16 different documents. Training data is large enough (11925 text line images), with a fair diversity Using additional data is forbidden, however, augmentation techniques can be used to artificially increase the size of the dataset and the generalization ability of the model.

Then, after training a model on these data, the objective is to test how much labelled data is required to obtain good performance. For that task, the organizers provided five additional documents which are not present in the training data. Each document is divided in a retraining set and a testing set, with a null intersection. While the retraining set provide all transcription, the testing set

does not include any. Retraining set also comes with a list of subsets, composed of a different number of pages. Subsets is either composed of 0 (in that case, it is only needed to test the model trained on the large and general training data), 1, 4 or 16 pages of the document. The objective is then to retrain a model for each subset and then to evaluate the retrained model on the test data of that same document. When retraining a model (with e.g. 4 pages), we simulate the case where the other labels (e.g. those from the 16 pages that are not in the 4 pages), are not known and restrict the data to only those from the current subset.

The competition is already completed, but it is still possible to submit the results of the different retrained model to be evaluated like other participants in the competition. While the best solution would have been to optimize the different hyper parameters of the architecture, process cross-validation on the smaller dataset when retraining, we choose to use our best performing model on IAM (see section 4.1) because of time constraints. Additionally, in the objective of avoiding unfortunate bad validation dataset, we divided the retraining subset in a training set and a validation set, but with the validation set bigger than usual. 70% of the image of lines were used for the training set and the remaining 30% for the validation set, while only 10% of the general training data where put in the validation set.

Model	CER
Assiut University	**13.02**
Ohio State University	17.86
LITIS	18.81
UOB and ParisTech	19.24
Universitat Politècnica de València	21.54
Our model	21.65
University of Pernambuco	27.33

Table 4. Results of the ICFHR2018 Competition on Automated Text Recognition on a READ Dataset taken from `https://scriptnet.iit.demokritos.gr/competitions/10/viewresults/` The CER corresponds to the global CER on every test image.

Results showed in table 4 demonstrate that the model we trained with our implementation and our architecture performs as well as the other models. With more time, we believe that our model could have been optimized and obtain a better ranking. This experiment mostly shows that our implementation works as expected and are relevant to the task.

5 Conclusion

In this report, we presented our implementation of a Handwritten Text Recognition Framework and one custom Deep Learning architecture following the

principles of a CRNN architecture. We tested it on the IAM and the ICFHR 2018 READ dataset, and compared our results with others. Despite the fact that we do not manage to match the best results, we still obtain fair results proving that our implementation does what it is intended to do. As an extension of DeepDIVA, the framework has the good properties of achieving good performance, visualisations of the results while training, and a good modularity. However, because of the use of CUDA kernels, the exact reproduction of experiments is limited.

Nevertheless, there are still works to be done mostly regarding the implementation of other architectures such as MDLSTM. Language Models could also be added as part of the framework, and further experiments could be run to further improve the results of the CRNN architecture.

References

1. Alberti, M., Pondenkandath, V., Würsch, M., Ingold, R., Liwicki, M.: DeepDIVA: A Highly-Functional Python Framework for Reproducible Experiments (apr 2018)
2. Bluche, T., Louradour, J., Knibbe, M., Moysset, B., Benzeghiba, M.F., Kermorvant, C.: The a2ia arabic handwritten text recognition system at the open hart2013 evaluation. In: 2014 11th IAPR International Workshop on Document Analysis Systems. pp. 161–165 (April 2014). https://doi.org/10.1109/DAS.2014.40
3. Bluche, T., Messina, R.: Gated convolutional recurrent neural networks for multilingual handwriting recognition. In: Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on. vol. 1, pp. 646–651. IEEE (2017)
4. Doetsch, P., Kozielski, M., Ney, H.: Fast and robust training of recurrent neural networks for offline handwriting recognition. In: 2014 14th International Conference on Frontiers in Handwriting Recognition. pp. 279–284 (Sept 2014). https://doi.org/10.1109/ICFHR.2014.54
5. Doetsch, P., Zeyer, A., Voigtlaender, P., Kulikov, I., Schlüter, R., Ney, H.: Returnn: The rwth extensible training framework for universal recurrent neural networks. In: Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on. pp. 5345–5349. IEEE (2017)
6. Frinken, V., Uchida, S.: Deep blstm neural networks for unconstrained continuous handwritten text recognition. In: 2015 13th International Conference on Document Analysis and Recognition (ICDAR). pp. 911–915 (Aug 2015). https://doi.org/10.1109/ICDAR.2015.7333894
7. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: Continual prediction with lstm (1999)
8. Graves, A., Jaitly, N., r. Mohamed, A.: Hybrid speech recognition with deep bidirectional lstm. In: 2013 IEEE Workshop on Automatic Speech Recognition and Understanding. pp. 273–278 (Dec 2013). https://doi.org/10.1109/ASRU.2013.6707742
9. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence **31**(5), 855–868 (May 2009). https://doi.org/10.1109/TPAMI.2008.137
10. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In:

Proceedings of the 23rd international conference on Machine learning. pp. 369–376. ACM (2006)
11. Graves, A., Fernández, S., Schmidhuber, J.: Bidirectional lstm networks for improved phoneme classification and recognition. In: Proceedings of the 15th International Conference on Artificial Neural Networks: Formal Models and Their Applications - Volume Part II. pp. 799–804. ICANN'05, Springer-Verlag, Berlin, Heidelberg (2005), http://dl.acm.org/citation.cfm?id=1986079.1986220
12. Graves, A., Fernández, S., Schmidhuber, J.: Multi-dimensional recurrent neural networks. CoRR **abs/0705.2011** (2007), http://arxiv.org/abs/0705.2011
13. Graves, A., Schmidhuber, J.: Offline handwriting recognition with multidimensional recurrent neural networks. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) Advances in neural information processing systems 21. pp. 545–552. Curran Associates, Inc. (2009), http://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks.pdf
14. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (Nov 1997). https://doi.org/10.1162/neco.1997.9.8.1735, http://dx.doi.org/10.1162/neco.1997.9.8.1735
15. Kozielski, M., Doetsch, P., Ney, H.: Improvements in rwth's system for off-line handwriting recognition. In: Document Analysis and Recognition (ICDAR), 2013 12th International Conference on. pp. 935–939. IEEE (2013)
16. Liwicki, M., Graves, A., Bunke, H., Schmidhuber, J.: 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)
17. Marti, U.V., Bunke, H.: The iam-database: an english sentence database for offline handwriting recognition. International Journal on Document Analysis and Recognition **5**(1), 39–46 (2002)
18. Messina, R., Louradour, J.: Segmentation-free handwritten chinese text recognition with lstm-rnn. In: 2015 13th International Conference on Document Analysis and Recognition (ICDAR). pp. 171–175 (Aug 2015). https://doi.org/10.1109/ICDAR.2015.7333746
19. Mouchère, H., Kermorvant, C., Rojas, A., Coustaty, M., Chazalon, J., Coüasnon, B.: Piff: a pivot file format **1** (2017)
20. Moysset, B., Bluche, T., Knibbe, M., Benzeghiba, M.F., Messina, R., Louradour, J., Kermorvant, C.: The a2ia multi-lingual text recognition system at the second maurdor evaluation. In: 2014 14th International Conference on Frontiers in Handwriting Recognition. pp. 297–302 (Sept 2014). https://doi.org/10.1109/ICFHR.2014.57
21. Naren, S.: Pytorch bindings for warp-ctc. https://github.com/SeanNaren/warp-ctc
22. Pham, V., Bluche, T., Kermorvant, C., Louradour, J.: Dropout improves recurrent neural networks for handwriting recognition. In: Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on. pp. 285–290. IEEE (2014)
23. Puigcerver, J., Martin-Albo, D., Villegas, M.: Laia: A deep learning toolkit for htr. https://github.com/jpuigcerver/Laia (2016), gitHub repository
24. Ray, A., Rajeswar, S., Chaudhury, S.: Text recognition using deep blstm networks. In: 2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR). pp. 1–6 (Jan 2015). https://doi.org/10.1109/ICAPR.2015.7050699
25. Research, B.: warp-ctc: A fast parallel implementation of ctc, on both cpu and gpu. https://github.com/baidu-research/warp-ctc

26. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing **45**(11), 2673–2681 (Nov 1997). https://doi.org/10.1109/78.650093
27. Shi, B., Bai, X., Yao, C.: 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 **39**(11), 2298–2304 (Nov 2017). https://doi.org/10.1109/TPAMI.2016.2646371
28. Stratonovich, R.L.: Conditional markov processes. Theory of Probability & Its Applications **5**(2), 156–178 (1960)
29. Strauss, T., Leifert, G., Labahn, R., Hodel, T., Mühlberger, G.: Dataset for ICFHR2018 Competition on Automated Text Recognition on a READ Dataset (Oct 2018)
30. Suryani, D., Doetsch, P., Ney, H.: On the benefits of convolutional neural network combinations in offline handwriting recognition. In: 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR). pp. 193–198. IEEE (Oct 2016). https://doi.org/10.1109/ICFHR.2016.0046
31. Voigtlaender, P., Doetsch, P., Ney, H.: Handwriting recognition with large multidimensional long short-term memory recurrent neural networks. In: 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR). pp. 228–233. IEEE (Oct 2016). https://doi.org/10.1109/ICFHR.2016.0052
32. Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., Stolcke, A.: The Microsoft 2017 Conversational Speech Recognition System. ArXiv e-prints (Aug 2017)
33. Zeyer, A., Alkhouli, T., Ney, H.: Returnn as a generic flexible neural toolkit with application to translation and speech recognition. arXiv preprint arXiv:1805.05225 (2018)