Christoph Wick, Christian Reul, Frank Puppe

Comparison of OCR Accuracy on Early Printed Books using the Open Source Engines Calamari and OCRopus

Abstract

This paper proposes a combination of a convolutional and an LSTM network to improve the accuracy of OCR on early printed books. While the default approach of line based OCR is to use a single LSTM layer as provided by the well-established OCR software OCRopus (OCRopy), we utilize a CNN- and Pooling-Layer combination in advance of an LSTM layer as implemented by the novel OCR software Calamari. Since historical prints often require book specific models trained on manually labeled ground truth (GT) the goal is to maximize the recognition accuracy of a trained model while keeping the needed manual effort to a minimum.

We show, that the deep model significantly outperforms the shallow LSTM network when using both many and only a few training examples, although the deep network has a higher amount of trainable parameters. Hereby, the error rate is reduced by a factor of up to 55%, yielding character error rates (CER) of 1% and below for 1,000 lines of training. To further improve the results, we apply a confidence voting mechanism to achieve CERs below 0.5%. A simple data augmentation scheme and the usage of pretrained models reduces the CER further by up to 62% if only few training data is available. Thus, we require only 100 lines of GT to reach an average CER of 1.2%. The runtime of the deep model for training and prediction of a book behaves very similar to a shallow network when trained on a CPU. However, the usage of a GPU, as supported by Calamari, reduces the prediction time by a factor of at least four and the training time by more than six.

1 Introduction

The best OCR engines on early printed books like Tesseract (4.0 beta)[1] or OCRopus[2] currently use Long Short Term Memory (LSTM) based models, which are a special kind of recurrent neural networks. In order to achieve low CERs below e.g. 1% or 2% on early printed books these models must be trained individually for a specific book due to a high variability among different typefaces used (see Springmann et al. 2016 or Springmann and Lüdeling 2017). Thereto, a certain amount of GT, in the case of OCRopus that is a pair of text line image and transcribed text, must be manually labeled. The primary goal is to reduce the number of labeled text lines to achieve a certain error rate. A

[1] https://github.com/tesseract-ocr
[2] https://github.com/tmbdev/ocropy

secondary goal is to continuously retrain the model, if more GT becomes available because e.g. all lines of a book are reviewed and corrected to achieve a final error rate of near 0%. The default OCRopus implementation uses a shallow one layer bidirectional LSTM network combined with the CTC-Loss to predict a text sequence from the line image. Since convolutional neural networks (CNN) showed an outstanding performance on many image processing tasks, see e.g. Mane and Kulkarni (2017), our aim is to train a mixed CNN-LSTM network to increase the overall performance of OCR. Therefore, we compare the default OCRopus implementation with the deep network architectures provided by the novel OCR engine Calamari[3] which is based on TensorFlow[4]. It is well known that voting the outputs of several different models improves the accuracy by a significant margin, which is why we use Calamari's cross fold training approach proposed by Reul et al. (2018). This approach trains five different models whose outputs are combined by a voting mechanism that considers the confidence values of each output. Moreover, Calamari offers data augmentation and pretrained models to increase the accuracy especially on small datasets.

The rest of the paper is structured as follows: Section 2 introduces and discusses related work regarding OCR on early printed books including deep models and voting. The used data and the applied methods are described in detail in Section 3. In Section 4, we evaluate and discuss the results achieved on three early printed books. After summing up the results and insights in Section 5 we conclude the paper with some ideas regarding future work.

2 Related Work

This section lists related work concerning the application of CNN-LSTM hybrids in the areas of speech, vision, and text processing. Furthermore, related work covering improvements on OCR using different voting algorithms are itemized.

2.1 Combinations of CNN-LSTM

Currently CNN-LSTM hybrids are used in a high diversity of fields to achieve state-of-the-art results. The combination of those diverse network structures is promising because CNNs are suited for hierarchical but location invariant tasks, whereas LSTMs are perfect at modelling temporal sequences.

For example, in the domain of speech processing Sainath et al. (2015) use a combination of LSTMs, CNNs and Fully Connected Layer for an automatic speech recognition task, or else Trigeorgis et al. (2016) train a CNN-LSTM for speech emotion recognition. Another excellently suited area for CNN-LSTM networks is video processing, since CNNs are perfect for handling images and the video itself is a sequence of images. For instance, in this area Donahue et al. (2015) propose a CNN-LSTM as basic structure

[3] https://github.com/Calamari-OCR/calamari
[4] https://www.tensorflow.org/

to automatically generate video descriptions or Fan et al. (2016) use this structure for recognizing emotions in videos.

In the field of text recognition, a combination of CNNs and LSTM was most recently proposed by Breuel (2017). The deep models yield superior results on the University of Washington Database III[5], which consists of modern English prints with more than 95,000 text lines for training. However, the effect of deep networks on historical books and using only a few dozens of training lines for training book individual models has not been considered, yet.

A very similar task to OCR is handwriting recognition, e.g. by Graves and Schmidhuber (2009) or Bluche (2015) or scene text recognition, e.g. by Shi et al. (2017). These applications usually act on contemporary data, which is why it is meaningful to include a language model or a dictionary to achieve a higher accuracy. However, for early printed books, e.g. medieval books, general language models are less effective mostly due to variability in spelling.

2.2 Voting

Using voting methods to improve the OCR results of different models was investigated by many different researchers, an overview is given by Handley (1998). Here, we list only a subset of the most important and the most recent work in the field of OCR and focus mostly on historical documents.

Rice et al. (1996) showed that voting the outputs of a variety of commercial OCR engines reduces the CER from values between 9.90% and 1.17% to 0.85%. The voting algorithm first aligns the outputs by using a Longest Common Substring algorithm (Rice et al., 1994) and afterwards applies a majority voting to determine the best character including a heuristic to break ties.

Al Azawi et al. (2015) trained an LSTM network as voter of the aligned output of two different OCR engines. A comparison using printings with German Fraktur and the University of Washington Database III the LSTM approach led to CERs around 0.40%, while the ISRI voting tool achieved CERs around 2%. A major reason for this high improvement is that the LSTM-based voting algorithm learns some sort of dictionary. Thus, the voter was able to predict a correct result even in cases where each individual voter failed. However, this behaviour might not be desired since the method not only corrects OCR errors but also normalizes historical spellings.

Most recently, in Reul et al. (2018) we showed that a cross-fold training procedure with subsequent confidence voting reduces the CER on several early printed books by a high amount of up to and over 50%. This voting procedure not only takes the actual predictions of a single voter into account, but also their confidence about each individual character. Therefore, instead of a majority voting for the top-1 output class the best character is chosen for the top-N outputs. This led to improvements by another 5% to 10% compared to the standard ISRI sequence voting approach.

[5] http://isis-data.science.uva.nl/events/dlia/datasets/uwash3.html

2.3 Data Augmentation and Pretraining

A crucial point for the performance of DNNs is the availability of huge labeled datasets. However, if only a few data points are available or if the GT has to be labeled manually the GT can be augmented to generate new examples. The applied operations must be label preserving, e.g. a line image must still show the same content after augmentation. The augmentations used in Calamari are provided by OCRodeg[6] and consist of padding, distortions, blobs, and multiscale noise.

Furthermore, in Reul et al. (2017b), we successfully applied transfer learning on early printed books using OCRopy: Instead of training a network from scratch with random weights, the weights can be copied from a network that was trained on different GT. Thus, the new network starts its training already with knowledge about the task, i.e. the transferred features are meaningful on the new data as-well. The network only has to adapt for the new typeface and possibly some new characters in the codec. Therefore, in general, the network requires less lines of GT to reach the CER of a network trained from scratch.

3 Material and Methods

The OCR pipeline of Calamari is based on the OCRopus workflow, whose fundamental idea is to use a full text line as input and train an LSTM network to predict the GT character sequence of that line. This is achieved by the usage of the CTC-Loss during training and a CTC-Decoder for prediction. For a deeper insight in this pipeline, in this section, we first introduce the used datasets. Afterwards, we explain our training and evaluation procedure. Finally, the differences of OCRopy and Calamari concerning implementation, hyperparameters, and network architectures are listed.

3.1 Datasets

For our experiments we employ three different early printed books (see Table 1). Only lines from running text are used, whereas headings, marginalia, page numbers etc. are excluded, because these elements vary in line length, font size or their characters e.g. numbers are underrepresented. Thus, the actual data is not affected by unwanted side effects resulting from these elements. 1505 represents an exception to that rule as we chose the extensive commentary lines instead, as they presented a bigger challenge due to very small inter character distances and a higher degree of degradation. Figure 1 shows one line per book as an example.

1505 is an edition of the *Ship of Fools* (*Narrenschiff* by Sebastian Brant) and was digitized as part of an effort to support the *Narragonien digital* project at the University of Würzburg[7]. 1488 was gathered during a case study of highly automated layout

[6] `https://github.com/NVlabs/ocrodeg`
[7] `http://kallimachos.de/kallimachos/index.php/Narragonien`

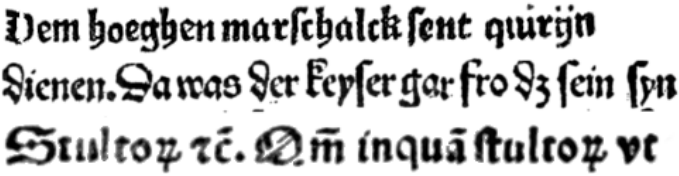

Figure 1: An example line from each of the used books. From top to bottom: 1476, 1488, 1505.

Table 1: Books used for evaluation and their respective number of ground truth lines available for training and validation.

Year	Language	GT Train	GT Validation
1476	German	2,000	1,000
1488	German	3,178	1,000
1505	Latin	2,289	1,000

analysis Reul et al. (2017a) and 1476 is part of the Early New High German Reference Corpus[8]. The GT data of all three books was published[9] by Springmann et al. (2018).

3.2 Training and Evaluation

To train the essential book specific models the human effort to annotate GT should be minimized. Therefore, we examine the effect of adding only a few dozen lines of GT on the CER of an individual book. The aim of a real world application is to train incrementally improved models when more GT becomes available to support the human annotator with increasingly better models.

As setup, each book in the dataset is first split into an evaluation and a training set. While the evaluation set is fixed, the training set size is chosen as 60, 100, 150, 250, 500, and 1,000 lines, with each set subsuming the smaller sets entirely. Then, each training set is divided into a 5-fold, where four parts are used for the actual training and one part for validation. For example, if 100 lines are chosen randomly from the full set, each fold uses 80 lines for training and the remaining 20 lines for validation. These 20 validation lines are distinct from each of the five folds resulting in five diverse models. Based on the validation set the best performing model for each fold is determined.

In summary, for three books and six different numbers of lines in the training set, we train five models, respectively. For each run, we compute the CER on the validation set every 1,000 iterations to determine the best model of this fold. This model is then evaluated on the initial evaluation dataset to obtain the final CER. Since the results of each of the five folds vary, we compute the average to compare the outcomes of the different network architectures, books, and number of lines. Furthermore the five

[8] http://www.ruhr-uni-bochum.de/wegera/ref/index.htm
[9] https://zenodo.org/record/1344132/

$$\mathrm{A}\left\{\begin{array}{ll}\mathrm{n}(0.6) & \mathrm{u}(0.1)\\ \mathrm{n}(0.2) & \mathrm{u}(0.3)\\ \mathrm{n}(0.4) & \mathrm{u}(0.5)\end{array}\right\}\ \text{example sentence}$$

Figure 2: An example for the improvement by using the confidence voting mechanism. Here only three voters are considered. Although the character "u" is the best character twice the "n" is chosen for the final output because of its higher average confidence.

predictions are used to compute a voted result using the voting mechanism described in Section 3.3.

At each iteration during training one line is randomly chosen out of the data fold to compute the gradient update. The number of iterations during training is chosen as 10,000, 12,000, 15,000, 20,000, 25,000, and 30,000 for the training set size of 60, 100, 150, 250, 500, and 1,000, respectively. These values are fixed for both OCRopus and Calamari.

3.3 Voting

In order to further improve the predictions, we implement the confidence voting scheme as proposed by Reul et al. (2018) to vote on the label sequences that are produced by the CTC-Greedy-Decoder for each fold considering the confidence of each predicted character. An example for only three voters is shown in Figure 2.

In a first step, all sentences are aligned as far as possible. Afterwards, all differences are resolved by averaging the confidence of all equal options and keeping the character with the highest value. Naturally, this procedure cannot guarantee a flawless output, but it significantly improves the overall result.

An important prerequisite for the models that are used to vote is that they are similarly performant, but diverse in their predictions. Errors that occur randomly in one or another sentence can easily be corrected, whereas errors that appear in every output cannot be identified. The above Cross-Fold-Training as proposed by Reul et al. (2018) yields different models that can be used for voting although a high amount of training data is shared among each pair of models.

The number of models that are used for voting in our experiments is set to five. Those are the individual models produced by the 5-fold Cross-Validation on each training set. A higher number of voters is expected to yield better results with the drawback to require a higher effort in training and prediction. Experiments showed that a fold size of five is reasonable trade-off.

3.4 Comparison of Calamari and OCRopy

While both Calamari and OCRopy are written in Python and their interfaces are identical (both require images of lines and their respective GT) there are many distinct

differences regarding the implementation, supported features and default settings.

To fasten up the computations of the neural network, OCRopy supports usage of CLSTM[10] which is a C++ implementation of the fixed LSTM network architectures and only runs on the CPU. Calamari however uses Googles TensorFlow library that both allows to design custom network architectures and to utilize a GPU.

Calamari supports the described Cross-Fold-Training mechanism and confidence voting as well as integrated data augmentation. To choose the best model based on the validation data set as provided by the Cross-Folds, Calamari implements early stopping so that an upper limit for the number of training iterations is optional. Thereto, after a certain number of iterations the early stopping algorithms tests whether the current model improves the accuracy on the validation data. If the accuracy has not improved for a given number of models (default 10), the training is stopped. Data augmentation of the training data is provided by a variable factor n_{aug}, i.e. each line is replicated n_{aug} times. Other features are the support of bidirectional text, and an automatic codec adaption if a pretrained model is loaded that shares only a subset of characters.

Calamari employs by default an Adam solver with an initial learning rate of 0.001, whereas OCRopy utilizes a learning rate of 0.0001 and a Momentum solver with a momentum of 0.9. The batch size of Calamari can be chosen arbitrarily, and is 1 by default. OCRopus does not allow to train or predict batchwise. Moreover, Calamari implements an automatic gradient clipping to tackle the exploding gradient problem of LSTMs as proposed by Pascanu et al. (2013).

The standard OCRopus network uses a single hidden LSTM layer with 100 nodes. Calamari extends this shallow structure by introducing two pairs of convolutional (40 and 60 kernels) and pooling layers before the default LSTM-Layer with 200 nodes. Calamari uses a convolutional kernel of size 3×3 with a stride of 1 and equal padding. The network uses max pooling layers with a kernel size and stride of 2×2. A stride or a kernel size of 2 in the first dimension, that is the time dimension, halves the width of the intermediate picture and therefore the number of LSTM operations. On the one hand, this makes the network faster but on the other hand repeated characters might not get resolved, because the CTC-Decoder requires an intermediate blank label. Finally, Calamari adds dropout with a factor of 0.5 to the last layer.

Even though the network architecture of Calamari can be adapted, preliminary experiments showed that Calamari's default network yields competitive results in both accuracy and speed, hence it will be used as the *deep* network in the following. The *shallow* network architecture is the default OCRopus network.

An overview of the differences and default values is listed in Table 2.

4 Experiments

In the following, we present our findings regarding the improvements of the deeper architecture, and the usage of voting. Additionally, we compare the time required for

[10] `https://github.com/tmbdev/clstm`

Table 2: Comparison and default values of OCRopy and Calamari regarding various aspects.

	OCRopy	**Calamari**
Language	Python 2	Python 3
Network Backend	Native/CLSTM	Tensorflow
GPU Support	No	Yes
Default Network Architecture	LSTM 100	CNN 40, Pool, CNN 60, Pool, LSTM 200
CNN Kernel Size	–	3×3
Pool Size	–	2×2
Dropout	No	Yes
Solver	Momentum (0.9)	Adam
Default Learning Rate	0.0001	0.001
Voting	No	Yes
Pretrained Models	Yes (identic codec)	Yes
Data Augmentation	No	Yes

training and prediction of the different architectures, and extend the training corpus to a size of up to 3,000 lines to investigate the behaviour of the deep models on many lines. Finally, we use data augmentation or pretraining to minimize the CER with the focus on only a few lines of GT. All experiments are conducted by using the default hyperparameters and network architectures of Calamari and OCRopy as described in Sec. 3.4.

4.1 Results of the Deep Network

Table 3 shows exemplarily the CER for each of the five folds on the three books for 60, 100 and 1,000 lines. The average of these five models is used to compute the average improvement of the deep network. Note, that in practice the individual results of the model for each fold must be combined e.g. by voting (see Section 3.3) in order to obtain a usable result. Without voting, only one fold could be used to predict a sequence.

The results of the individual folds show no obvious correlation between CERs on the same fold, that is the same training data, and its CERs on different network architectures. For example, the worst fold of the shallow network on book 1488 using 100 lines is the forth fold with a CER of 4.79%. However, the same fold results in the second best model for the deep network. The same can be observed for book 1505 for the second fold: The best outcome of the deep network with CER of 3.02% is the worst fold for the shallow network.

The relative improvements that are shown in Table 3 are listed for all the different number of lines in Table 4. In the left section the relative improvements of the average

Table 3: This table shows exemplarily the improvements of a deep CNN-LSTM compared to the default shallow architecture for 60, 100, and 1,000 lines in the training dataset. Both the individual results of each Cross-Fold plus their average, and the voting improvements are entirely listed. The last columns of the fold average and the voting average state the relative improvements of the deep network architecture. All numbers are given in percent.

60 Lines

		CERs per Fold					Avg.		Conf.	Voted
Book	**Network**	**1**	**2**	**3**	**4**	**5**	**CER**	**Imp.**	**CER**	**Imp.**
1476	Shallow	8.19	8.62	9.23	6.64	7.91	8.12		4.72	
	Deep	6.71	5.48	7.63	5.74	6.28	6.37	21.5	4.63	1.92
1488	Shallow	8.86	6.25	5.87	8.61	6.79	7.28		4.38	
	Deep	5.34	4.63	4.75	4.48	5.31	4.90	32.6	3.84	12.4
1505	Shallow	8.86	6.25	5.87	8.61	6.79	7.28		4.58	
	Deep	4.96	5.21	4.60	4.77	4.56	4.81	33.8	4.02	12.3
Avg.								29.3		8.9

100 Lines

		CERs per Fold					Avg.		Conf.	Voted
Book	**Network**	**1**	**2**	**3**	**4**	**5**	**CER**	**Imp.**	**CER**	**Imp.**
1476	Shallow	8.19	4.96	5.30	8.02	7.64	6.82		3.49	
	Deep	3.94	3.33	3.33	3.03	4.23	3.57	47.6	2.68	23.2
1488	Shallow	4.30	3.72	3.79	4.79	4.38	4.20		2.73	
	Deep	2.79	3.15	3.20	2.97	3.21	3.06	27.0	2.38	12.7
1505	Shallow	4.74	4.67	4.32	4.59	4.40	4.54		3.16	
	Deep	3.13	3.02	3.23	3.22	3.29	3.18	30.0	2.49	21.3
Avg.								34.9		19.1

1,000 Lines

		CERs per Fold					Avg.		Conf.	Voted
Book	**Network**	**1**	**2**	**3**	**4**	**5**	**CER**	**Imp.**	**CER**	**Imp.**
1476	Shallow	1.46	1.52	1.52	1.56	1.68	1.55		0.97	
	Deep	0.74	0.91	0.68	0.71	0.85	0.78	49.7	0.56	42.1
1488	Shallow	1.15	1.20	1.08	1.03	1.38	1.17		0.71	
	Deep	0.54	0.59	0.63	0.60	0.57	0.59	49.7	0.42	40.7
1505	Shallow	1.96	1.87	1.77	1.85	1.77	1.84		1.35	
	Deep	1.34	1.31	1.32	1.31	1.32	1.32	28.4	1.12	17.2
Avg.								43.6		33.3

of the folds is shown, on the right hand side, the improvement when using voting (see Section 4.2). A single deep network architecture yield an increasingly better average CER. For only 60 lines the average improvement is 29%, while for 1,000 lines the best improvement on a single book is 50% and the average over all books is increased by 43%.

A plot of the relative increase dependent on the number of lines is shown in Figure 3 (solid points). The increasing slope is flattening which indicates that more lines used for training a deep model will still yield a better model compared to the default model, but its relative improvement is eventually constant. An even deeper network might increase the relative improvement if more lines are available.

In general, as to expected, the shallow and the deep network yield better results with an increasing number of training data. Surprisingly, although the deep network has a higher amount of trainable parameters which increases the vulnerability for overfitting, it outperforms the shallow net even for a very small amount of training data.

4.2 Evaluation of Voting

The average error rates based on applying the confidence voting algorithm are shown in Table 3. As expected, voting improves the accuracy on all experiments by a significant amount and even reaches a optimum value of 0.42% CER on book 1488. The shallow network benefits by a higher margin compared to the deep network, especially when using only a few training examples. Yet, the deep network of Calamari always outperforms the shallow network of OCRopy before and after voting. The relative improvements shown in Table 4 and Figure 3 clarify this behaviour.

When training on just 60 lines the deep network architecture performs only slightly better than the default network, yielding an average improvement factor of 9%. However, if 1,000 lines are used an average improvement of 33% is obtained. Considering the slopes of Figure 3 it is observed that the improvement gap between voting and non-voting is narrowed from $29\% - 9\% = 20\%$ to $42\% - 33\% = 9\%$ for 60 and 1,000 lines, respectively. Furthermore, it is to be expected that the relative improvement approaches a limit value. Hence, the used deep model is expected to still perform better than the default model by absolute values, but the relative gap appears to be constant. The limit value, i.e. the performance for infinite data, is influenced by the network architecture. Thus, if more lines are available for training more complex models must be considered.

As shown, voting has a higher impact on the default OCRopus network than on the used deep network, especially if only a few lines are used for training. Thus, the individual models must be more diverse to allow for a more effective voting. The evaluation of errors (see Table 5 in Section 4.3) shows that, apart from insertion and deletions of spaces, the errors of deep networks are mostly missing characters, while the errors of the default network are mostly confusions, e.g. e↔c or n↔r in both directions Therefore, voting of deep models that all omit single character predictions (compare Table 5) and thus suffer from similar errors can not benefit by such a high amount compared to the default models whose errors are more random.

Table 4: Relative improvement of the deep CNN compared to default OCRopus listed for all three books and the six variations of the amount of training data. The left and right halves show the relative improvements without and with voting, respectively. All numbers are given in percent.

	Improvement over Folds				Improvement over Voting			
	Book				Book			
Lines	1476	1488	1505	Avg.	1476	1488	1505	Avg.
60	21.5	32.7	33.8	29.3	1.9	12.4	12.3	8.9
100	47.6	27.0	30.0	34.9	23.2	12.7	21.3	19.8
150	38.4	40.1	32.9	37.1	19.7	30.7	19.7	23.4
250	49.7	54.6	30.9	45.0	27.0	34.2	23.5	28.2
500	50.4	49.6	32.1	44.1	35.2	43.7	21.3	33.3
1,000	49.7	49.7	28.4	42.6	42.1	40.7	17.2	33.3

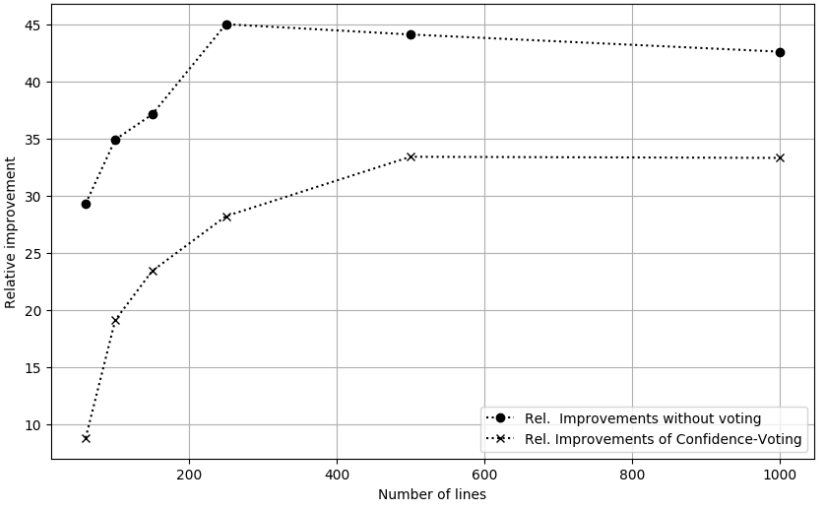

Figure 3: Relative averaged improvements of the deep network versus the default OCRopus network. The solid points indicate the relative improvements based on the averages of the individual Cross-Fold results. The crosses mark the relative improvements when using confidence voting.

Table 5: The 10 most common errors made by the deep network after voting on the evaluation dataset of book 1476. The left and right halves lists the results for 150 and 1,000 training lines, respectively. Both the count (Cnt.) of occurrences and their relative (Rel.) contribution to the total CER are shown. An underscore represents deletions or insertions of characters.

150 Lines				1000 Lines			
Cnt.	**Rel.**	**True**	**Predicted**	**Cnt.**	**Rel.**	**True**	**Predicted**
56	8.9%	SPACE	_	32	18.0%	SPACE	_
54	8.6%	_	SPACE	26	14.6%	_	SPACE
47	7.5%	i	_	11	6.2%	i	_
18	2.9%	l	_	6	3.8%	n	_
12	1.9%	G	E	3	1.7%	r	t
12	1.9%	n	_	2	1.1%	r	v
10	1.6%	r	_	2	1.1%	S	_
9	1.4%	r	t	2	1.1%	r	_
8	1.3%	d	_	2	1.1%	f	f
7	1.1%	o	_	2	1.1%	_	i

4.3 Error Analysis

Table 5 lists the ten most common errors of the deep network after voting when predicting the evaluation dataset of 1476. The results are shown for 150 lines and 1,000 lines on the left and right columns, respectively. For both networks the most common errors consist of insertions or deletions of spaces. Their relative contribution account for one fifth for 150 lines, but almost one third for 1,000 lines, which underlines the difficulty of the task of inserting correct spaces between narrowly printed text if no language model or dictionary is used.

The model trained with 150 lines then mostly misses the prediction of characters which shows the expected behaviour for CTC-trained networks that are only trained with few data: To minimize the CTC-Loss it is more profitable to predict a blank rather than an actual uncertain character. The confusion of G and E vanished because the G occurred rarely in the dataset with only a few lines, but it was successfully learned if more examples were available. The deep network trained with 1,000 lines shows errors among the top ten that are expected, e.g. confusions of f and f (long s).

4.4 Training and Prediction Times

In this section we compare the time required for training and for the prediction of an entire book of the default OCRopus network and our deep network. All times were measured on an Intel Core i7-5820K processor using 1, 2, 4, or 8 threads for each experiment. A NVIDIA Titan X is utilized to gauge the GPU times. During training the parallelism is internally used in Numpy for the default OCRopus implementation and in the TensorFlow-Backend for our deep networks. TensorFlow supports cuDNN

Table 6: Average times for training and prediction of a single line for all three books. Note that during prediction each line has to be processed five times due to the voting of the five folds. The timing procedure was conducted for a various number of threads.

	Training in seconds					**Prediction in seconds**				
Threads	1	2	4	8	GPU	1	2	4	8	GPU
Shallow	0.28	0.27	0.30	0.40	–	0.89	0.48	0.25	0.16	–
Deep	0.57	0.40	0.32	0.33	0.05	0.29	0.21	0.16	0.12	0.03

when using the GPU. For predicting, the default OCRopus implementation copies the individual model for each thread and uses only one thread in the internal operations of Numpy. Calamari instead creates only one model and predicts one batch consisting of 20 lines in our setup using all desired threads.

Note that each line has to be processed five times during the prediction due to the voting algorithm. Table 6 reports our findings for the averages across the three used books.

First of all, the results for the training time show that, despite the deeper network consists of way more parameters and more operations, the optimized TensorFlow implementation is only slightly slower than the default implementation based on Numpy, when only one thread is used. However, the shallow default LSTM net cannot benefit from a higher number of threads, instead it even suffers from a too high count. The reason is, that the underlying matrix multiplications are too low dimensional in order to be relevant for multiprocessing. As expected, the deep network benefits from up to 4 threads, as the training time is decreased by an average factor of approximately 40%. The reason is, that the convolution operations that are carried out on each pixel on the full image profit from a parallel computation. As a result, the deep network is faster than the default implementation when allowing several cores during the training. Our results show, that a number of 4 is sufficient.

The average time required to predict one line for the three books shows that the deep network requires less than half the time compared to the default OCRopus implementation, whereby the time for voting can be neglected. Using a higher number of threads, the required time for prediction almost shrinks linearly for the default implementation, since each thread computes a single independent model by construction. The TensorFlow implementation of Calamari still benefits from a higher thread count, but by a reduced factor due to the core sharing of batch versus convolutional operation. Yet, for all the tested thread counts the TensorFlow implementation is a bit faster than default OCRopus.

Usage of a GPU reduces the training and prediction times further by a factor of at least six and four, respectively. These times can easily be reduced when processing a whole batch of lines instead of a single line.

Table 7: Decrease of the CER for using more than 1,000 lines for training the deep network. All values are given in percent.

	Averaged CER of folds				CER using voting			
	Book				Book			
Lines	**1476**	**1488**	**1505**	**Avg.**	**1476**	**1488**	**1505**	**Avg.**
1,000	0.78	0.59	1.3	0.90	0.56	0.42	1.1	0.70
1,500	0.69	0.50	1.3	0.82	0.48	0.35	1.0	0.62
2,000	0.62	0.49	1.2	0.76	0.45	0.35	1.0	0.60
3,000	–	0.43	–	0.43	–	0.34	–	0.34

4.5 Increasing the Training Data above 1,000 Lines

The available amount of data for the three books allows us to increase the training set size up to 2,000, 3,000, and 2,000 lines for the books 1476, 1488, and 1505, respectively (compare Table 1). The averaged CER of the folds and after voting is shown in Table 7 by usage of the deep network. As expected, even more lines for training reduces the CER even durther in all experiments with and without voting by a significant amount. Thus, we reached an average CER of 0.6% after voting.

4.6 Data Augmentation

In this section we examine the effect of data augmentation with a factor of $n_{\text{aug}} = 10$ on the performances of deep network. In a first step, the training data consists of both the augmentations and the original data. A second step uses the resulting model of the first step as starting point and continues the training on solely the original data.

The results using confidence voting of five different voters each with its own augmentations are shown in Table 8. As expected, especially for only a few lines in the training data set data augmentation has a huge impact of up to 55%. Increasing the number of lines decreases the benefit of augmentations to a point where it even worsens the result (up to -19% on book 1488). This can be explained by the fact that the mixture of real lines and augmented lines forces the network to focus on both types of data which is why it loses its specific training on just the real lines. Thus, when continuing the training on only the real lines (step 2) the results clearly improve for many lines, but even for a few lines. On average, data augmentations in combination with the deep network leads to a CER of below 2% and approximately 1% when using only 60 and 150 lines for training, respectively. The shallow network as provided by OCRopus requires more than 1,000 lines to reach this level of accuracy (compare Table 3).

In summary, the experiments show the expected behaviour that the data augmentation as implemented in Calamari yields slightly improved results for a large data set and very high improvements of up to 55% for only a few lines.

Table 8: Relative improvements of data augmentation with $n_{\text{aug}} = 10$. The first step trains on both the real and augmented data, the second step continues the training only on the real data. The last column shows the absolute CER of the final model (step 2) averaged over all three books.

	Improvement after Step 1				Improvement after Step 2				
	Book				Book				Avg.
Lines	1476	1488	1505	Avg.	1476	1488	1505	Avg.	CER.
60	56.2	60.4	40.2	52.3	61.4	62.1	41.2	54.9	1.87
100	43.8	46.8	28.0	39.5	46.6	52.0	29.6	42.7	1.43
150	43.4	44.6	21.5	36.5	44.8	48.1	26.4	39.7	1.04
250	32.0	22.3	19.7	24.6	37.9	31.4	25.4	31.5	0.83
500	25.9	6.7	14.2	15.6	34.7	23.6	17.1	25.1	0.64
1,000	21.5	9.7	8.7	13.3	22.0	14.6	16.2	17.6	0.58
1,500	12.6	-6.5	9.5	5.2	15.2	3.9	13.9	11.0	0.54
2,000	4.3	-5.8	5.2	1.2	17.0	11.6	11.6	13.4	0.52
3,000	–	-18.5	–	-18.5	–	7.3	–	7.3	0.32

Table 9: Improvement of using both pretraining (PT) and data augmentation (AUG). All values are voted and shown as percentage.

	Improvement after PT				Improvement after PT and AUG				
	Book				Book				Avg.
Lines	1476	1488	1505	Avg.	1476	1488	1505	Avg.	CER.
60	51.1	46.8	30.0	42.7	70.5	68.8	48.0	62.4	1.6
100	37.9	41.9	22.7	34.2	63.6	55.8	39.8	53.1	1.2

4.7 Incorporating Pretrained Models

To evaluate the effect of pretraining, we use three different pretrained Calamari models[11] designed for Fraktur (FRK), historical Latin (LH), and Modern English (EN). Two models of the five voters are trained with FRK, two more with LH and only one with EN since the trained typefaces of FRK and LH are closer to the target font. These outcomes are voted by the confidence voter to obtain the final CER. Furthermore, we combine pretraining with a data augmentation of 10. Both setups are only trained for 60 and 100 lines of GT, because here on the one hand the effect of pretraining and data augmentation is expected to be the largest and on the other hand we want to simulate the lack of manual annotated GT to train a book specific model.

As shown in Table 9, the CER for both 60 and 100 lines benefits from pretraining (left) and the combination with data augmentation (right). We reach an improvement of over 62% for 60 lines of GT compared to the model without pretraining or data

[11] `https://github.com/Calamari-OCR/calamari_models`

Table 10: Improvements of voting, data augmentation, and pretraining (PT) for 60 and 100 lines comparing Calamari to OCRopy. All values are given in percent.

60 Lines					
	Books			**Averages**	
Model	**1476**	**1488**	**1505**	**CER**	**Imp.**
Shallow (OCRopy)	8.1	7.3	7.3	7.6	–
Deep (Calamari)	6.4	4.9	4.8	5.4	28.9
Deep + Voting	4.6	3.8	4.0	4.2	44.7
Deep + Voting + Data aug.	2.0	1.5	2.4	2.0	73.7
Deep + Voting + Data aug. + PT	1.4	1.2	2.1	1.6	78.9
100 Lines					
	Books			**Averages**	
Model	**1476**	**1488**	**1505**	**CER**	**Imp.**
Shallow (OCRopy)	6.8	4.2	4.5	5.2	–
Deep (Calamari)	4.7	4.4	4.6	4.6	11.5
Deep + Voting	2.7	2.4	2.5	2.5	51.9
Deep + Voting + Data aug.	1.5	1.3	1.8	1.5	71.2
Deep + Voting + Data aug. + PT	1.0	1.1	1.5	1.2	76.9

augmentation, with a final CER of 1.6%. An increase to only 100 lines drops the CER to 1.2%.

Comparing pretraining (Table 9) to the results of only data augmentation (Table 8) shows that data augmentation has a higher impact on the performance (e.g. 55% vs 43% using 60 lines). However, data augmentation requires more training time than using pretrained models, because the initial random weights must be trained from scratch.

5 Conclusion and future work

In this paper, we compared the combinations of CNN- and LSTM-Networks as implemented by Calamari to the default LSTM network of OCRopy achieving optimum values considerably below 1% CER and a relative improvement of above 50% compared to standard models. The enhancements are increased by a larger amount of available training data and the introduction of voting mechanisms, data augmentation, and pretrained models. Thus, to train a book specific model, only 60 or 100 lines of GT are sufficient to achieve an average CER of below 2% or about 1%, respectively, as shown in Table 10.

Although the proposed deeper network has a higher number of parameters the absolute training and prediction time is in the same order of magnitude compared to the standard model. However, the usage of a GPU quickens these times by a factor of at least four depending on the lines processed in parallel.

To further improve the voted results, distinct voters are required. These voters could be created by variations of the network architectures and the usage by several different datasets for pretraining.

Moreover, training of an even deeper network using many different books sharing similarities in typeface, is expected to result in a generic model that has very low error rates on a high variety of fonts. To reduce its training time, a GPU combined with batch-wise training should be considered to achieve a high throughput of lines per second.

Recently, the popular OCR engine Tesseract[12] published a new version that implements Deep Neural Networks. We expect similar improvements compared to shallow networks, however voting, data augmentation, and pretraining in the form of Calamari are not supported, yet. Moreover, GPUs can not be used to speed up the training time.

In summary, it can be stated that the application of Calamari implementing deep CNN-LSTM-networks, Cross-Fold-Voting, data augmentation, and pretraining opens the door to a very promising approach to establish a new benchmark for OCR both on early printed books and despite the historical focus of this paper also on any other print.

References

Al Azawi, M., Liwicki, M., and Breuel, T. M. (2015). Combination of multiple aligned recognition outputs using WFST and LSTM. In *Document Analysis and Recognition (ICDAR), 2015 13th Int. Conf. on*, pages 31–35. IEEE.

Bluche, T. (2015). *Deep Neural Networks for Large Vocabulary Handwritten Text Recognition. (Réseaux de Neurones Profonds pour la Reconnaissance de Texte Manucrit à Large Vocabulaire).* PhD thesis, University of Paris-Sud, Orsay, France.

Breuel, T. M. (2017). High performance text recognition using a hybrid convolutional-lstm implementation. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, pages 11–16. IEEE.

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634.

Fan, Y., Lu, X., Li, D., and Liu, Y. (2016). Video-based emotion recognition using cnn-rnn and c3d hybrid networks. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, ICMI 2016, pages 445–450, New York, NY, USA. ACM.

Graves, A. and Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552.

Handley, J. C. (1998). Improving OCR accuracy through combination: A survey. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE Int. Conf. on*, volume 5, pages 4330–4333. IEEE.

[12] `https://github.com/tesseract-ocr/tesseract`

Mane, D. T. and Kulkarni, U. V. (2017). A survey on supervised convolutional neural network and its major applications. *IJRSDA*, 4(3):71–82.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1310–1318. JMLR.org.

Reul, C., Dittrich, M., and Gruner, M. (2017a). Case study of a highly automated layout analysis and ocr of an incunabulum: 'der heiligen leben' (1488). In *Proceedings of the 2Nd Int. Conf. on Digital Access to Textual Cultural Heritage*, DATeCH2017, pages 155–160, New York, NY, USA. ACM.

Reul, C., Springmann, U., Wick, C., and Puppe, F. (2018). Improving OCR accuracy on early printed books by utilizing cross fold training and voting. In *13th IAPR International Workshop on Document Analysis Systems, DAS 2018, Vienna, Austria, April 24-27, 2018*, pages 423–428. IEEE Computer Society.

Reul, C., Wick, C., Springmann, U., and Puppe, F. (2017b). Transfer learning for OCRopus model training on early printed books. *027.7 Zeitschrift für Bibliothekskultur / Journal for Library Culture*, 5(1):38–51.

Rice, S. V., Jenkins, F. R., and Nartker, T. A. (1996). *The fifth annual test of OCR accuracy*. Information Science Research Institute.

Rice, S. V., Kanai, J., and Nartker, T. A. (1994). An algorithm for matching OCR-generated text strings. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(05):1259–1268.

Sainath, T. N., Vinyals, O., Senior, A., and Sak, H. (2015). Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4580–4584.

Shi, B., Bai, X., and Yao, C. (2017). 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.

Springmann, U., Fink, F., and Schulz, K. U. (2016). Automatic quality evaluation and (semi-) automatic improvement of mixed models for ocr on historical documents. *CoRR*.

Springmann, U. and Lüdeling, A. (2017). OCR of historical printings with an application to building diachronic corpora: A case study using the RIDGES herbal corpus. *Digital Humanities Quarterly*, 11(2).

Springmann, U., Reul, C., Dipper, S., and Baiter, J. (2018). GT4HistOCR: Ground Truth for training OCR engines on historical documents in German Fraktur and Early Modern Latin.

Trigeorgis, G., Ringeval, F., Brueckner, R., Marchi, E., Nicolaou, M. A., Schuller, B., and Zafeiriou, S. (2016). Adieu features? end-to-end speech emotion recognition using a deep convolutional recurrent network. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5200–5204.